Composition Adaptative et Vérification Formelle de Logiciel en Informatique Ubiquitaire

Ines Sarray

Stage de Fin d'Études
2013 – 2014

R

Composition adaptative et vérification formelle de logiciel en informatique ubiquitaire

Participant(s) :

- SARRAY, Ines, ines.sarray@esprit.tn , Master2 IFI IAM

Encadrant(s)

- RESSOUCHE, Annie, annie.ressouche@inria.fr
- COLLAVIZZA, Hélène, helen@polytech.unice.fr
- GAFFE, Daniel, daniel.gaffe@unice.fr
- TIGLI, Jean-Yves, tigli@polytech.unice.fr

Dédicaces

A ma mère, mon père, ma sœur et mon frère qui étaient toujours à mes côtés pour m'encourager et me supporter,

A mes amis et à tous ceux qui sont chers à mon cœur,

A ma Tunisie, qui vient de renaître de ces cendres et qui a impressionné le monde entier,

ET à la France, le pays qui m'a bien accueilli et qui m'a offert de très beaux souvenirs

Je dédie ce travail,
Ines.

Remerciements

Je tiens tout d'abord à exprimer ma gratitude à toutes les personnes ayant contribué pour réussir ce stage de fin d'étude et de faire une formation dans un cadre professionnel agréable.

Je remercie mes encadrants madame Annie RESSOUCHE et monsieur Daniel GAFFE , mes encadrants, pour leur confiance et patience, leur aide importante et orientations judicieuses, leurs critiques pertinentes et leurs conseils qui m'ont été bénéfiques tout au long de ce stage.

Je tiens également à remercier monsieur Jean-yves TIGLI et monsieur Stéphane LAVIROTTE pour leur assistance, leurs critiques et leurs encouragements.

Je remercie aussi madame Hélène COLLAVIZZA pour ses remarques et conseils avisés qui m'ont beaucoup servi.

J'exprime ma gratitude en particulier à M. Hassen JEDIDI pour sa confiance, son soutien, et son encouragement.

Je remercie également ma famille qui était toujours là pour moi pour me pousser vers l'avant , et mes amis , spécialement monsieur Ahmed DAOUD qui était toujours à mes côtés pour me soutenir, me supporter et m'encourager ,et mademoiselle Rahma DAIKHI, ma chère collègue, amie et sœur qui m'a accompagnée tout au long de notre parcours scolaire.

J'adresse mes sincères remerciements aux membres de jury pour m'avoir fait l'honneur de bien vouloir accepter d'évaluer mon travail.

Pour finir, j'exprime ma reconnaissance à tous les membres de l'équipe STARS d'INRIA Sophia-Antipolis, pour le climat convivial et chaleureux qu'ils m'ont apporté tout au long de mon stage.

Résumé Exécutif

L'informatique ubiquitaire est un nouveau paradigme pour caractériser l'ensemble des objets intelligents et communicants. Il est utilisé aujourd'hui dans la plupart des domaines et systèmes critiques.

Cette classe d'applications nécessite un contrôle et une vérification permanents de ses applications et composants pour éviter les conséquences dramatiques d'un disfonctionnement.

Un composant critique doit toujours être en écoute des changements de son environnement, en particulier des fonctionnalités mise à disposition par les autres composants et doit vite s'y adapter.

Le but de ce projet est donc de garantir une adaptation automatique et continue à ces changements. Le mécanisme d'adaptation doit à son tour permettre une vérification formelle et une validation.

Abstract

Ubiquitous computing is a new paradigm which characterizes the set of smart and communicating objects. It is used today in most of areas and critical systems.

This class of applications requires a permanent control and verification of these applications and components to avoid the dramatic consequences of a malfunction.

A critical component must always be in touch with changes in their environment, especially the features which are accessible by other components, and must quickly adapt to them.

The purpose of this project is to ensure a continuous and automatic adaptation to these changes. This adaptation mechanism should in turn allow formal verification and validation.

Table des matières

Table des figures

Introduction générale

Introduction générale

L'informatique ubiquitaire est une idée proposée par Mark Weiser qui a vu en elle la possibilité pour les systèmes fondés sur le traitement de l'information d'affranchir le paradigme de l'ordinateur pour se libérer de l'espace qui nous entoure en s'incorporant dans notre vie quotidienne.

Ce nouveau concept a amélioré l'interaction, de ces systèmes entre eux, avec leurs environnements et avec l'être humain.

Cette interaction peut être d'une façon synchrone (en temps réel) ou asynchrone. Plusieurs techniques ont vu le jour pour pouvoir assurer cette interaction, nous trouvons des intergiciels (middlewares), des automates et plein d'autres applications.

Ces systèmes doivent s'adapter d'une façon continue aux changements de leurs environnements. Aussi, la notion de middleware pour concevoir des applications en informatique ubiquitaire est apparue, ce qui a facilité leurs usages dans la plupart des domaines comme la santé, l'aéronautique le transport…. Toutefois certains de ces systèmes peuvent s'avérer critiques.

Les systèmes critiques sont des systèmes dont une panne ou un mauvais fonctionnement peut causer des résultats dangereux voire dramatiques pour les êtres humains, l'environnement, le matériel …

De plus, ces systèmes peuvent supporter des contraintes temporelles fortes : il faut agir au bon moment dans un laps de temps court pour éviter de mauvaises conséquences.

Pour faire face à cette double contrainte (système temps réels et critiques), le modèle synchrone a été introduit. C'est un modèle du temps qui permet de modéliser des systèmes temps réels ainsi que d'appliquer des techniques de vérification formelle comme le model checking afin de valider le comportement des systèmes critiques.

Le model cheking consiste en « *un groupe de techniques de vérification automatique des systèmes dynamiques* ».Il s'agit de vérifier si un modèle répond aux contraintes et spécifications imposées par ces systèmes critiques qui sont souvent présentés en termes de logique temporelle.

C'est dans ce cadre que s'inscrit mon projet de fin d'études. En effet, l'objectif de mon travail consiste à définir le comportement de composants logiciels reliés à des composants critiques et vérifier ce comportement dans un middleware adaptatif, à base de composants (WComp). De plus, si la durée de mon stage le permet, je vais également étudier la composition de ces composants logiciels en cas d'accès multiple à un composant critique.

Deux grands chapitres structurent notre présent rapport:

Le premier est consacré au contexte du projet et l'état de l'art : dans ce chapitre nous allons mieux étudier l'informatique ubiquitaire, les middlewares, les systèmes temps réel, les systèmes synchrones et asynchrones et d'autres technologies qui facilitent mieux la compréhension du sujet.

Le deuxième chapitre sera consacré à l'explication de ce qui a été fait et ce qui nous reste à faire, nous présenterons aussi dans ce chapitre les outils que nous avons utilisé, nous expliquerons son apport et parlons des difficultés rencontrés.

Enfin, nous retiendrons dans la conclusion générale les grandes lignes décrivant notre projet et nous exposerons ensuite les extensions possibles du sujet.

Chapitre1 :

Organisme d'accueil

1.1 INRIA

L'Institut National de Recherche en Informatique et Automatique(INRIA) [10] est un établissement public français à caractère scientifique et technologique (EPST), qui a été créé à Rocquencourt en 1967, et qui s'est étendu depuis cette date pour être désormais composé de huit sièges répartis sur tout le territoire français (Rocquencourt, Bordeaux, Grenoble, Lille, Nancy, Rennes, Saclay, et Sophia Antipolis).

L'INRIA est reconnu à l'échelle internationale, son but est de produire une recherche d'excellence en Mathématiques des Sciences du Numérique, et en Informatique et d'assurer son impact.

Aujourd'hui plus de 4000 personnes de différentes nationalités travaillent à l'INRIA, dont 3450 scientifiques (1678 chercheurs de l'institut et 1778 universitaires ou chercheurs d'autres organismes) qui sont regroupés en 172 équipes de recherche réparties dans les huit centres de recherche.

Les thèmes de recherche de ces équipes s'articulent autour de cinq domaines principaux qui sont :

1- Mathématiques appliquées, calcul et simulation

2- Algorithmique, programmation, logiciels et architectures

3- Réseaux, systèmes et services, calcul distribué

4- Perception, Cognition, Interaction

5- Santé, biologie et planète numérique

1.2 INRIA Sophia Antipolis

INRIA Sophia Antipolis est le troisième site créé de l'INRIA (après Rocquencourt et Rennes). Il est présent sur la technopole de Sophia Antipolis depuis 1981.

Plus de 600 personnes travaillent dans ce centre de recherche, et au sein de 32 équipes (dont STARS).

Ces équipes poursuivent un bon engagement avec plusieurs acteurs académiques et économiques (des établissements de recherche, des associations, des entreprises …), et s'impliquent dans des réseaux de recherche différents, en s'appuyant sur la qualité de ses chercheurs et ses services, afin d'ajouter de nouvelles idées innovantes qui enrichissent les domaines de recherche technologiques diversifiés tels que les mathématiques appliquées, les langages de programmation, l'algorithmique, les réseaux et systèmes distribués….. C'est un catalyseur économique qui offre de bonnes perspectives d'emplois dans ces domaines.

1.3 Equipe-projet d'accueil: STARS

L'équipe de recherche STARS (Spatio-Temporal Activity Recognition Systems) se concentre sur deux domaines d'application : la vidéosurveillance et le maintien des personnes âgées à domicile. Elle se focalise sur la conception et le développement des systèmes cognitifs pour la reconnaissance d'activités.

Cette équipe étudie les activités spatio-temporelles à long terme des êtres humains, des véhicules ou des animaux. Cette étude se fait à l'aide de l'interprétation sémantique et en temps réel de plusieurs scènes dynamiques surveillées par des capteurs et des caméras vidéo.

STARS se focalise sur deux axes de recherche principaux qui sont :

1- **L'interprétation de scènes pour la reconnaissance d'activités :**

Cette interprétation a pour but de trouver une solution pour tout le problème d'interprétation, de l'analyse bas-niveau du signal jusqu'à la description sémantique du contenu de la scène contrôlée par les capteurs et/ou les caméras vidéo.

Plus précisément, STARS travaille en perception, interprétation et apprentissage.

2- **L'architecture logicielle pour la reconnaissance d'activités :**

Ceci consiste à étudier les systèmes génériques pour la reconnaissance d'activités, et à élaborer des méthodologies de conception de ces systèmes, en assurant la généricité, la modularité, l'extensibilité, la réutilisabilité, la fiabilité et la maintenabilité.

Chapitre2 :

Contexte général

2.1 Introduction

Le projet étant fixé, il faut réaliser une étude complète de l'existant pour mieux éclaircir les idées relatives aux différents concepts du projet. Dans ce chapitre, nous allons présenter le contexte de notre projet, sans oublier l'état de l'art qui nous est indispensable pour mieux comprendre notre problématique.

2.2 Description du projet

WCOMP est un middleware adaptatif à base de composants qui assure une liaison et communication asynchrone entre eux. Afin de vérifier le comportement des composants critiques de WComp, un moniteur synchrone décrivant le comportement de ce composant sera introduit dans l'application. Ce dernier sera intercalé entre les composants reliés au composant critique d'une part et au composant critique lui-même. Ce moniteur est un modèle synchrone sur lequel on peut appliquer des méthodes de model checking afin de vérifier formellement des propriétés de sureté de fonctionnement du composant critique (voir figure1).

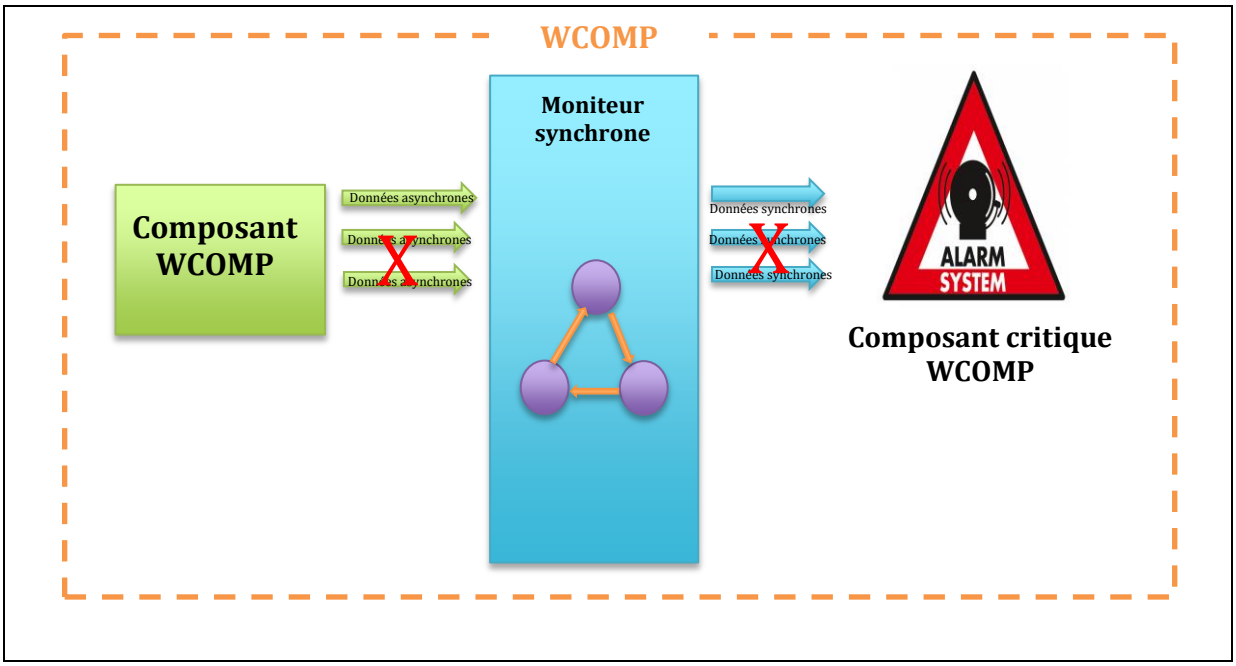

Figure 1: Problème de communication synchrone/asynchrone dans WCOMP

Le problème qui se pose ici, est que notre moniteur synchrone fonctionne d'une façon synchrone, c'est-à-dire qu'il n'accepte que des données synchrones en entrées et n'émet que des données synchrones en sortie, il ne peut pas traiter des données asynchrones.
De l'autre côté, WCOMP est un environnement de communication asynchrone, d'où, ses composants ne peuvent traiter ou émettre que des données asynchrones (voir figure1).

Le but de ce projet est aussi de garantir une adaptation automatique et continue aux changements des environnements des systèmes critiques d'une façon synchrone, en créant une machine d'exécution ou un transformateur asynchrone/synchrone qui va regrouper les données asynchrones en des instants synchrones pour les envoyer au moniteur synchrone. Ce moniteur

synchrone contient un automate qui permet une vérification formelle et une validation de ces informations avant de l'envoyer à une deuxième partie de la machine d'exécution qui est le désynchronisateur qui va replonger ces données dans le monde de l'asynchrone (voir figure2).

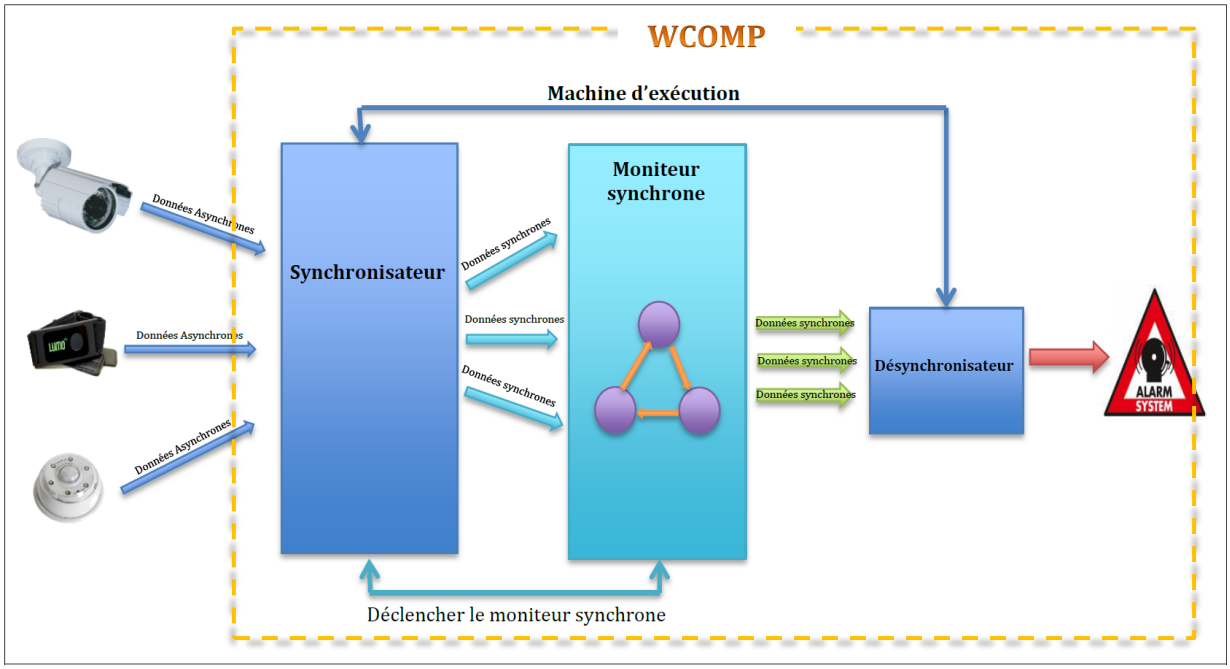

Figure 2 : Objectif du stage

De plus un composant critique peut avoir plusieurs moniteurs synchrones donc plusieurs accès à une seule entrée.

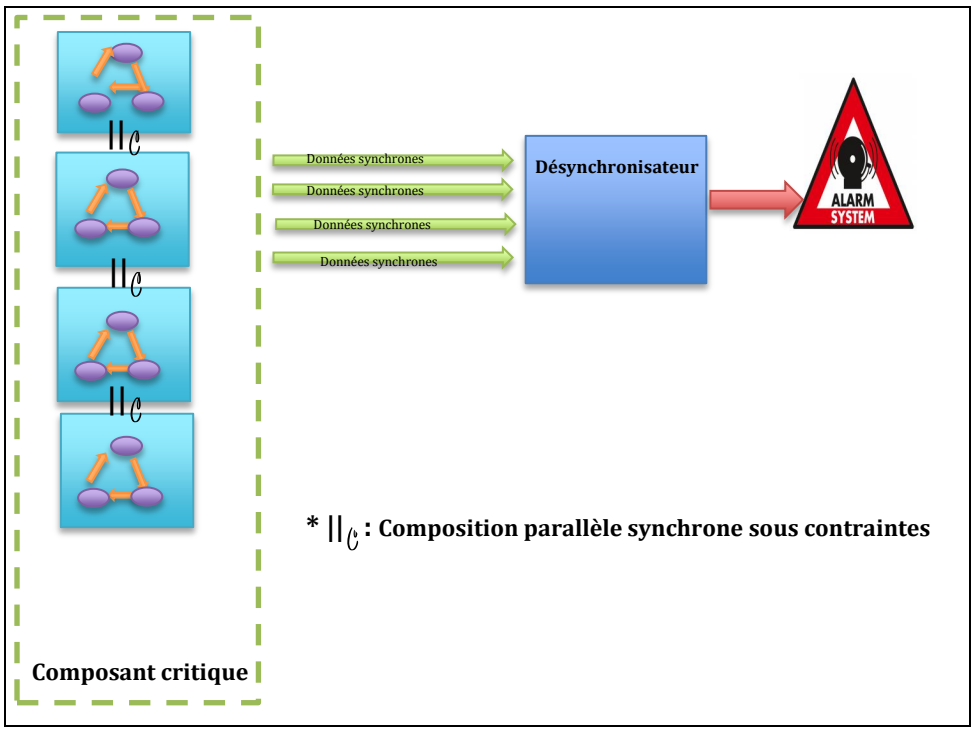

Figure 3 : principe de la composition synchrone sous contraintes

Une composition synchrone sous contrainte est composée du parallèle synchrone et d'une fonction de contrainte qui s'applique sur les transitions de l'automate résultant de la composition synchrone.
Elle peut être une solution viable pour regrouper les entrées, mais l'inconvénient est que cette solution n'est pas incrémentale, donc elle n'est pas adaptative (voir figure3).

Nous verrons dans le chapitre travaux et réalisation l'approche que nous proposons pour pallier ces problèmes.

2.3 Etat de l'art

2.3.1 L'informatique ubiquitaire

"Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives."
--Mark Weiser

"*Ubiquitous computing is roughly the opposite of virtual reality. Where virtual reality puts people inside a computer-generated world, ubiquitous computing forces the computer to live out here in the world with people. Virtual reality is primarily a horse power problem; ubiquitous computing is a very difficult integration of human factors, computer science, engineering, and social sciences."*
--*http://www.ubiq.com/*

L'ubiquité [1][13] désigne le fait de pouvoir être présent partout et dans des lieux différents en même moment. L'environnement ubiquitaire présente un groupe de dispositifs dotés d'une capacité de calcul et de stockage, qui nous entourent et qui sont capables de communiquer avec nous, qui sont toujours présents et qui nous offrent des services accessibles n'importe où et n'importe quand.

L'informatique ubiquitaire présente une nouvelle façon de penser notre relation avec les ordinateurs dans un environnement dédié. Mark Weiser, le père fondateur de ce nouveau concept voyait un nouveau monde, dans lequel les gens peuvent interagir facilement avec les ordinateurs n'importe où et à n'importe quel moment, il a voulu « dissoudre » ces ordinateur dans notre vie quotidienne et de les confondre avec elle.
Ceci a rendu aujourd'hui l'usage de la technologie beaucoup plus aisé et pratique, tout en assurant une utilisation plus fluide dans tous les domaines et a permis aux machines d'éviter d'accaparer notre attention.

Les technologies ubiquitaire qui existent aujourd'hui sont souvent sans fil, mobile et situées dans un réseau. Chaque technologie a ses propres caractéristiques propres à elles, ce qui impose parfois des problèmes de communication avec les autres technologies qui ne possèdent pas les mêmes caractéristiques; d'où la création des middlewares (ou intergiciels) pour résoudre ce problème et pour faciliter la communication entre les composants du réseau.

A. Les middlewares (intergiciels)

Un intergiciel ou middleware [2] présente une couche logicielle du « milieu » qui permet la communication et l'échange des informations entre des applications informatiques de différents types.

Les middlewares sont basés sur plusieurs concepts importants tels que :

- ✓ **L'adaptation** : qui signifie que l'intergiciel doit être capable de réagir aux changements de contexte, nous parlons ici de deux type d'adaptation :
  - l'adaptation structurelle : qui consiste à la modification d'un assemblage de composants, sans modification de son mode de fonctionnement
  - l'adaptation comportementale : qui consiste à modifier la façon qu'ont les composants de s'exécuter.
- ✓ **L'Hétérogénéité :** qui consiste à l'aptitude de fonctionner avec différents matériels, logiciels, systèmes d'exploitation et mêmes protocoles**.**
- ✓ **L'Extensibilité** : qui signifie que nous pouvons ajouter de nouvelles fonctionnalités.
- ✓ **L'Evolutivité:** qui consiste à la capacité de ces systèmes à être amélioré et étendu.

Aujourd'hui, il existe une multitude de middleware pour l'informatique ubiquitaire qui permet la communication et la coordination entre plusieurs composants, tels que :

- ✓ **Gaia :** Gaia est une plateforme middleware pour les espaces actifs, qui fait la coordination entre les entités logicielles et périphériques qui sont présents dans un réseau hétérogène d'un espace physique. Ce middleware fournit un support pour les applications spatiales mobiles qui sont actives et centrées sur l'utilisateur, en gérant leurs ressources et services. Gaia offre aussi des services pour la localisation, le contexte, les évènements et les référentiels de l'espace actif.
- ✓ **ExORB :** c'est un middleware pour les téléphones mobiles. Il a été inspiré du DPRS (Dynamically Programmable and Reconfigurable Software). Il emploie une nouvelle technique appelée « externalisation » qui permet de décrire l'état du middleware, sa logique et son architecture de composants, pour permettre aux concepteurs des dispositifs de reconfigurer leurs logiciels, de les mettre à jours et de les débuguer facilement et sans intervention manuelle.
- ✓ **CORTEX :** C'est un intergiciel qui a été créé pour les environnements pervasifs (ubiquitaire) et ad-hoc qui exigent une base de calcul de temps pour le traitement des interactions temps réel. Ce middleware utilise la technologie des composants et de réflexion. Il fournit de nombreux mécanismes efficaces pour la sensibilité au contexte et pour la prise de décisions intelligentes.

  CORTEX modifie le comportement des objets pour modifier les évènements. Ce middleware ne traite pas la sécurité mais il offre une très bonne qualité de service.
- ✓ **WCOMP :** c'est une plate-forme pour les composants légers destinés à développer de nouvelles applications ambiantes, en regroupant les composants logiciels et gérant l'accès à leurs services. Il supporte de nombreux protocoles tels que UPnP (Universal Plug and Play).

Nous allons utiliser dans notre projet le middleware WCOMP, et nous allons présenter maintenant cet intergiciel.

B. WCOMP

WCOMP [2][3] est un intergiciel (middleware) développé en .NET par l'équipe « Rainbow » du laboratoire I3S.
Il est utilisé dans l'informatique ubiquitaire. Il se base sur trois paradigmes importants qui sont :

a. Les services basés sur les évènements :

Ceux sont des services qui communiquent entre eux en se basant sur les évènements.
Nous avons deux types de services : les **services composites** qui peuvent invoquer d'autres services composites ou services simples. Ils présentent deux interfaces : **une interface fonctionnelle** qui permet la communication avec les fonctionnalités offertes par le service, et une **interface structurelle** qui permet d'accéder à un assemblage de composants et de le modifier.
Les services composites présentent généralement des web services pour dispositifs comme UPnP (Universal Plug and Play) et DPWS (Device Profile For Web Services).
L'autre type de services s'appelle les **services basiques** qui sont des services indépendants et ne font pas appel à d'autres services.
Dans WCOMP nous nous intéressons aux services composites.

b. Les composants légers à l'intérieur des web services :

Nous parlons ici du modèle SLCA ou « Service Lightweight Component Architecture » qui utilise trois concepts importants qui sont les services, les composants légers et les évènements.
Dans ce modèle, les services composites se basent sur un groupe interne de composants légers pour pouvoir gérer la composition entre les autres web services basée sur les évènements et de concevoir l'interface d'un nouveau service composite.
L'ensemble interne de ces composants supportent la réactivité et la haute dynamicité du modèle, en utilisant la communication basée sur les évènements et en fournissant un moyen permettant d'être structurellement modifié et adapté.

c. Le paradigme d'adaptation basé sur l'aspect d'assemblage (Aspect of Assembly)

Ce concept permet de créer de nouveaux régimes d'adaptations indépendants et qui peuvent faire face aux problèmes de la séparation des intérêts, et peuvent être introduits et appliqués dans n'importe quel web service composite de l'application.
Ces aspects assurent une adaptation structurelle du modèle, vu que la modification n'est faite que dans l'ensemble des composants internes des services composites
L'adaptation basée sur l'aspect d'assemblage (AA) est conçue pour modifier les web services basées sur les évènements en prenant en considération l'apparition et la disparition des dispositifs dans leur environnement (évolution de l'infrastructure).

L'architecture de WCOMP est basée sur deux composants qui sont **le container** qui gèrent les assemblages de composants légers et qui représentent les services composites, et **le designer** qui manipule les applications via le container.

➡ La majorité des intergiciels utilisent un modèle asynchrone pour établir la communication entre leurs composants.

C. Les systèmes asynchrones

Les systèmes asynchrones [5][6] sont des systèmes faiblement couplés et qui n'utilisent pas de signal d'horloge pour gérer leurs entrées. Ils gèrent les communications d'une façon locale, à l'aide de la synchronisation entre les blocs fonctionnels. Ils mémorisent les données et peuvent fonctionner dans un mode déconnecté. L'état des communications peut par la suite être modifié à n'importe quel instant.

Les systèmes asynchrones se basent sur un concept très important qui est le contrôle distribué, ce principe consiste au fait que chaque unité ne peut se synchroniser qu'avec les blocs avec lesquels cette synchronisation à un résultat fonctionnel, indépendamment de son environnement et ce uniquement au moment où cette unité doit fonctionner.

Il existe plusieurs langages de programmation asynchrone comme Electre qui mémorisent les signaux d'entrée en attendant leur traitement, SDL…

2.3.2 Les systèmes temps réel

"*En informatique temps réel, le comportement correct d'un système dépend, non seulement des résultats logiques des traitements, mais aussi du temps auquel les résultats sont produits*" [STA 88].

Les systèmes temps réel [4] sont des systèmes assez souvent réactifs c'est-à-dire qui dépendent de leurs environnements et sont en interaction permanente avec eux. Ils doivent répondre aux stimuli de ces dernier en respectant certaines contraintes dont la plus importantes est la contrainte temporelle. En effet, Les systèmes temps réel sont des systèmes dont on peut évaluer ou borner le temps de réaction, et leurs réactions aux stimuli peuvent avoir un effet sur l'environnement.

Il existe trois catégories de systèmes temps réel :

- ✓ **Les systèmes temps réel durs ou critiques (Hard Real Time) :**

Ce sont les systèmes dont la réponse est vitale, elle doit être exacte ou dans un lapse de temps considéré. L'absence de la réponse ou son retard aura des conséquences catastrophiques (voir figure 4).

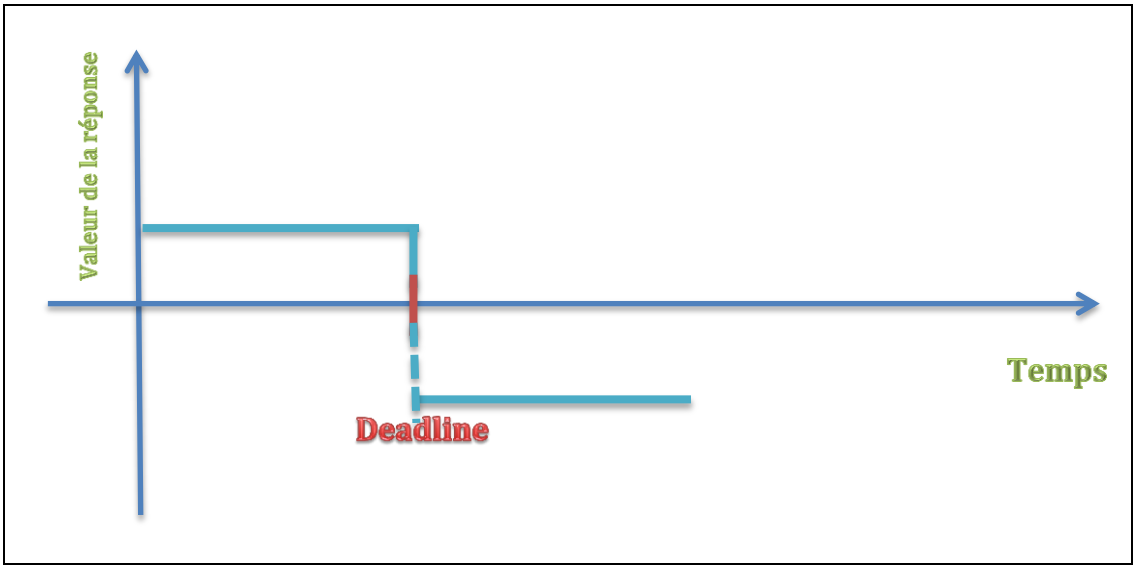

Figure 4: Hard Real Time

Ce type de systèmes critiques est présent dans plusieurs domaines comme le domaine de l'aéronautique ou le transport.

- ✓ **Les systèmes temps réel mous ou souples (Soft Real Time)**

La réponse tardive du système n'a pas d'effets dangereux, mais elle perd son intérêt au fur et à mesure qu'elle dépasse sa deadline (voir figure 5).

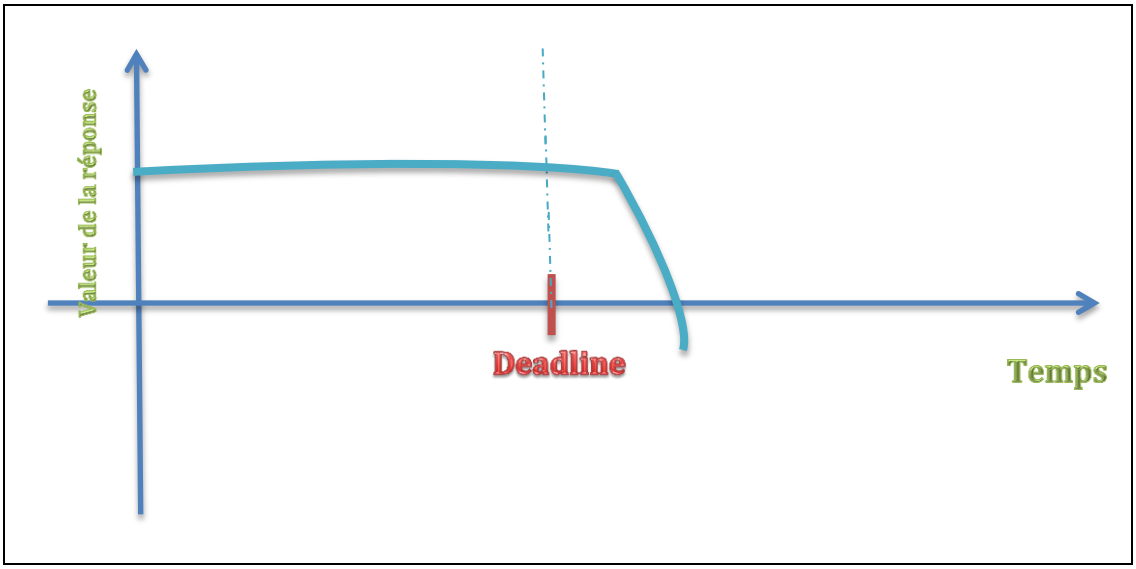

Figure 5: Soft Real Time

Ce système critique est utilisé dans plusieurs domaines des téléphones mobiles ou autres.

✓ **Les systèmes temps réel ouverts ou fermes (Firm Real Time)**

Le système doit essentiellement répondre aux stimuli de l'environnement et il ne faut pas dépasser le deadline, sinon, cette réponse sera inutile (voir figure 6).

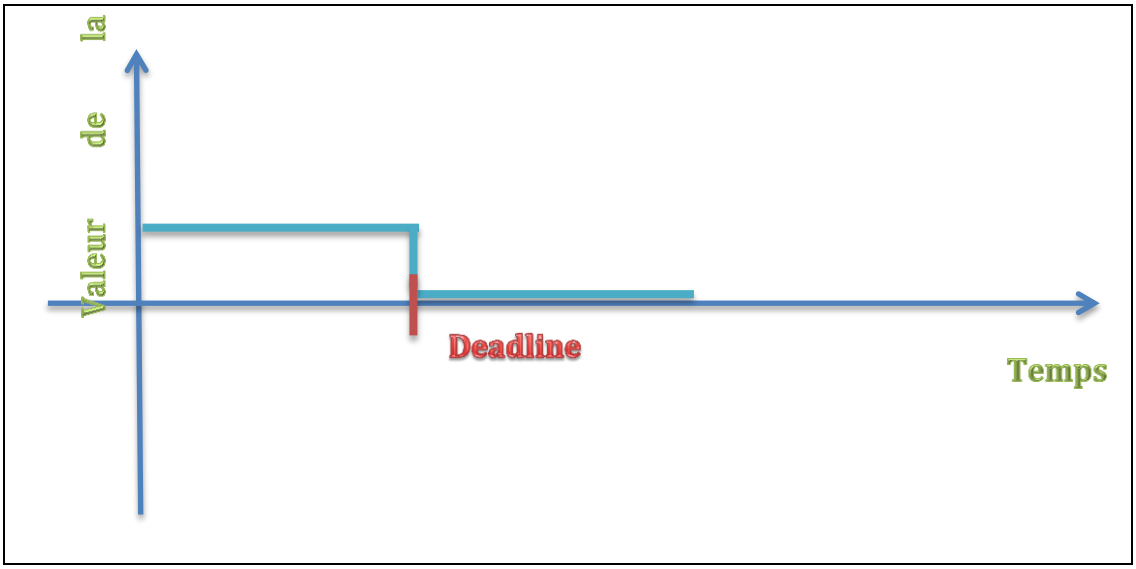

Figure 6: Firm Real Time

Ce type de systèmes critiques est présent dans plusieurs domaines comme le domaine de la bourse.

2.3.3 Les systèmes synchrones

Les systèmes synchrones [5][11] sont des systèmes qui évoluent sur instant logique, dialoguent et s'exécutent simultanément (voir figure 7).

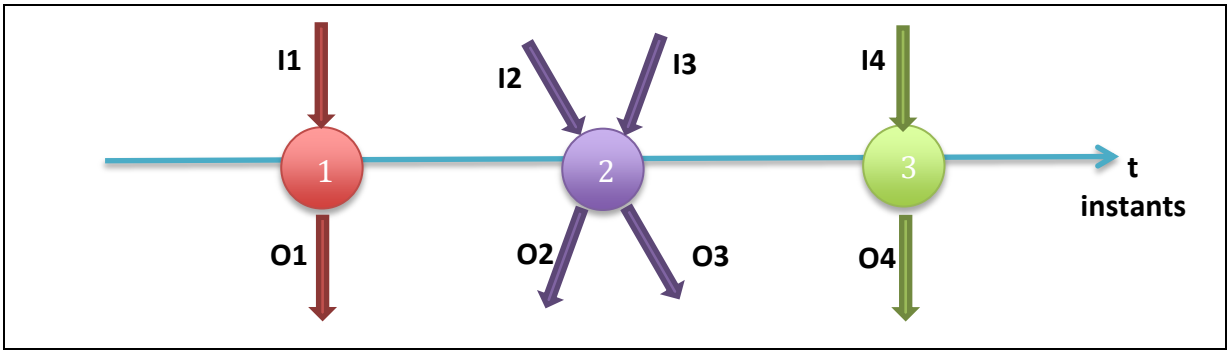

Figure 7: principe du système synchrone

Les systèmes synchrones mettent en évidence trois concepts importants que sont le parallélisme, la communication et le déterminisme.
Les systèmes synchrones permettent de définir des systèmes réactifs embarqués très sûrs et très complexes grâce à l'aide de plusieurs langages de programmation synchrones dont les plus

connus sont : ESTEREL, QUARTZ, LUSTRE (SCADE), SIGNAL. Ces langages synchrones sont basés sur plusieurs hypothèses (qui constitue le paradigme ou hypothèse synchrones) telles que :

1- **Les signaux** qui doivent abstraire la communication.
2- **L'échantillonnage parfait** qui donne une vue « vectorielle » de l'environnement
3- **Le temps logique** qui est opposé au temps physique et qui donne une importance à la notion d'instant et d'évènement.
4- **La durée nulle** qui signifie que les signaux produits lors d'une réaction sont simultanés avec ceux qui ont déclenchés cette dernière.
5- **La perception globale** qui nécessite une communication instantanée des informations pour obtenir un programme cohérent.

Ces langages synchrones permettent de décrire des modèles synchrones qui sont des automates à états finis et qui sont des modèles d'entrée pour les outils de model checking. Une autre façon naturelle de décrire de tels modèles est d'utiliser le formalisme des FSM : ceux sont des automates à états finis où les transitions sont étiquetées par des couples trigger/action. Le « trigger » déclenche la transition tandis que « l'action »est exécutée en réaction à ce déclenchement.

A. Le Langage ESTEREL

Le langage ESTEREL est parmi les langages importants dans la programmation réactive synchrone. C'est un langage impératif qui présente des instructions impératives (composition parallèle, séquence..) et des instructions réactives (pause, attente d'évènements..) et qui peut gérer plusieurs modules en parallèle. La communication entre modules est basée sur la diffusion synchrone de signaux.

B. Le Langage LUSTRE (SCADE)

Lustre est un langage de programmation synchrone des systèmes réactifs. Il est déclaratif et il manipule des flots de données. Il est utilisé pour la conception des logiciels critiques dans l'environnement de développement SCADE (Safety Critical Application Development).
Un programme lustre est basé sur un réseau d'opérateurs qui consomment et produisent des flots de données à chaque cycle d'activation.
Son domaine d'application immédiat est la modélisation et l'implémentation des filtres numériques.

C. Le Langage Light ESTEREL

Light Esterel est un nouveau langage synchrone qui est dédié à la spécification, la mise en œuvre et la vérification des systèmes de contrôle réactif. Trois syntaxes sont supportées (voir figure 8) :
1/ Représentation sous forme d'automates hiérarchiques.
2/ D'une syntaxe déclaratives proche de Lustre
3/D'une syntaxe impérative inspirée du langage Esterel V5

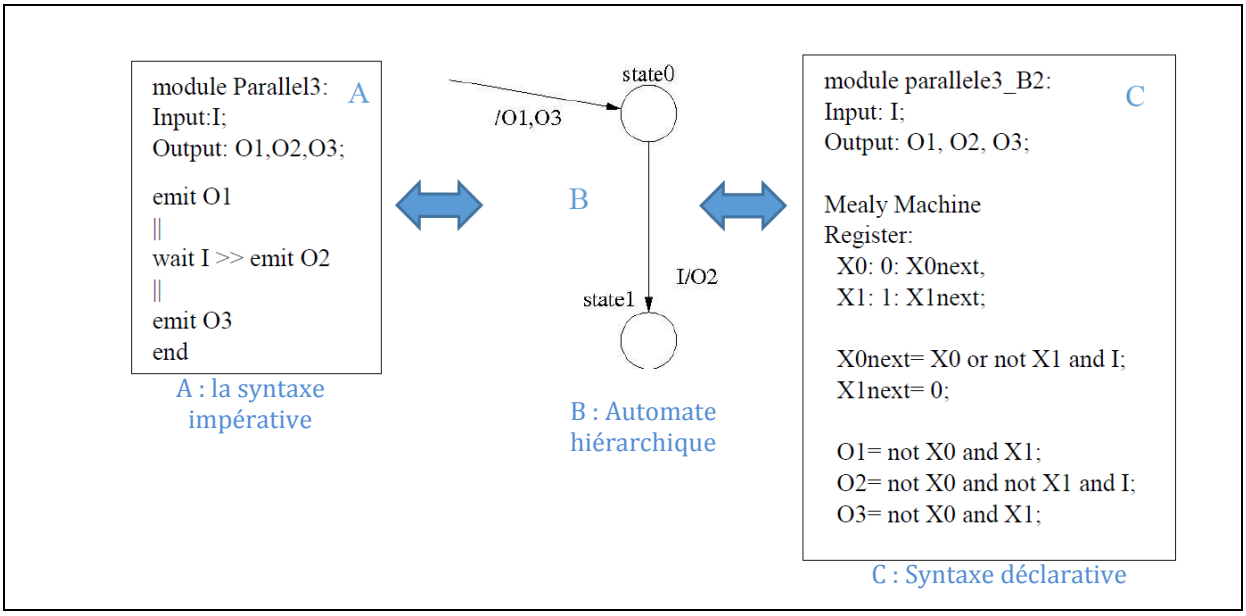

Figure 8: Les syntaxes supportées par Light Esterel

En particulier, les automates sont des constructions natives du langage LE, ce dernier est capable de les compiler et d'intégrer leurs instances dans d'autres modules déjà compilés.
Le langage LE suit les principes du Model Driven Software Developement qui est aujourd'hui bien connu en tant qu'un moyen pour gérer la complexité et atteindre de hauts niveaux de réutilisation et réduire l'effort de développement.
En conséquence, LE présente un cadre formel qui est bien adapté à la compilation et la validation formelle.

Dans la pratique, le langage LE présente deux sémantiques :

a. **Une sémantique du comportement** qui définit un programme à l'aide par l'ensemble de ses comportements, en évitant les ambiguïtés de son interprétation.
b. **Une sémantique équationnelle** qui permet la compilation modulaire du programme pour les cibles matérielles et logicielles (la synthèse FPGA, SystemC, C, VHDL….)

On peut utiliser l'outil CLEM (Compiler of Light Esterel Modules) pour compiler les programmes Light Esterel et Clef (CLEm Finaliser) pour générer leurs codes.

2.3.4 Le projet CLEM

CLEM [7][8] est une boite à outils développée par l'équipe STARS et l'équipe LEAT (Laboratoire d'Electronique, Antennes et Télécommunications), et conçue autour du langage synchrone LE (Light Esterel). Cette boite à outils permet à la fois de concevoir, simuler, vérifier et générer du code pour différents programmes et vers différentes cibles.
CLEM répond aux deux exigences principales pour traiter les applications critiques temps réel :

1- Il permet la compilation modulaire pour pouvoir gérer les grands systèmes.
2- Il se base sur un modèle d'états pour assurer la validation formelle.

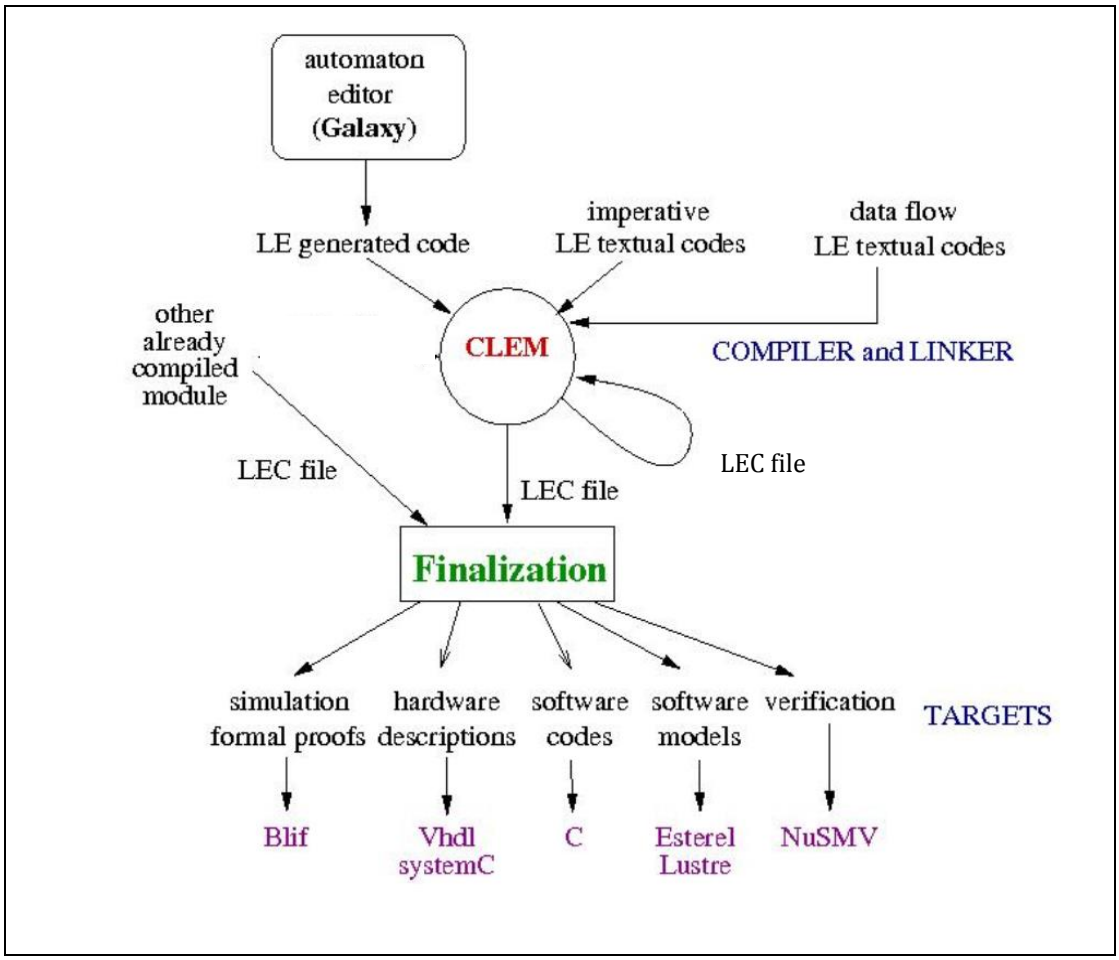

Figure 9: Composition et fonctionnement de CLEM

Le fonctionnement de CLEM se déroule en trois phases principales (voir figure 9) :

A. La conception :

La conception désigne la description des applications en s'appuyant sur le langage LE. L'unité de ce langage est un module nommé dont l'interface déclare les évènements d'entrées qu'il utilise et les évènements de sortie qu'il génère.

Comme nous l'avons cité précédemment, LE nous offre trois formes de conception :

- **Graphique :** sous la forme d'automates hiérarchiques.
- **Textuelle déclarative :** en utilisant une syntaxe déclaratives proche de Lustre.
- **Textuelle impérative :** en utilisant une syntaxe impérative inspirée du langage Esterel V5

B. La compilation :

Le compilateur de CLEM s'appuie sur la sémantique équationnelle constructive du langage LE, cette sémantique permet la compilation modulaire. Ces règles sémantiques donnent une traduction du modèle en un système d'équations quadri-valuées, qui va permettre de calculer les statuts des signaux de sortie et locaux à partir des statuts des signaux d'éntrée dans une algèbre à quatre valeurs : (absent, présent, bottom, erreur).

Intuitivement, absent et présent représentent respectivement le statut d'un signal absent ou présent dans l'environnement. Bottom représente le statut d'un signal pour lequel il n'y a pas d'information (indéfini).

Quand à erreur, il représente la notion de « sur-connaissance » vu que le signal est à la fois présent et absent.
Un système d'équation sera associé à chaque programme. Puis, ce système d'équation sera transformé en un autre système d'équation booléenne (on transforme chaque système d'équation en deux équations booléenne).
Pour pouvoir générer du code, simuler ou faire la liaison avec un code externe, il faut trouver un ordre d'évaluation qui sera valide pour tous les instants synchrones. Cet ordre d'évaluation est total dans la plupart des langages synchrones existants. Tout ordre total ne permet pas de faire la compilation séparée.
CLEM s'appuie sur un algorithme inspiré de la Méthode CPM (Critical Path Method) pour calculer tous les ordres partiels d'un système d'équation. On peut ainsi composer deux systèmes d'équations ordonnées et en déduire un système ordonné (par rapport à ces ordres partiels).
Afin de d'effectuer la compilation séparée des programmes LE, CLEM utilise un format de compilation interne appelée LEC (Light Esterel Compilation). Il permet la représentation des équations booléennes et la réutilisation du code qui a été déjà compilé.

C. La finalisation

La finalisation est la phase finale de la compilation des programmes LE. Dans cette phase, les signaux quadri-valuées sont projetés sur uniquement deux valeurs (absent,present). Tous les signaux d'entrée avec l'état bottom sont vus comme absents. Ce prossesus de finalisation est **irréversible**, il n'est présent que dans la phase finale de la compilation, lors de la génération des codes cibles. La phase de finalisation est implémentée dans l'outil CLEF. Cet outils permet aussi des générer le code pour les différentes cibles matérielles et logicielles.

D. La simulation et la vérification

Cette phase consiste à simuler les programmes LE compilés. Pour effectuer la simulation, on traduit les systèmes d'équations booléennes finalisés au format blif, puis on utilise l'outil Blif_Simul pour faire la simulation de ces programmes, ce dernier permet d'afficher graphiquement les valeurs et le comportement des signaux.
Le format blif permet également d'interfacer des outils de model-checking développés à l'Inria (Blif_check et Xeve). Par ailleurs, CLEM génère également du code pour le model checker NuSMV.

E. La génération

Pour pouvoir exécuter des applications dans des langages différents, CLEM nous permet plusieurs formats de sortie :

- Pour les modèles logiciels : lustre, esterel
- Pour la description hardware : VHDL, SystemC
- Code de programmation logicielle : code C

2.3.6 Le model checking

Le model checking [9][14] est un ensemble de techniques pour la vérification automatique des propriétés des systèmes réactifs dynamiques (il s'agit souvent des systèmes d'origine informatique ou électronique). Il est très important surtout pour les systèmes critiques pour vérifier leur sureté de fonctionnement, afin de s'assurer qu'ils ne posent aucun problème critique qui peut avoir des conséquences dangereuses.

Le model checking est généralement réservé à des systèmes finis, ou des systèmes qui ont une abstraction finie facilement car les techniques standards du model checking sont limitées. En effet, le programme ne doit généralement manipuler que des variables de domaines finis.

Les modèle checking s'appuie sur 3 points :

1- Un modèle du système qui représente tous les comportements du système (sous la forme d'un automate à états finis)
2- La possibilité d'exprimer des propriétés : ceux sont les logiques temporelles. On peut avoir deux classes de logique temporelles :
    a. Logique Temporelle Linéaire(LTL) : qui expriment la vivacité du système.
    b. Logique Temporelle Arborescente(CTL) : qui expriment la sureté et la sécurité du système.
3- Un algorithme pour la vérification de ces propriétés dans le modèle : nous avons deux types d'algorithmes :
    a. L'algorithme LTL : C'est un algorithme global récursif sur les formules du modèle
    b. L'algorithme CTL : C'est un algorithme local récursif sur la structure du modèle.

Le plus grand avantage du model checking est sa décidabilité, il peut décider à l'aide de ces algorithmes de calcul si le modèle vérifie ou non le modèles de propriétés.

Il existe plusieurs outils que nous pouvons utiliser pour le model checking, tels que : SCADE, Bandera, nuSMV...

NuSMV est le model checking utilisé par CLEM pour effectuer la vérification des modèles des systèmes. C'est une ré-implémentation et extension de SMV (un model checker), c'est le premier outil de vérification basé sur les diagrammes de décision binaire (ou BDD : Binary Decision Diagrams). Il a été développé en collaboration entre le ITC-IRST (Istituto di Cultura Trentin à Trento, Italie), l'Université Carnegie Mellon, l'Université de Gênes et l'Université de Trento.

NuSMV a été conçu en tant qu'une architecture ouverte pour le model checking. Il offre une vérification fiable pour les modèles de taille industrielle. Il peut être utilisé comme arrière-plan d'autres outils de vérification et comme un outil de recherche pour les techniques de vérification formelle.

NuSMV supporte l'analyse des spécifications exprimées en CTL et LTL. L'interaction avec l'utilisateur se fait à travers une interface textuelle.

➡ Même si le model checking n'est pas mis en pratique dans mon stage, nous avons étudié ce concept et fait un état de l'art parce que c'est la pierre angulaire pour notre approche (qui est la vérification symbolique des composants critiques de WCOMP).

2.3.7 Synchronisateur/Désynchronisateur

Les synchronisateur/désynchronisateurs sont des entités qui permettent de plonger les systèmes synchrones dans un univers asynchrone (ici WCOMP). Ceci passe par plusieurs principes comme la construction de l'instant logique en fonction des évènements donnés en entrées. Nous trouvons plusieurs approches tels que : GALS (Globally Asynchronous Locally Synchronous) et les machines d'exécution.

A. GALS :

GALS [12] est un modèle de calcul (ou Model of Computation (MoC)) qui est basé sur la coopération de calcul synchrone et asynchrone.
Le modèle GALS utilise le synchronisme pour concevoir des systèmes basés sur plusieurs modules localement synchrones et qui interagissent l'un avec l'autre à l'aide d'une communication asynchrone (enveloppes asynchrones).

B. Machine d'exécution

Les machines d'exécution sont des machines introduites dans un environnement donné et qui peuvent fournir des informations précises sur son comportement. Elles peuvent transformer ces données pour que d'autres environnements ou systèmes puissent les lire.

Les machines d'exécution existantes, sont toutes spécifiques et liées à un environnement particulier. Nous cherchons à créer une machine d'exécution plus générique pour qu'elle soit apte à s'intégrer dans n'importe quel environnement.

2.4 Conclusion

Le but principal de ce chapitre était de cadrer notre projet. Dans ce chapitre, nous avons défini le contexte de notre projet et présenter les concepts nécessaires pour la compréhension de notre sujet. Dans la partie qui suit, nous allons parler de ce que nous avons fait durant ce stage et nous allons présenter notre travail.

Chapitre3 :

Travaux & Réalisation

3.1 Introduction

Après avoir fait une bonne étude des concepts existants, nous passons à la réalisation.
Dans ce chapitre, nous allons décrire les différentes étapes et lots de notre projet, expliquer l'état de l'avancement de notre travail, présenter les outils que nous avons utilisés, parler des difficultés et mettre le point sur l'apport de notre projet.

3.1 Etat d'avancement

3.1.1 Lot1 : Etude Bibliographique et test de l'existant

Dans ce lot, j'ai étudié et analysé les concepts et les approches existantes pour bien comprendre le sujet, en particulier j'ai étudié les systèmes temps réel, les systèmes synchrones et asynchrones et les machines d'exécution. Puis, j'ai fait des tests sur les logiciels que je vais utiliser durant ce stage qui sont : galaxy, autom2circuit, merge_blif, blifto,clem et WCOMP (décrits ci-dessous) .

3.1.2 Lot2 : Création automatique d'un bean

Ce lot consiste principalement à utiliser le travail précédent pour compléter un travail existant et générer automatiquement un moniteur synchrone dans WComp à partir d'une spécification sous forme de FSM(Machine à Etats Finis). En effet, l'outil Galaxy permet de décrire des FSM synchrones. Ensuite, l'outil autom2circuit génère le code vers différentes cibles pour les automates décrits en Galaxy. Le but de notre travail est de compléter autom2circuit pour qu'il génère du code C# instrumenté qui est le code interne des beans qui implémentent les composants de WComp.

Nous avons utilisé un exemple de référence (« Traffic Light »). Cet exemple consiste à décrire le comportement du Traffic Light sous forme d'un simple automate en Galaxy et à comprendre le code correct qu'il faut générer.
Notre exemple est composé de deux « feux tricolores » qui contrôlent deux rues orthogonales (EST-Ouest et Nord-Sud). Chaque feu de trafic fonctionne comme suit : le feu gère trois sorties booléennes à chaque instant qui sont : rouge, orange, vert.
Ces sorties sont exclusives et ne sont vraies qu'à travers la séquence suivante :
Rouge -> Orangé -> Vert.
Ces deux feux sont décrits par une machine d'états synchrones qui contient deux automates à états finis : un automate pour le feu NS(North/South) qui contient les états green_NS, red_NS et orange_NS, et un automate pour le feu EW(East/West) qui contient les états green_EW, red_EW et orange_EW.
Il n'y a aucune entrée, nous supposons que le passage d'une lumière de feu à une autre se fait en fonction du temps logique.
Les transitions d'un état à un autre sont déclenchées par « 1 » (qui représente un instant). Green_NS est l'état initial d'un feu(NS) alors que Red_EW l'état initial de l'autre feu(EW).
Il 'y a plusieurs contraintes à respecter :

1- On ne doit pas avoir les lumières green_NS et orange_EW en même temps
2- On ne doit pas avoir les lumières green_EW et orange_NS en même moment.

Nous avons représenté l'exemple de « Traffic light » dans galaxy en dessinant deux automates parallèles (Traffic_NS et Traffic_EW) qui n'ont pas d'entrées et qui ont trois états pour chaque automate qui sont green, orange et red.
Nous avons ajouté un autre état « red » dans chaque automate afin de décaler le changement d'un automate par rapport à l'autre, ce qui nous a permis de répondre aux contraintes déclarées ci-dessus, et de respecter le bon fonctionnement des feux en même temps (voir figure10).

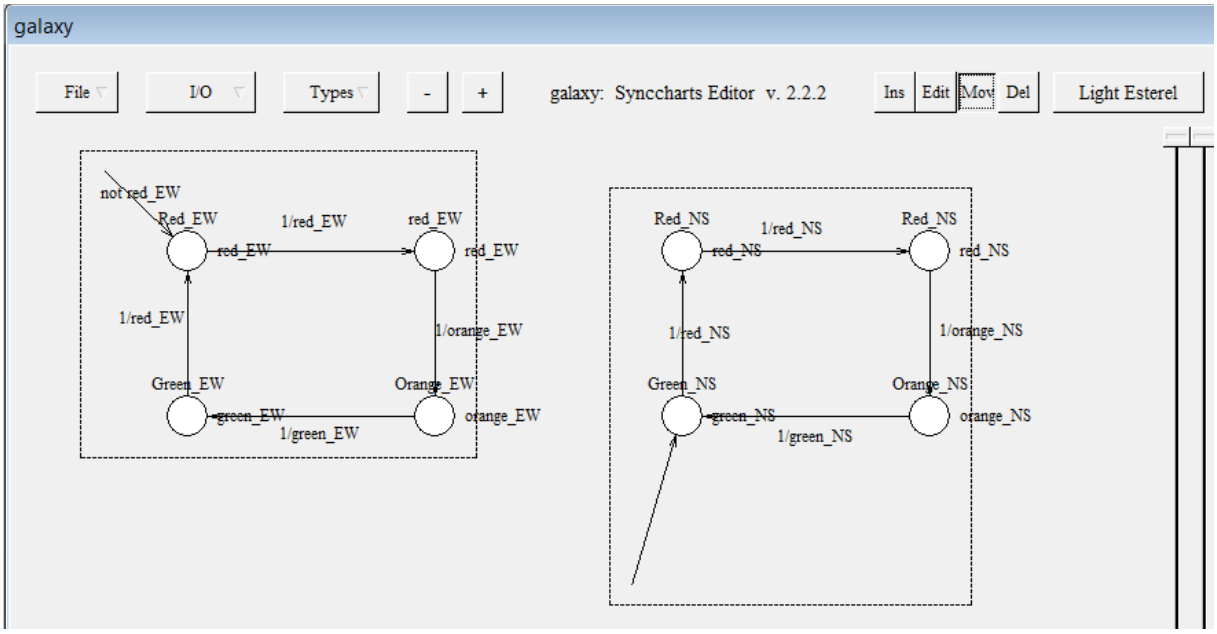

Figure 10 : L'exemple de traffic light décrit dans galaxy

Ensuite nous avons étendu autom2circuit afin qu'il génère du code C# pour WComp et nous l'avons testé.
Cette phase terminée, nous voulons maintenant passer à la création d'une machine d'exécution, Pour cela nous allons travailler sur l'exemple du Traffic Light en le rendant plus complexe, en ajoutant d'autres automates (un compteur et un sélecteur) et plusieurs autres contraintes (selon la présence des piétons et le nombre des voitures) .

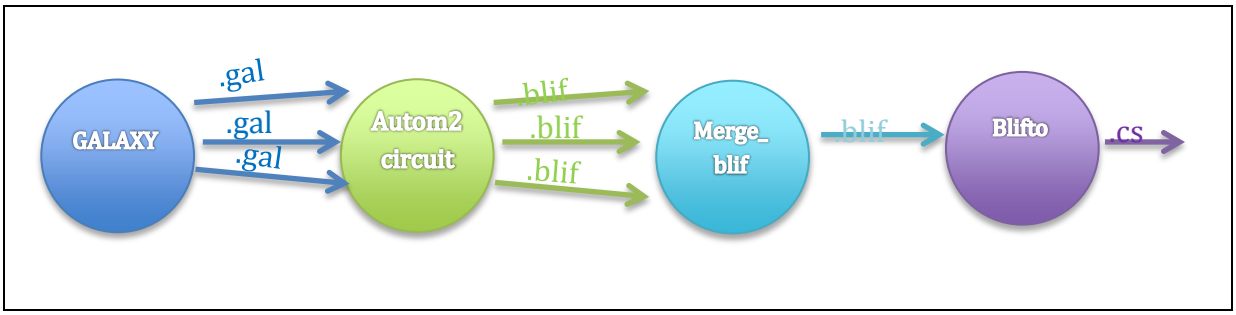

Figure 11: Etapes de la création d'un bean

Comme le montre la figure ci-dessus(figure11), nous avons créé les automates à l'aide de « galaxy » qui nous permet de générer les fichiers « .gal ». Ces fichiers sont compilés en blif par autom2circuit. Les automates sont ensuite fusionnés à l'aide de « merge_blif » qui va pour générer un fichier blif unique représentant la mise en parallèle des trois automates. Enfin, « blifto » convertira le fichier blif en fichier C # qui va présenter notre bean, cette dernière étape fait aussi l'objet de mon stage.
***.bilf** : (Berkeley Logic Interface Format) un standard crée par l'université de berkeley.
***.gal :** spécifique à galaxy.

3.1.3 Lot3 : Adaptation synchrone/asynchrone

Ce lot consiste à développer un synchronisateur(ou générateur d'entrées) qui va regrouper les données asynchrones en des instants logiques pour les envoyer au moniteur synchrone dont le rôle est de traiter ces informations. Puis nous devons créer un désynchronisateur(ou générateur de sorties) qui va recevoir les données du moniteur synchrones et les plonger dans l'univers asynchrone.
L'ensemble synchronisateur/désynchronisateur présente une machine d'exécution.
Toutes les machines d'exécution actuelles ne sont pas génériques, chacune est destinée à un environnement/fonctionnement précis. Nous avons créé une machine d'exécution générique qui sera utilisée dans n'importe quel environnement et qui est capable de communiquer avec des automates qui assurent des fonctionnements différents.
Pour réaliser cette machine d'exécution, nous avons divisé ce lot en trois parties importantes qui sont :

1- Le générateur d'entrées

Le générateur d'entrées dans notre projet est un outil qui doit transformer un flot asynchrone en un flot synchrone qui sera envoyé au moniteur synchrone (voir figure 12).

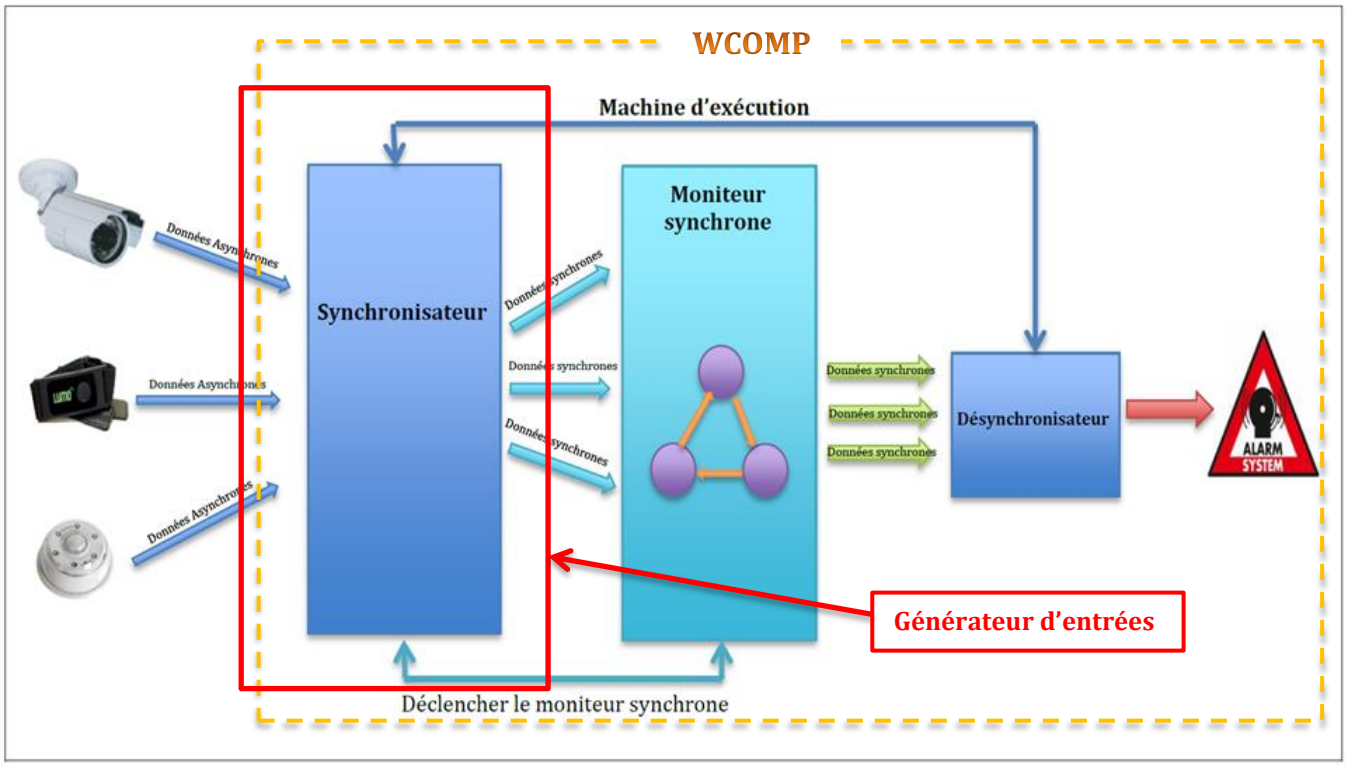

Figure 12 : partie du générateur d'entrées

Pour cela, nous avons pensé tout d'abord à deux stratégies :

1- Créer un automate qui envoie directement les données dès qu'il les reçoit.

2- Considérer que l'instant logique est déterminé par deux occurrences successives d'un même évènement (O). L'instant logique est alors composé de l'ensemble des occurrences des autres évènements arrivés entre la première et la deuxième occurrence de l'évènement(O).

➡ Cette solution n'est envisageable que si on considère un 'timeout' après lequel l'instant logique sera construit de toute façon.

Nous avons adopté la deuxième stratégie et nous l'avons développée, mais il s'est avéré que ces stratégies n'assurent pas la généricité du générateur d'entrées, parce que la politique d'envoi est toujours la même.

Donc nous avons amélioré notre approche et nous avons défini une structure générique d'évènements qui peut être fournies par n'importe quel capteur. Les évènements envoyés par les capteurs doivent avoir un nom, une valeur de présence, un type de valeur et une valeur, de plus, puisque le temps entre deux évènements envoyés n'est pas le même pour tous les capteurs, nous avons ajouté une autre propriété, appelée « elapsedTime », qui permet à chaque capteur de définir son laps de temps.

Nous avons aussi créé un générateur d'entrées qui intègre plusieurs politiques d'envoi synchrone d'évènements. Ce générateur d'entrées va recevoir les évènements asynchrones envoyés par les capteurs, créer un buffer circulaire dynamique pour chaque d'évènement envoyé par un capteur (par exemple : un buffer pour les évènements envoyés par le capteur de température, un autre buffer pour les évènements envoyés par le capteur de lumière….etc)(voir figure 13) . La création et le remplissage de ces buffers se font d'une façon automatique, en effet,

nous avons implémenté dans ce générateur d'entrée une fonction qui va prendre en considération la propriété « elapsedTime » de chaque évènement pour se déclencher automatiquement à chaque laps de temps désigné et voir s'il y'a un nouvel évènement envoyé pour le mettre dans son buffer. S'il 'y a pas d'évènement, une case vide est créée et on passe à la case suivante (voir figure14).
Après la création de ces buffers, notre générateur d'entrées va les transformer dans des flots synchrones qui contiennent l'ensemble des évènements valués et les envoyer au moniteur synchrone, selon une politique d'envoi choisie par l'utilisateur (voir figure15).
Les schémas ci-dessous expliquent mieux le fonctionnement de notre générateur d'entrées :

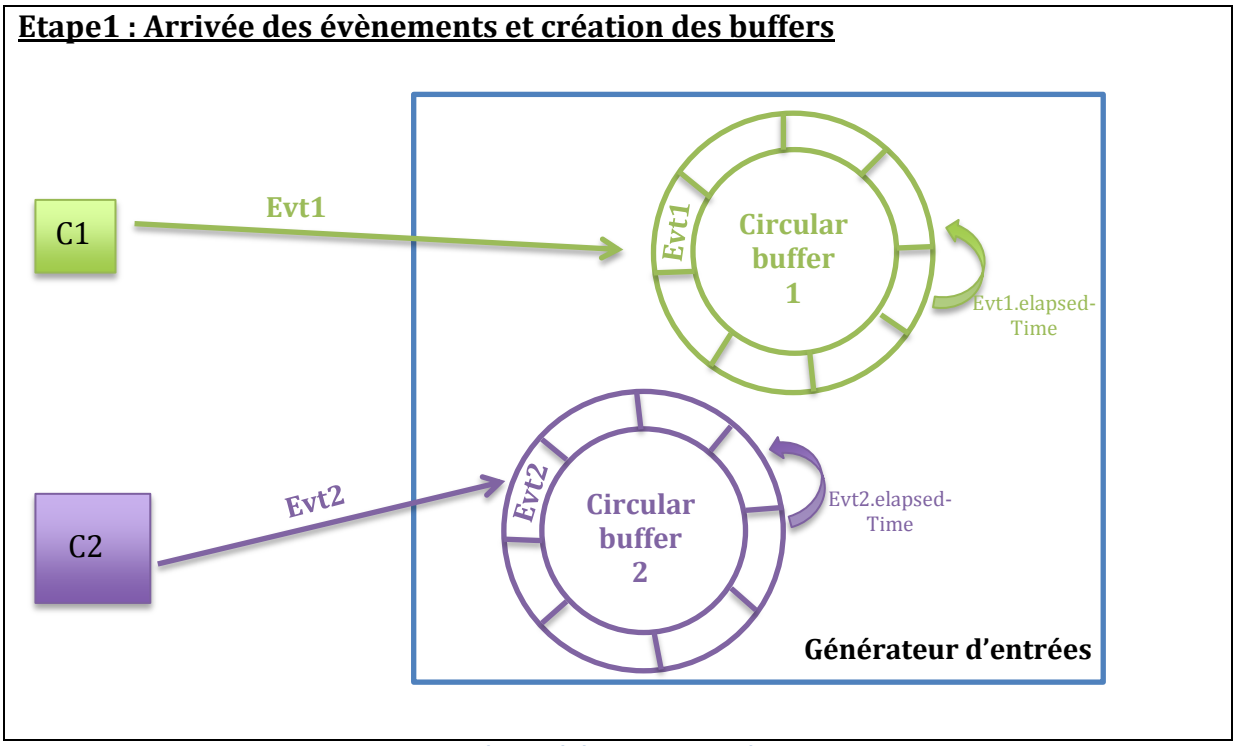

Figure 13: Arrivée des évènements et création des buffers

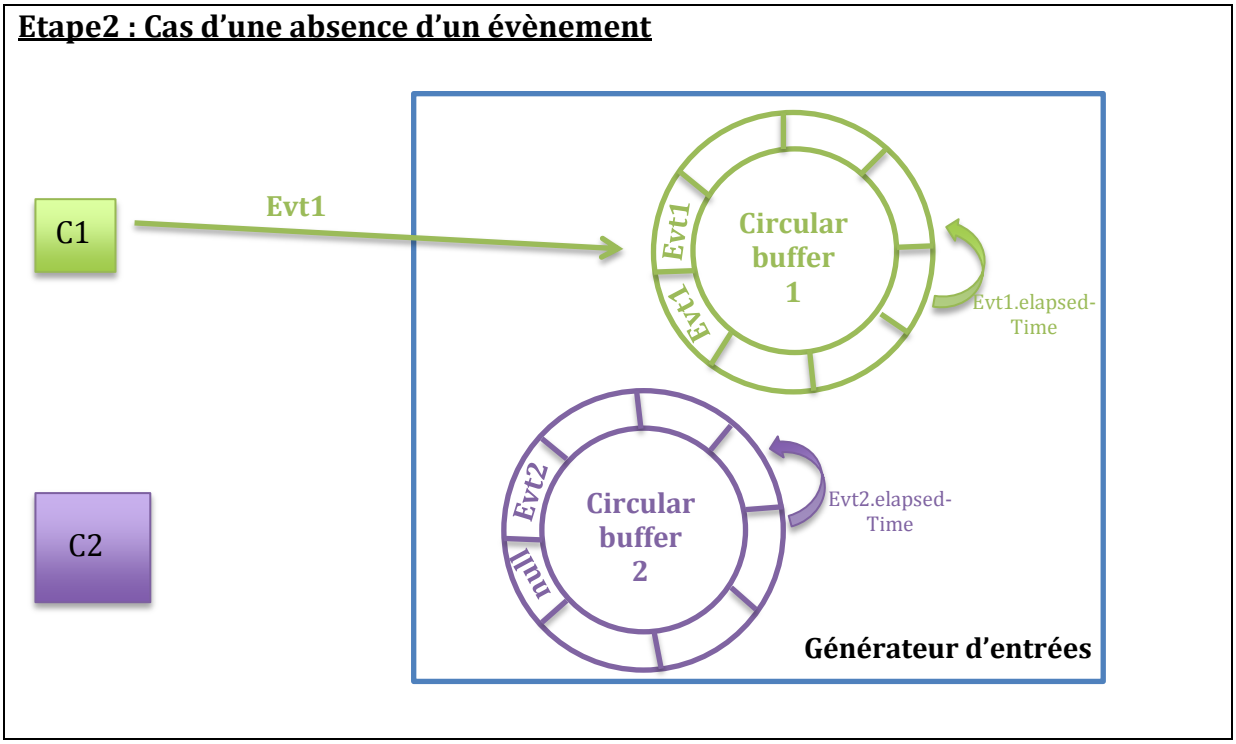

Figure 14: Cas d'une absence d'un évènement

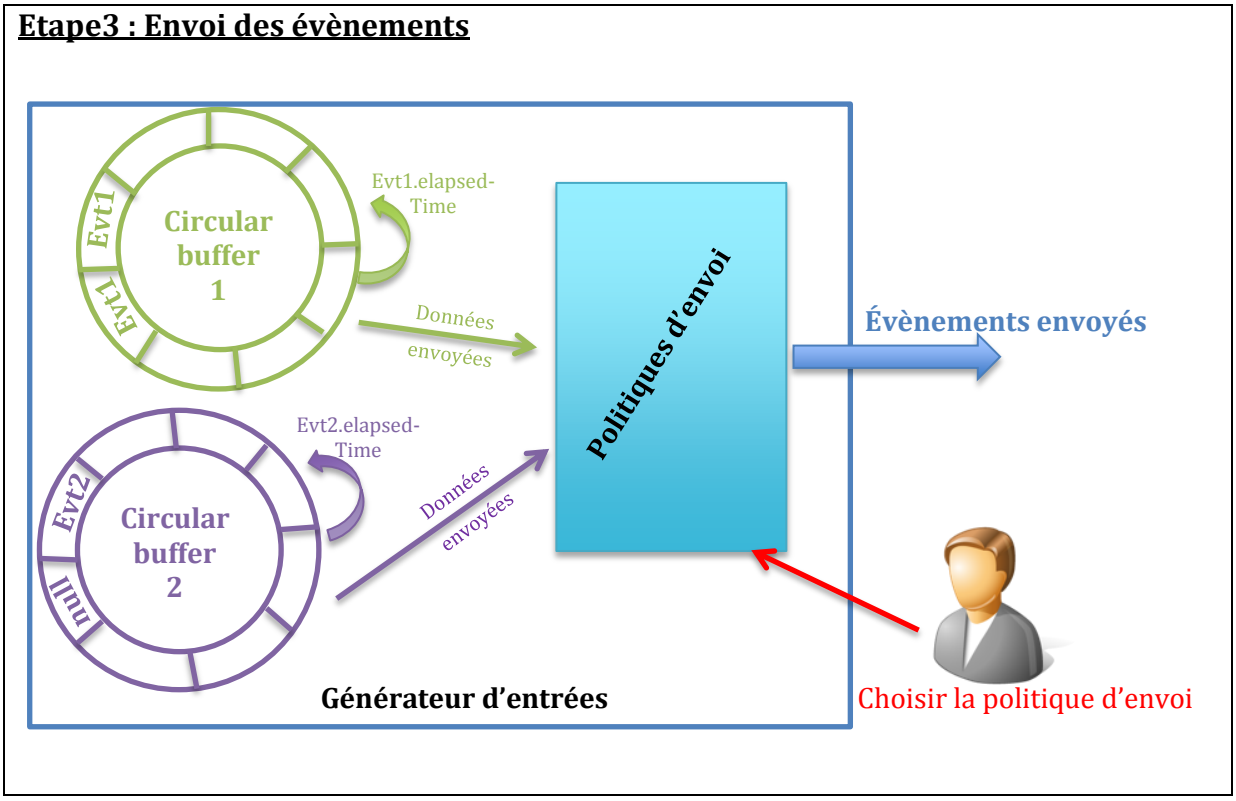

Figure 15: Envoi des évènements au moniteur synchrone

2- Le Trigger

Notre générateur d'entrées doit envoyer des évènements valués à l'automate synchrone.
Comme nous l'avons mentionné dans le lot 2, nous avons réussi à générer le bean de cet automate automatiquement à l'aide de la modification que nous avons fait dans le logiciel Aurom2circuit. Le problème est qu'autom2circuit ne traite que de simple machines de mealy qui n'utilise que des évènements purs.
D'où, nous avons pensé à créer un trigger (déclencheur) qui sera placé entre le générateur d'entrées et l'automate du moniteur synchrone. Il va prendre en entrée les évènements valués envoyés par le générateur, séparer ces valeurs et générer des évènements purs qui seront envoyés à l'automate du moniteur synchrone. Nous avons aussi besoin d'un autre composant qui va recevoir les valeurs des évènements séparées par le trigger, et qui va les relier aux évènements purs issus de l'automate du moniteur synchrone pour avoir de nouveau des évènements valués qui seront utilisés par les composants de WCOMP (voir figure 16).

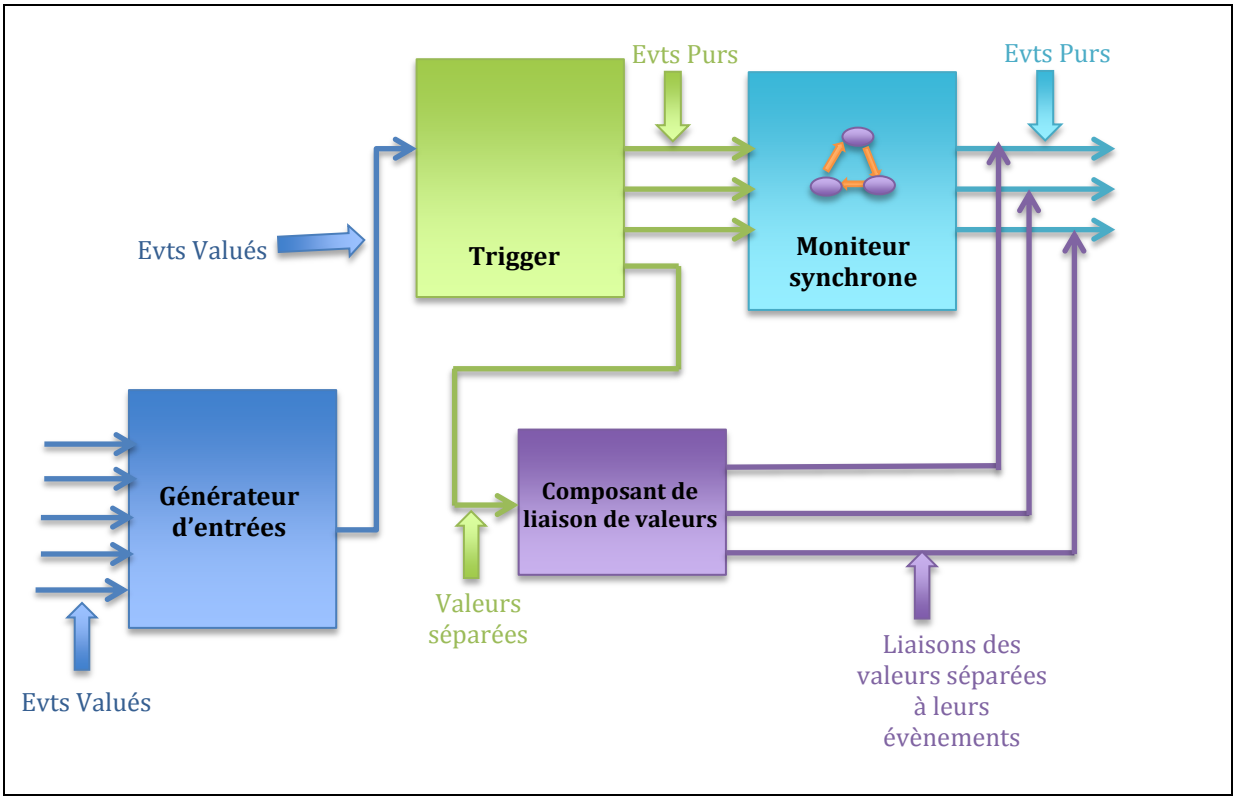

Figure 16: cas d'insertion d'un trigger

Cette solution est très lourde à mettre en pratique car elle impose d'avoir un moyen de relier les valeurs aux évènements purs de sortie de l'automate, ce qui imposerait à l'utilisateur à préciser ces relations, et à fournir un moyen d'expression de celles-ci, nous avons donc cherché une autre solution.
En effet, mes encadrants (Mme Annie RESSOUCHE et M. Daniel GAFFE), les créateurs des outils avec lesquels je travaille (à part WCOMP), ont modifié et mis à jour ces logiciels pour qu'ils puissent accepter des valeurs et créer des automates qui peuvent utiliser des types valués et des

valeurs. Donc, nous avons éliminé l'idée du trigger et nous avons travaillé directement sur l'automate de notre moniteur synchrone.

3- Le moniteur synchrone

Notre moniteur synchrone sera sous la forme d'un automate créé par l'utilisateur selon ses besoins (voir figure17).

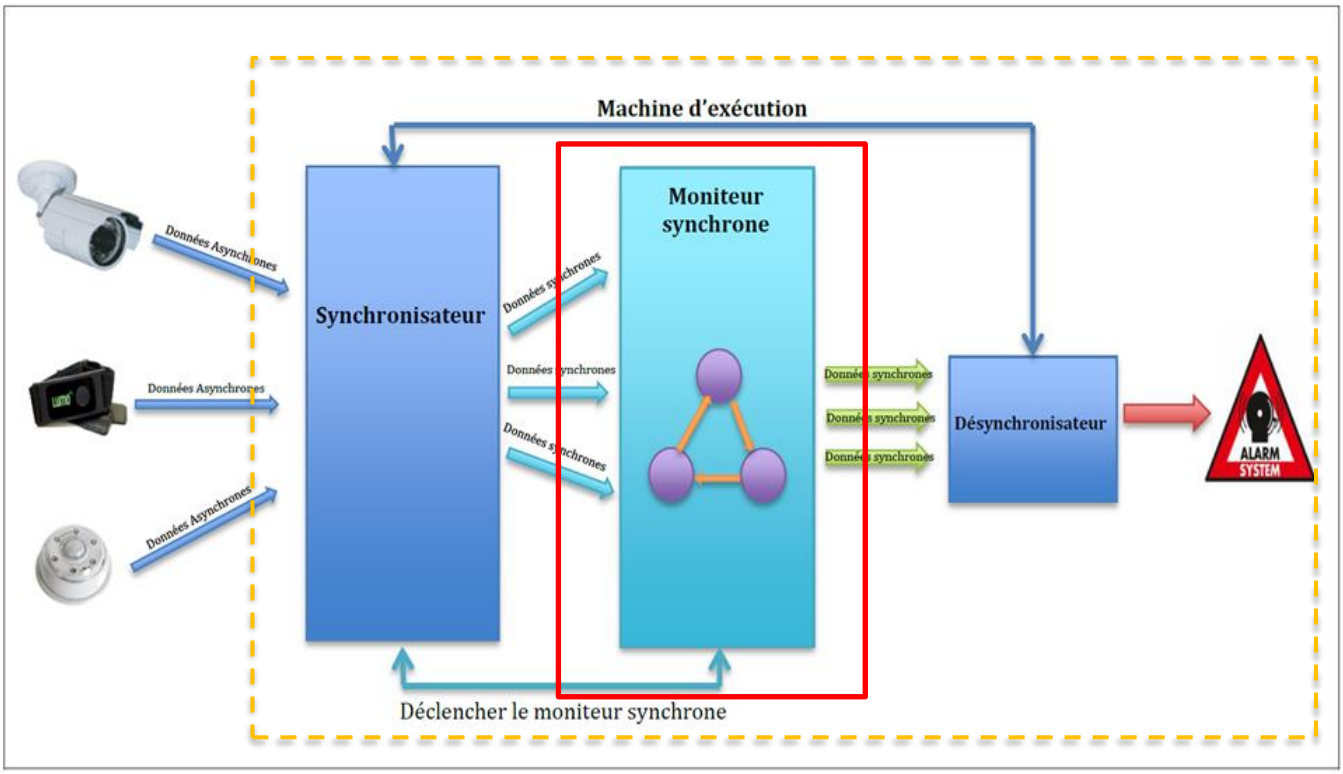

Figure 17: partie du moniteur synchrone

Dans cette nouvelle approche, l'utilisation de l'outil « Galaxy » pour décrire des automates qui manipulent des signaux valués, est couplée avec l'outil « CLEM ».
En effet, seul le langage Light Esterel (LE) accepte de tels automates. De plus la génération de code C# pour WCOMP doit maintenant s'intégrer à la chaine de compilation CLEM. Nous avons intégré dans CLEM le code qui permettra la génération du bean de l'automate (créé par galaxy) en C#, nous avons aussi fait des modifications dans « CLEF » (un logiciel qui appartient à la boite à outils de CLEM) pour pouvoir générer le fichier « .cs » (voir figure 18).

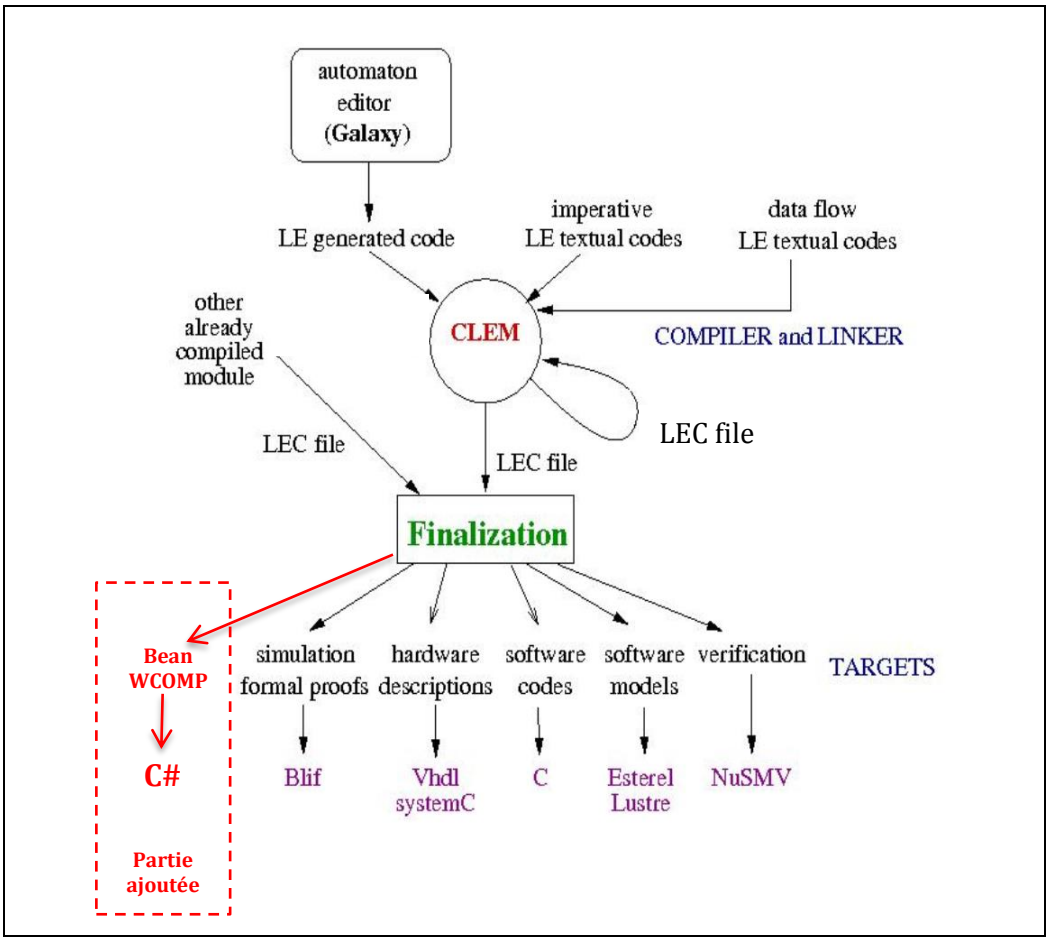

Figure 18 : Clem après mise à jour

Maintenant, nous pouvons spécifier notre moniteur synchrone avec des automates qui manipulent des évènements valués et qui peuvent aussi les envoyer directement à notre générateur de sortie.

4- Le générateur de sortie

Comme on a un générateur d'entrées, on a aussi un générateur de sortie. Le générateur de sortie est le composant qui va passer du monde synchrone au monde asynchrone de nouveau et replonger les données dans ce monde. Il va recevoir les données de l'automate du moniteur synchrone, les traiter et les envoyer d'une façon asynchrone en suivant une des politiques d'envoi vers les autres composants existants dans WCOMP(voir figure 19).

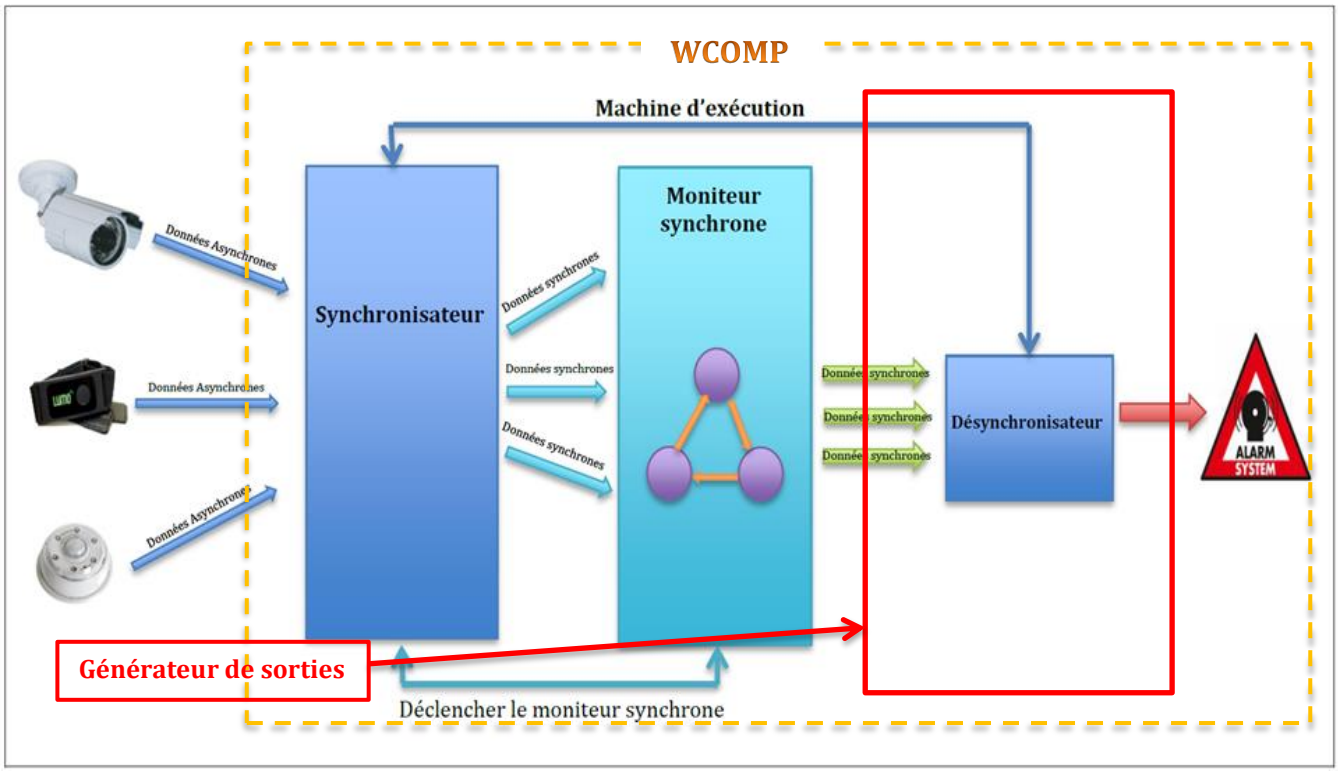

Figure 19: partie générateur de sortie

Pour cela, nous avons créé une classe abstraite pour les politiques d'envoi asynchrone, à partir de laquelle, nous pouvons intégrer, ajouter et/ou supprimer des politiques d'envoi plus facilement. Notre générateur de sortie va faire appel à ces politiques d'envoi et permettre à l'utilisateur d'en choisir une pour qu'il puisse effectuer le traitement nécessaire. Après avoir envoyé ses données asynchrones, c'est aux composants WCOMP qui vont les recevoir de choisir les évènements dont ils ont besoin pour leurs traitements.

5- La sérialisation/désérialisation :

Pour établir une relation correcte entre les méthodes d'entrée et les évènements de sortie des composants de WCOMP, et avoir un format unique et cohérent de liaison, nous avons introduit un moyen de faire la sérialisation/désérialisation des évènements. Ce qui nous a permis de générer un évènement unique et de format unique pour les entrées et les sorties de nos composants créés.

Cette sérialisation/désérialisation nécessite la définition d'une grammaire à respecter par tous les composants pour avoir ce format.

Nous avons envisagé deux grammaires possibles pour sérialiser/désérialiser nos évènements:

A- Grammaire A :

Cette grammaire va nous générer une seule chaine de caractères qui sera sous la forme suivante :

« [<Name> = <Occurrence>, [<Type>, <Valeur>] ;]+ »

Exemple :

Nous avons un programme P qui écoute les évènements « a », « b » et « v ». « a » et « b » sont des évènements purs alors que « v » est un évènement valué de type byte. Une chaine possible sera de la forme suivante :

s= « a = false ; b=true ; v=true, byte,7 ;»

Cette chaine nous informe que l'évènement « a » n'est pas présent, l'évènement «b » est présent, et l'évènement « v » est aussi présent et sa valeur est égale à 7.

B- Grammaire B :

Cette grammaire va nous générer N chaines de caractères (une chaine par évènement) qui seront sous la forme suivante :

« [<Name> <Occurrence> <Type> <Valeur>] »

Nous allons reprendre l'exemple ci-dessus avec une sérialisation qui utilise la grammaire B, nous obtenons les chaines de caractères suivantes :

"a false " "b true " "v true byte 7 "

Une seule de ces grammaires est utilisée à la fois par tous les composants de ce lot pour pouvoir envoyer le même format de données, d'où faciliter la communication entre eux.

- **Pour le générateur d'entrée :**

Le générateur d'entrée va recevoir les évènements, faire son traitement et ensuite utiliser la fonction de sérialisation pour générer un évènement unique (format unique) à utiliser et l'envoyer au moniteur synchrone (voir figure20).

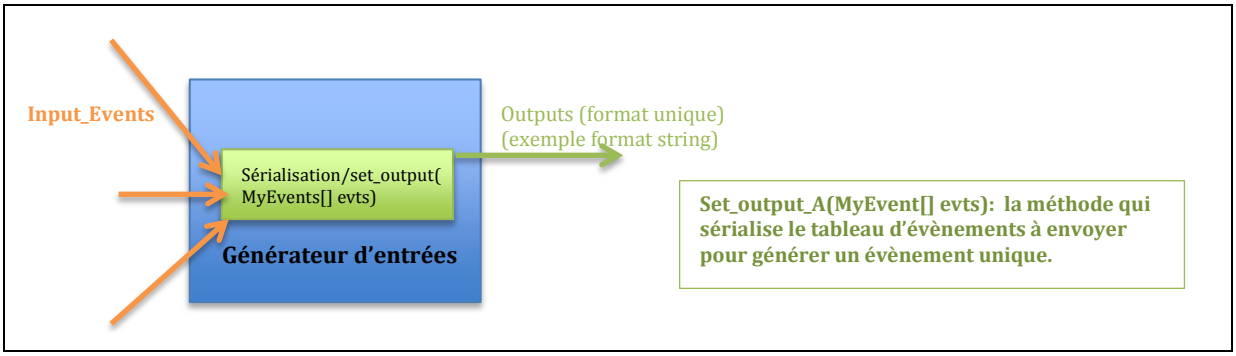

Figure 20: utilisation de la sérialisation/désérialisation dans le générateur d'entrée

- **Pour le moniteur synchrone**

L'automate généré du moniteur synchrone doit aussi respecter et utiliser la même grammaire que nous avons utilisée pour le cas du générateur d'évènements, pour ses entrées et ses sorties. Il doit recevoir un évènement unique en entrée et générer un évènement unique en sortie (voir figure21).

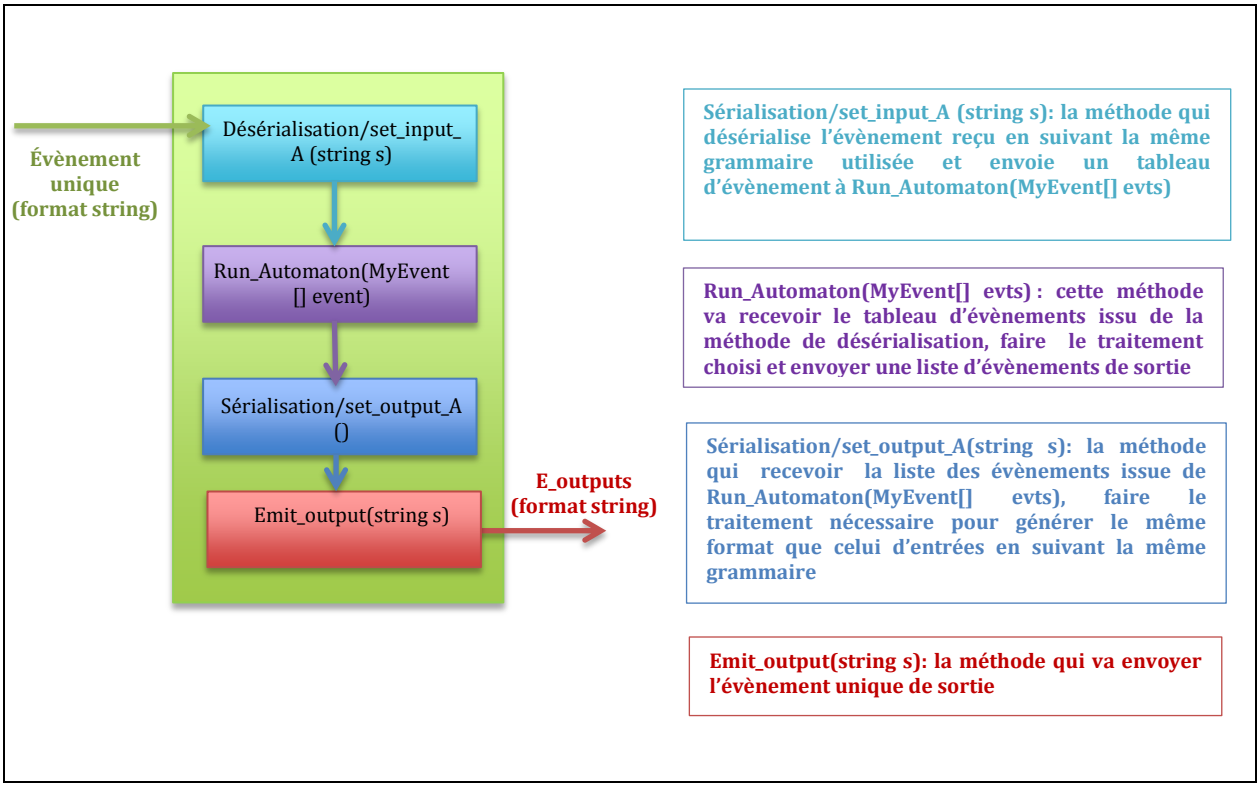

Figure 21 : utilisation de la sérialisation/désérialisation dans le moniteur synchrone

La méthode de désérialisation des entrées de l'automate (set_input_A(string s)) est la méthode qui va déclencher le mécanisme de l'itération de l'automate .

- **Pour le générateur de sortie**

Le générateur de sortie va recevoir l'évènement unique envoyé par l'automate du moniteur synchrone et va le désérialiser pour avoir l'ensemble des évènements sur lesquels il va faire son traitement.

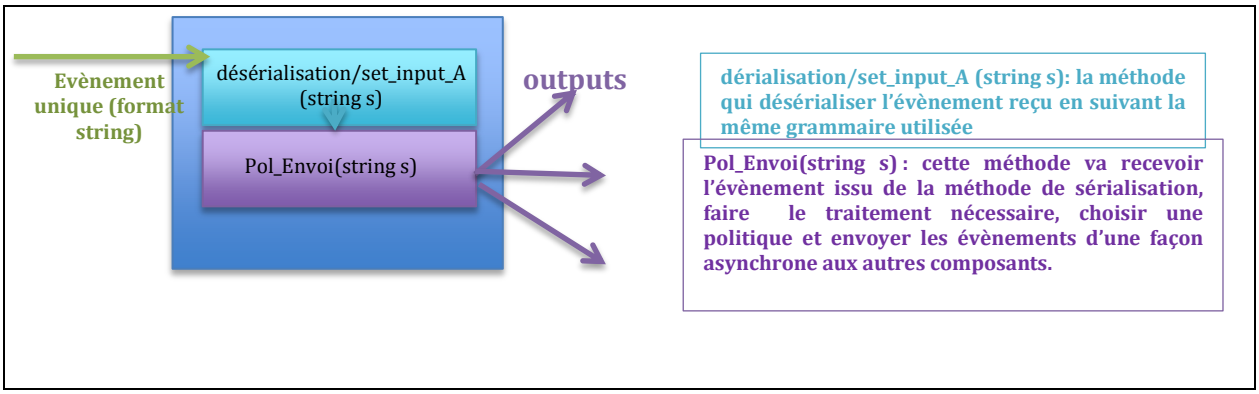

Figure 22: utilisation de la sérialisation/désérialisation dans le générateur de sorties

Le schéma et le diagramme de classe qui suivent expliquent mieux notre machine d'exécution, son fonctionnement et sa composition après l'utilisation de sérialisation/désérialisation.

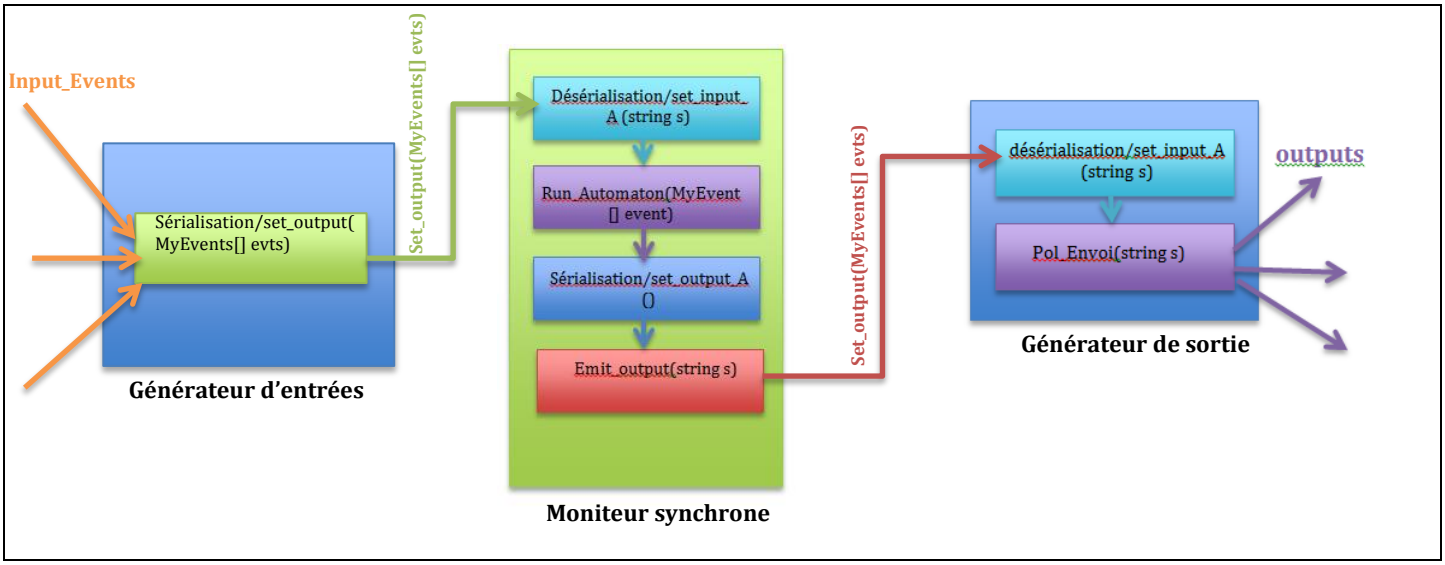

Figure 23: sérialisation/désérialisation : fonctionnement complet

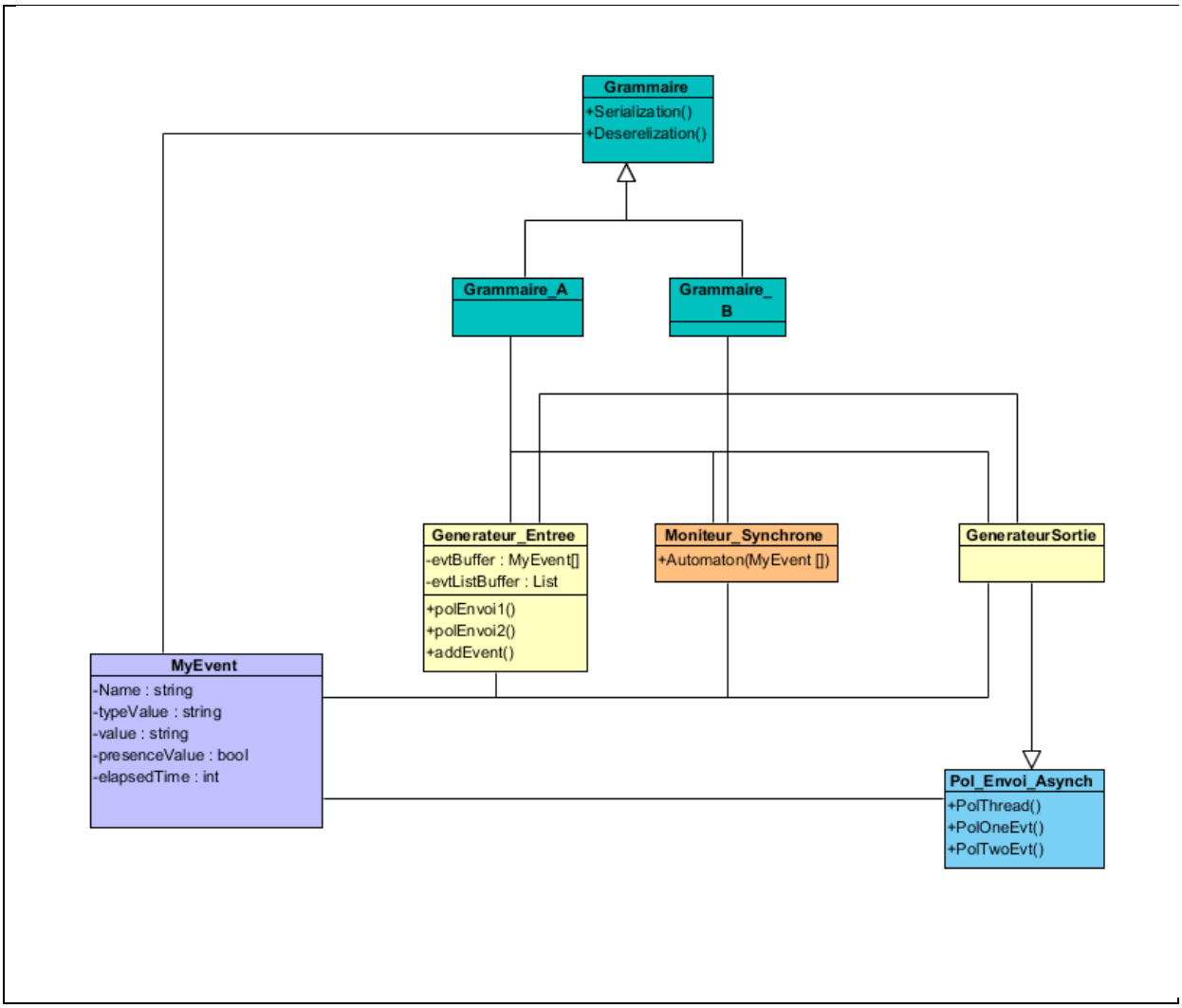

Figure 24: diagramme de classes complet

Notre travail est composé de huit classes principales qui sont (voir figure25) :

1- **La classe « MyEvent »** : c'est la classe qui contient une description générique des évènements qui seront envoyés par des capteurs différents.
2- **La classe « Generateur_Entree »** : c'est la classe du générateur d'entrées.
3- **La classe « Moniteur_Synchrone »** : c'est la classe qui présente l'automate du moniteur synchrone, cette classe est générée automatiquement.
4- **La classe « GenerateurSortie »** : c'est la classe qui présente notre générateur de sortie.
5- **La classe « Grammaire »** : c'est une classe générique et abstraite, elle contient les signatures des méthodes de sérialisation/désérialiation
6- **La classe « GrammaireA »** : elle va hériter les méthode de la classe « Grammaire » pour développer sa propre façon de sérialisation/désérialisation.
7- **La classe « GrammaireB »** : de même que la classe « GrammaireA ».
8- **La classe « Pol_Envoi_Asynch »** : c'est une classe abstraite qui contient les signatures des méthodes d'envoi asynchrone. La classe « GenerateurSortie » va hériter de cette classe pour implémenter ses méthodes d'envoi asynchrone.

Nous avons mis une réalisation d'exemple qui explique mieux le fonctionnement de nos composants.

3.1.4 Lot4 : Composition des moniteurs synchrones

Comme nous avons expliqué dans le chapitre2, un composant critique peut avoir plusieurs moniteurs synchrones, d'où on peut avoir plusieurs accès à une seule entrée. La composition synchrone sous contraintes peut être une solution pour remédier à ce problème. Mais cette solution va aussi poser plusieurs contraintes puisqu'elle n'est pas incrémentale ni adaptative aux changements, d'où, dans le cas d'une apparition d'un nouveau composant ou d'une disparition d'un composant existant, la mise à jour de ce composant doit être faite manuellement, de plus les modification vont entrainer une nouvelle composition sous contraintes des moniteurs synchrones modifiés (voir figure 25).

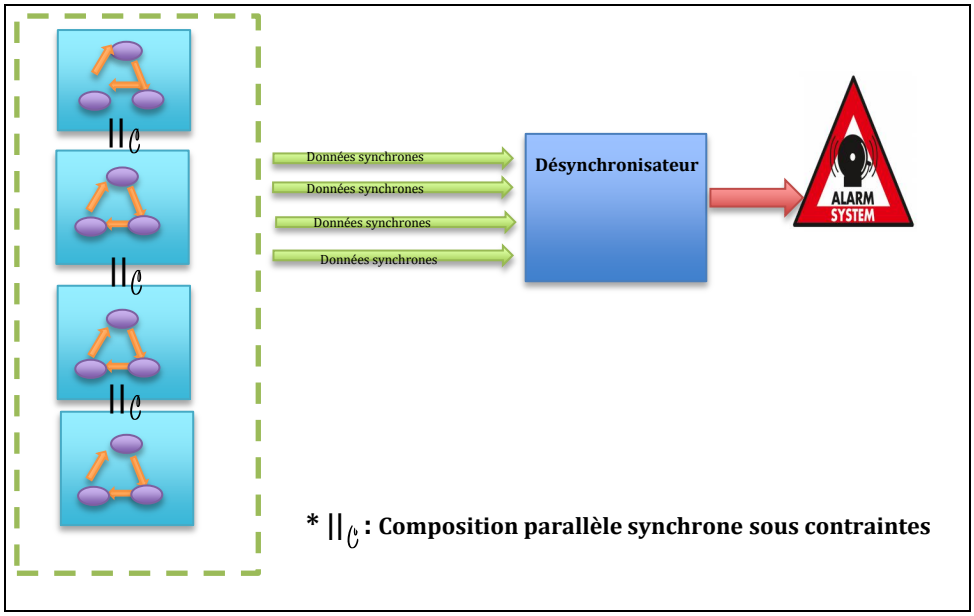

Figure 25: La composition parallèle synchrone

Pour cela, nous avons pensé à une autre solution qui répond à ce problème. Nous allons ajouter un moniteur de combinaison qui implémente la fonction de contraintes. Ce moniteur sera mis en parallèle avec les moniteurs synchrones. Ainsi, nous remplacerons une opération complexe(de composition sous contraintes) par une traditionnelle composition parallèle synchrone (voir figure 26).

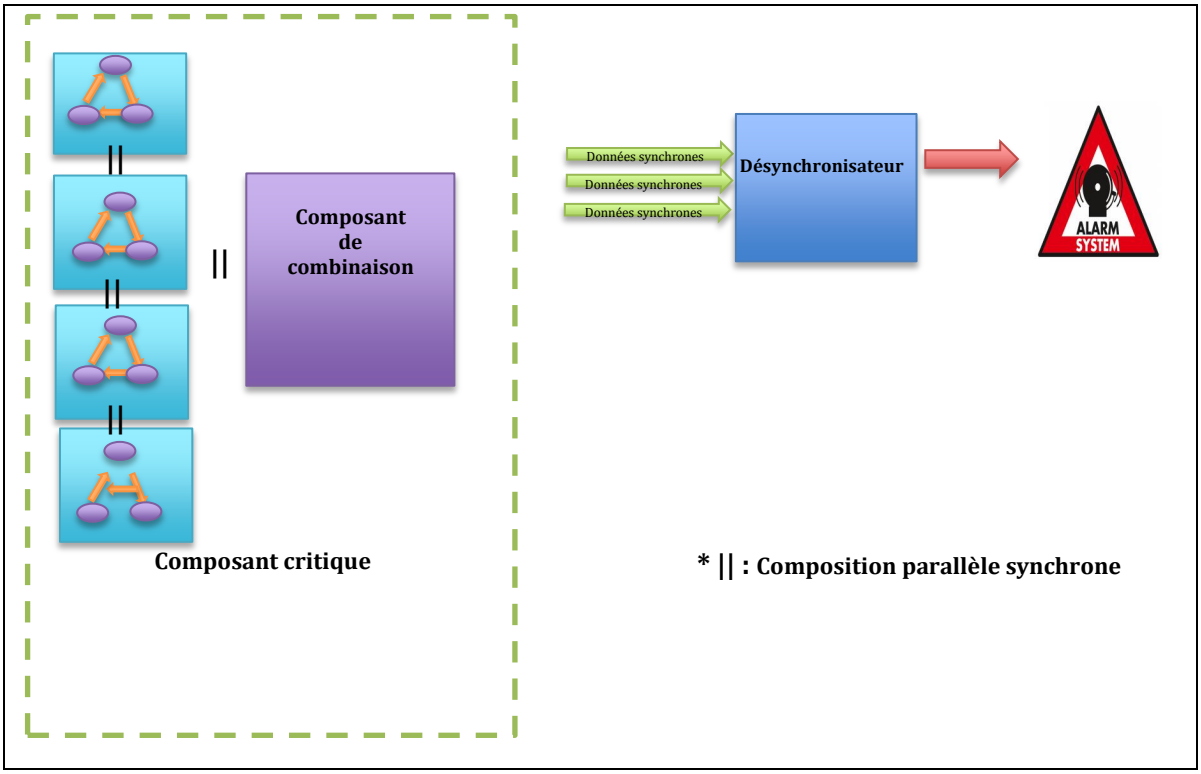

Figure 26: composition parallèle synchrone

L'avantage de cette solution est :

A. Dans le cas d'une apparition d'un nouveau moniteur synchrone :

En étend seulement le moniteur de composition et on ajoute le nouveau moniteur synchrone.

.

B. Dans le cas d'une disparition d'un nouveau moniteur synchrone :

En fait, dans l'approche synchrone, on peut représenter d'une façon équivalente un automate par un système d'équations booléennes. Dans cette représentation, l'opération de parallélisme se résume en la juxtaposition des systèmes d'équations respectifs. Donc, pour gérer la disparition d'un moniteur synchrone, il suffit de changer le moniteur de composition et d'inhiber les équations du moniteur qui sont devenus obsolètes (voir figure27).

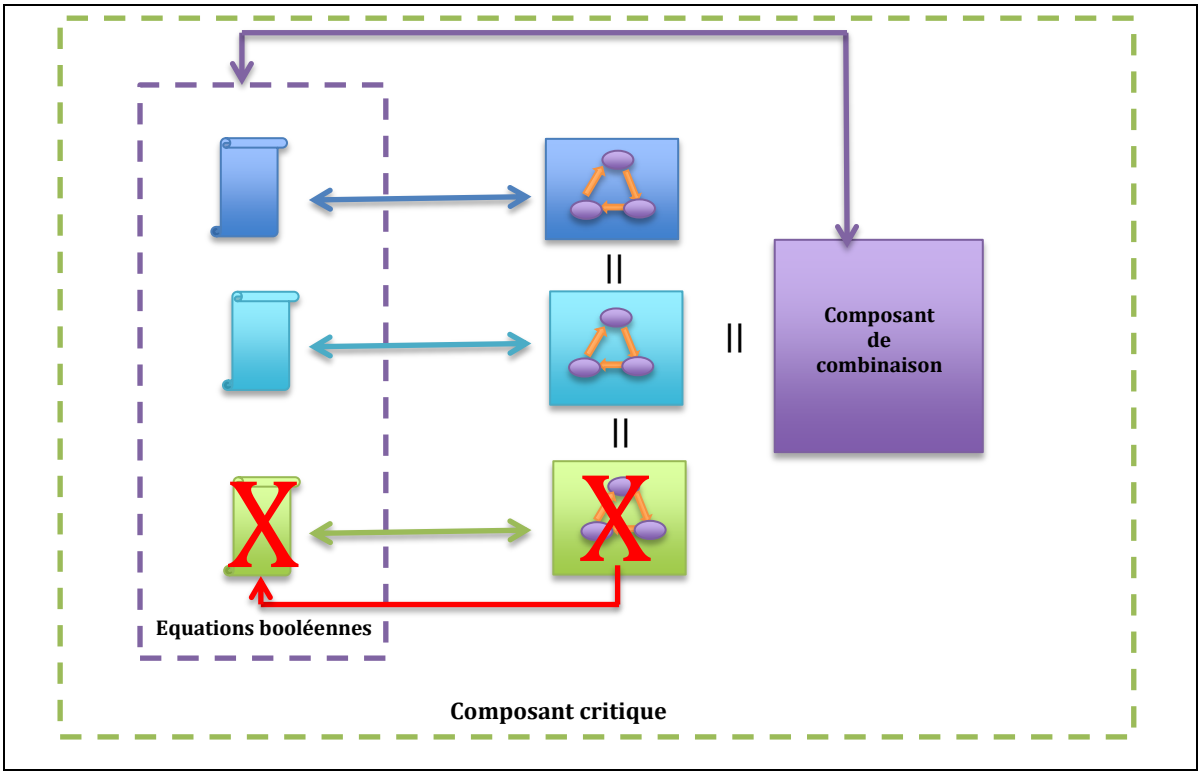

Figure 27: fonctionnement du composant de combinaison

J'ai commencé à étudier la composition et ses principes et je vais finir cette partie dans le mois qui me reste.

3.2 Outils utilisés

3.2.1 Galaxy

Galaxy est un logiciel qui permet de créer et d'éditer quatre modèles d'automates d'états finis (FSM), qui sont (voir figure28) :

- Les automates simples (en mode « basic automaton »)
- Les automates parallèles (en mode « parallel automaton »)
- Les automates hiérarchiques (en mode « LightEsterel »)
- Les SyncCharts (en mode « SyncCharts ») : ce modèle d'états est le plus complet vu qu'il intègre tous les comportements Light Esterel et qu'il dispose de la notion de transition immédiate et de suspension.

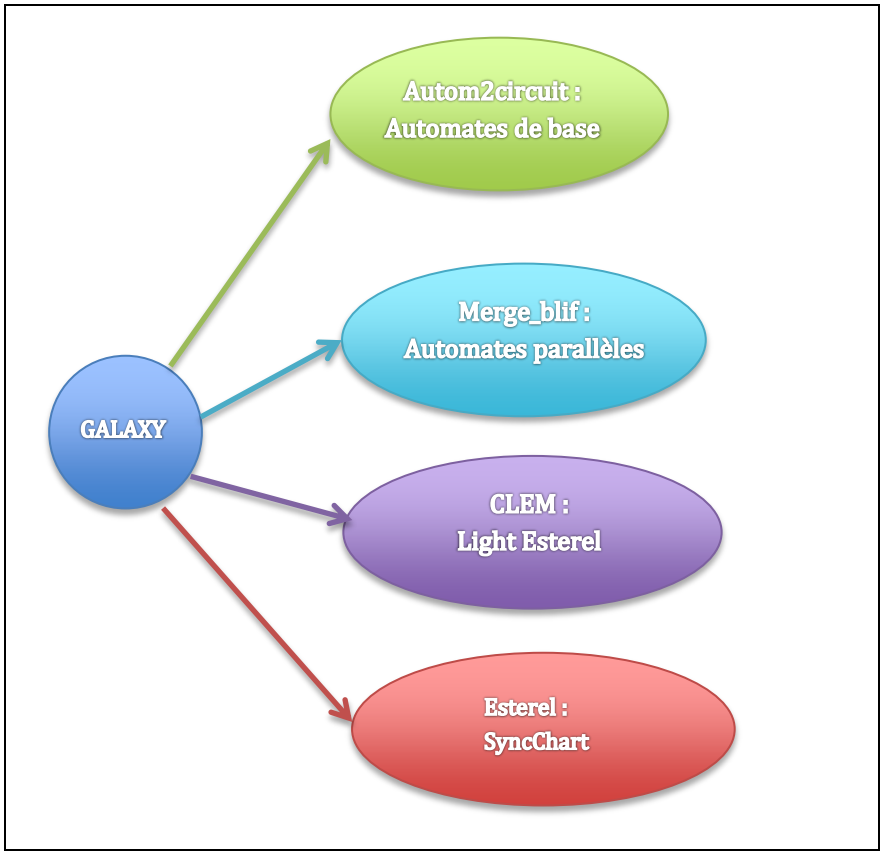

Figure 28: Les modèles d'automates

Ce logiciel traite deux aspect importants pour notre projet qui sont les machines à états finis et les langages synchrones. Il présente ces automates sous la forme d'un fichier textuel (.gal). Ce fichier est le point d'entrée d'autom2circuit.

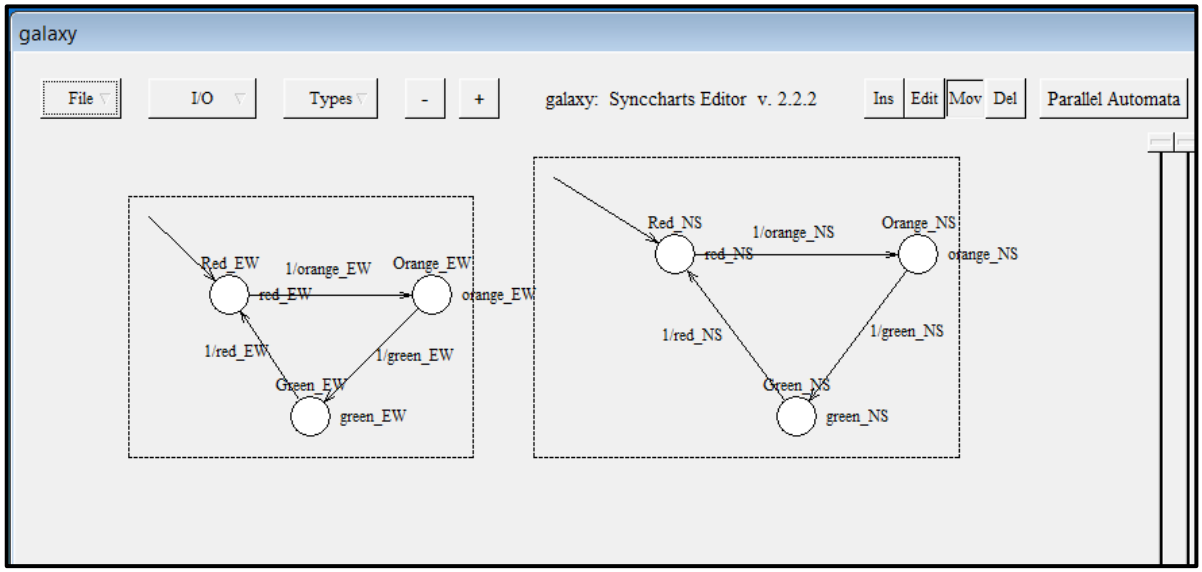

Figure 29: l'outil galaxy

3.2.2 Autom2Circuit

Autom2circuit est un logiciel qui utilise les fichiers « .gal » créé par galaxy pour synthétiser les automates dans un format implicite (machines de Mealy ou Moore booléenne) .Il permet de représenter leurs comportements dynamiques. Cet outil peut générer plusieurs types de fichiers selon le langage qu'on souhaite utiliser : cible logicielle (C, Esterl, Lustre, C#), cible matérielle (vhdl), outil de vérification et simulateur (blif, bac) et enfin, documentation (doc et latex).

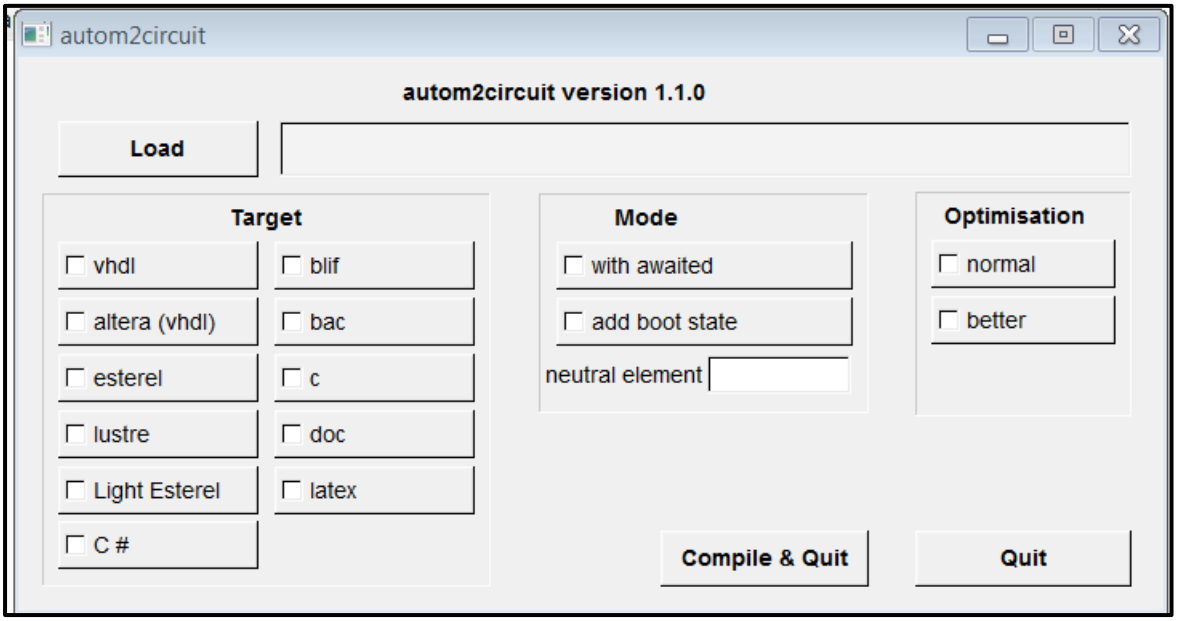

Figure 30: L'outil Autom2Circuit

3.2.3 Blif_simul

Blif_simul est un simulateur graphique d'automate implicite au format blif. C'est un logiciel qui nous aide à simuler et tester le fonctionnement des automates que nous avons créés à l'aide de galaxy, et compilés par autom2circuit.

Figure 31: L'outil blif_simul

3.2.4 Merge_blif

Merge_blif est un logiciel qui permet la concaténation ou la fusion de deux automates et d'avoir par la suite un seul automate. Ce logiciel utilise les deux fichiers « .blif » et qui présentent les deux automates et génère un troisième fichier « .blif » qui présente la fusion de ces derniers, et dont les entrées/sorties communes ont été interconnectées.

3.2.5 Blifto

Blifto est un convertisseur de fichier « .blif ». Nous allons modifier ce logiciel pour convertir notre fichier blif en « C# », pour la création de notre bean qui sera utilisé dans WCOMP.

3.2.6 WCOMP[6]

Ce middleware présente notre environnement asynchrone dans lequel, nous allons intégrer nos composants synchrones. En effet tous les composants à créer (synchronisateur, désynchronisateur ou le moniteur synchrone) doivent être sous la forme des bean utilisé par WCOMP, pour que nous puissions avoir une transition de l'asynchrone vers le synchrone, en utilisant les données asynchrone envoyées par les capteurs, les lier à notre générateur d'évènements (synchronisateur) qui va faire son traitement et les envoyer au moniteur synchrone(Automate), qui va à son tour faire son traitement et l'envoyer au désynchronisateur pour qu'il les replonges de nouveau dans le monde asynchrone et les lier aux autres composants de WCOMP.

3.2.7 CLEM[1.3.0]

CLEM est une boite à outils qui s'appuie sur le langage LE et qui permet la compilation, la simulation et la génération du code pour plusieurs cibles.
Il utilise les fichiers « .lec » qui ont été généré à partir du fichier « .le » issu par galaxy (après la mise à jour de galaxy, il génère deux fichiers : un fichier « .gal » et un fichier « .le »).
Cet outil peut générer plusieurs types de fichiers selon le langage qu'on souhaite utiliser : cible logicielle (C, Esterl, Lustre, C#), cible matérielle (vhdl, systemC), outil de simulation (blif) et de vérification (nuSMV). Nous avons utilisé la version qui supporte les automates avec des signaux valués.

3.2.8 CLEF

CLEF est aussi un moyen qui permet la génération du code vers différente cibles tels que le code C, VHDL, systemC…., en utilisant le fichier « .LEC ». nous avons modifié le fichier source de CLEF pour qu'il puisse aussi générer du code C#.

3.3 Travaux

3.3.1 Travaux effectués

#	Titre du lot	Type	Début	Fin
L1	Etude Bibliographique + test de l'existant	RECH	07/04/2014	30/04/2014
L2	Création d'un bean automatique	RECH+IMPL	1/05/2014	15/06/2014
L3	Adaptation synchrone/asynchrone	RECH	16/06/2014	30/08/2014

Figure 32: lots réalisés

Ces travaux ont été expliqués dans la partie 3.1 : Etat d'avancement

3.3.1 Travaux restants

#	Titre du lot	Type	Début	Fin
L4	Composition des moniteurs synchrones	RECH	1/09/2014	15/09/2014
L5	Amélioration	RECH/IMPL	15/09/2014	30/09/2014

Figure 33: Lots restants

3.3.2 Planning prévisionnel

#	**Lot** / Tâche	*Avril*	*Mai*	*Juin*	*Juillet*	*Aout*	*Septembre*
L4	**Composition des moniteurs synchrones**						
4.1	Pré expérimentation concrète et création d'un premier prototypage						
4.2	Etablissement des informations et des principes d'automatisation						
4.3	Implémentation de la nouvelle approche						
L5	**Amélioration**						

Figure 34 – Diagramme de Gantt

✓ Lot L4 :

Identifiant	*L4*	Date de démarrage	*01/08*
Titre	**Composition des moniteurs synchrones**		
Type	*RECH + IMPL*		

Objectifs du lot

Ce lot consiste à faire la composition des moniteurs synchrones d'une façon incrémentale.

Description du lot

Tache 4.1 *:* Pré expérimentation concrète et création d'un premier prototypage
Cette tâche consiste à faire des tests de ce qui est existant et de faire un premier prototypage pour apprécier le résultat.

Tache 4.2 *:* Etablissement des informations et des principes d'automatisation
Cette tâche consiste à choisir les principes nécessaires pour la composition des moniteurs synchrones.

Tache 4.3 *:* Implémentation de la nouvelle approche
Cette tâche consiste à implémenter et tester cette nouvelle approche.

✓ Lot L5 :

Identifiant	*L5*	Date de démarrage	*15/09*
Titre	*Amélioration*		
Type	*RECH + IMPL*		

Objectifs du lot

Ce lot consiste à proposer et établir de nouvelles approches pour améliorer le fonctionnement de la machine d'exécution et du parallèle synchrone.

Description du lot

Tache 5.1 *:* Amélioration

Cette tâche consiste à chercher de nouvelles idées et de nouveaux principes qui vont permettre un fonctionnement plus rapide et plus efficace de la machine d'exécution et du moniteur synchrone.

3.4 L'apport du projet de stage

Ce projet permet d'introduire effectivement un moyen de vérifier les composants critiques de WComp. En effet, toutes les machines d'exécution qui existent aujourd'hui sont des machines spécifiques à un certain environnement et nous ne pouvons pas les utiliser dans d'autres contextes.

Nous avons réussi dans ce projet à créer une machine d'exécution générique qui va être utilisée dans plusieurs contexte et qui a aussi éliminé le problème de la communication et la liaison entre le synchrone et l'asynchrone.

D'autre part le fait de créer les beans et d'intégrer une partie qui permet la génération automatique des beans de l'automate va non seulement faciliter beaucoup la tâche aux concepteus/développeur et utilisateur de WCOMP mais aussi gagner du temps, surtout qu'avant il fallait développer tout le code à la main.

Aujourd'hui un utilisateur qui souhaite établir une communication synchrone/asynchrone dans WCOMP va directement choisir les bean du générateur d'entrées et du générateur de sortie, dessiner l'automate qu'il souhaite avec galaxy et générer son bean automatiquement à l'aide de CLEM.

3.6 Intérêt et difficultés de stage

Après cinq mois de Stage à l'INRIA, je me permets de dire que je suis chanceuse d'avoir fait un stage pareil et dans un établissement pareil.

Sur le plan humain, Vu que l'équipe STARS de L'INRIA contient des chercheurs, des ingénieurs et des stagiaires de nationalités différentes, ceci m'a donnée l'occasion de connaitre de nouvelles cultures, de nouvelles traditions et même de nouvelles langues. Dans cette équipe on ne sent pas la différence entre un directeur, chercheur, ingénieur ou stagiaire et c'est ce que j'ai beaucoup apprécié, tout le monde est ouvert et gentil, et chacun essaie d'aider les autres et de les connaître plus, c'est une merveilleuse équipe, et je ne me suis jamais sentie seule ou isolée. Quant à mes superviseurs, Mme Annie RESSOUCHE et M. DANIEL GAFFE, ils étaient toujours souriants, patients et compréhensifs, ils essaient toujours de m'aider, me trouver des astuces, de discuter et de suivre mon avancement et de me donner de nouvelles idée pour améliorer mon travail.

M. Jean-Yves TIGLI et M. Stéphane LAVIROTTE aussi m'ont donnée de nouvelles idées pour améliorer le travail pendant les réunions que nous avons organisées, vu qu'ils ont participé à la création de WCOMP, le middleware sur lequel je travaille, donc ils n'ont jamais hésité à m'encourager et me donner des conseils qui m'ont servi dans mon travail.

Je remercie infiniment tous les membres de cette équipe pour les beaux souvenirs que je garderais tout au long de ma vie.
Sur le plan Technique, ce stage m'a permis de connaître un nouveau domaine de recherche qui est vraiment intéressant (le domaine du synchrone), de renfoncer mes connaissance en C# et en C++, de mieux connaitre le middleware WCOMP, vu que je l'ai déjà utilisé l'année dernière (dans mon projet de fin d'étude) et cette année (dans mon projet de fin d'étude) et de travailler avec de nouveaux logiciels comme galaxy, merge_blif et de participer au développement d'une petite partie dans d'autre logiciels comme clem et Autom2circuit.
Au niveau de difficultés, il faut dire que m'ont travail n'a pas été facile, J'ai fait plusieurs essais et nous avons proposés plusieurs solutions pendant les réunions qui m'ont permis de mieux comprendre ce qu'il faut faire et améliorer mon travail.
Plusieurs contraintes ont été posés jusqu'à l'instant : il fallait avoir un générateur d'évènement générique qui pourra être utilisé dans n'importe qu'elle situation et communiquer avec n'importe quel capteur, en plus, nous avons eu un problème de traitement des évènements valués, vu que les automates crées par les logiciels avec lesquels je devais travailler ne traitent que des évènements purs.
Nous avons réussi à créer le bean de ce générateur d'évènements générique en prenant en considération toutes les informations qui peuvent être envoyées par n'importe quel capteur. Quant au problème, des évènements valués, Mme Annie RESSOUCHE et M.Daniel GAFFE ont modifié et mis à jours leurs logiciels pour qu'ils puissent accepter des évènements valués. Quant à moi, j'ai intégré dans Clem la partie qui permet de générer automatiquement le Bean de l'automate qui sera utilisé dans le middleware WCOMP et lié au générateur d'évènements.

3.7 Conclusion

Dans ce chapitre, nous avons décrit les lots et les de réalisation du projet en spécifiant les outils que nous avons utilisés, nous avons aussi parlé de l'apport du projet et les difficultés que nous avons rencontrées. Nous avons ainsi pu conclure le dernier chapitre de notre rapport.

Conclusion générale

Conclusion générale

Les systèmes ubiquitaires présentent un nouveau concept et même une nouvelle ère de l'informatique, qui consiste en l'utilisation de systèmes informatiques partout et de façon transparente.

Ces systèmes sont aujourd'hui présents dans une diversité de domaines dont le domaine des applications critiques.

Ce projet nous a permis d'introduire un nouveau concept dans l'informatique ubiquitaire qui nous permet de faciliter la communication synchrone/asynchrone avec les composants critique situé dans un environnement asynchrone (le middleware WCOMP), à l'aide d'une adaptation synchrone/asynchrone.

Cette adaptation a été créée à l'aide d'une machine d'exécution générique qui pourra être utilisée dans plusieurs contextes et qui aide à plonger d'une part les données synchrones dans un monde asynchrone et d'autre part les données asynchrones dans le monde synchrone, du middleware WCOMP, pour une communication efficace et facile avec ses composants.

Ce projet m'a permis d'améliorer voire consolider mes connaissances en langage C#, en systèmes synchrones, l'utilisation de WCOMP, et il m'a permis d'autre part de découvrir de nombreux concepts liés aux machines d'états et à leur synthèse. Ce stage m'a permis aussi comment appréhender à la vérification symbolique des middlewares.

Toutefois, j'étais contrainte de faire face à une multitude de problèmes qui ont été résolus, comme la nécessité de l'utilisation des évènements purs pour certains logiciels alors que nous avons des évènements valués. J'ai également été confrontée à des bugs que j'ai dus contournés.

Pour finir, les résultats de ce qui a été fait sont certes satisfaisants mais ils nécessitent certaines améliorations. Je compte mettre en pratique le dernier mois qui me reste de ce stage pour avancer sur plusieurs points. En particulier, je vais mettre à profit le temps qui me reste pour améliorer la composition des moniteurs synchrones.

Comme perspectives, nous pouvons rajouter d'autres politiques d'envoi au générateur d'entrées, on peut aussi étudier en profondeur la partie de triggering et ajouter un trigger qui permettra de séparer les valeurs des évènements et les relier après avoir fait le traitement. D'une autre part, nous pouvons améliorer la relation entre le générateur et le moniteur synchrone de telle façon que le fonctionnement du générateur d'entrée ne soit déclenché qu'avec un évènement déclencheur envoyé par le moniteur synchrone.

Nous pouvons aussi introduire la préemption sur le cycle du contrôle de l'automate et poser des contraintes sur le temps de traitement des systèmes dynamiques synchrones pour garantir l'atomicité de la réaction. De plus la vérification en CLEM ne se fait que pour les évènements purs, nous pouvons envisager d'utiliser d'autres model-checkers capables de faire la vérification sur les évènements valués.

Bibliographie

[1] : *Informatique ubiquitaire et pervasive : Issam REBAI- Equipe 3S – Departement informatique, 20/02/2014.*
[2] : *A middleware for ubiquitous computing : WCOMP: Jean-Yves TIGLI, Michel RIVEiLL, Gaeten REY, Stéphane LAVIROTTE, Vicent HOURDIN, Daniel CHEUNG-FOO-WO, Eric CALLEGARI, Equipe RAINBOW, Janvier 2008*
[3] : *Composition and Formal Validation in Reactive Adaptive Middleware: Annie Ressouche — Jean-Yves Tigli — Oscar Carrillo, 2011*
[4] : *Systèmes Temps Réels , Caractérisation d'un système temps réel : claude baron*
[5] : *Approches Synchrones et Asynchrones pour la conception de systèmes temps réel critiques : F. Boniol ,ONERA-CERT,*
[6] : *LOGIQUE PROGRAMMABLE ASYNCHRONE POUR SYSTÈMES EMBARQUÉS SÉCURISÉS : Taha Beyrouthy, 2009*
[7] : *Compilation modulaire d'un langage synchrone : spécification, simulation, implémentation et vérification d'applications synchrones : Daniel Gaffé, Annie Ressouche, Laboratoire LEAT, Université de Nice Sophia-Antipolis, CNRS , INRIA Sophia Antipolis Méditerranée*
[8] : *UbiComp Middleware and Verification : Annie RESSOUCHE*
[9] : *Introduction au Model Checking ENSTA, Sébastien Bardin, CEA,LIST, Laboratoire de Sûreté logicielle,24/10/2008*

Netographie

[10] : http://www.inria.fr/
(30/08/2014 à 20h21)
[11] :http://www-sop.inria.fr/members/Michel.Cosnard/ESSI/Annee-2005 2006/Programmation_Concurrente/PC06-Chap2.pdf
(30/08/2014 à 22h53) ;
[12] :http://en.wikipedia.org/wiki/Globally_asynchronous_locally_synchronous
(30/08/2014 à 20h37)
[13] : http://www.ubiq.com/
(30/08/2014 à 18h26)
[14] : http://fr.wikipedia.org/wiki/Model_checking
(30/08/2014 à 17h58)

Annexe

Nous allons présenter dans l'annexe un exemple qui explique mieux l'utilisation de nos beans (du générateur d'entrée, de l'automate du moniteur synchrone et du générateur de sortie), nous allons aussi présenter le format unique envoyé et reçu en utilisant l'exemple de grammaire A.

Pour cela, nous allons créer un exemple d'automate qui prendra des informations venues d'un capteur de position qui va indiquer si une personne est assise ou couchée, un capteur qui va détecter si la porte du réfrigérateur est ouverte ou fermée(true ou false) et d'un timer va envoyer la durée dans laquelle la porte du réfrigérateur est restée ouverte et qui va nous envoyer en sortie la valeur de l'alarme.

Nous avons créé cet automate à l'aide de galaxy. Il contient des signaux valués et des signaux non valués.
Nous avons trois signaux en entrées qui sont :

1- position : un signal de type « int » qui envoie deux valeurs (1 ou 0)
2- doorOpen : un signal booléen.
3- minute : un signal de type « int » qui envoie la durée du temps en minute.

Nous avons un signal de sortie qui est :

1- alarm : un signal de type int qui va envoyer la valeur de l'alarme selon plusieurs cas.

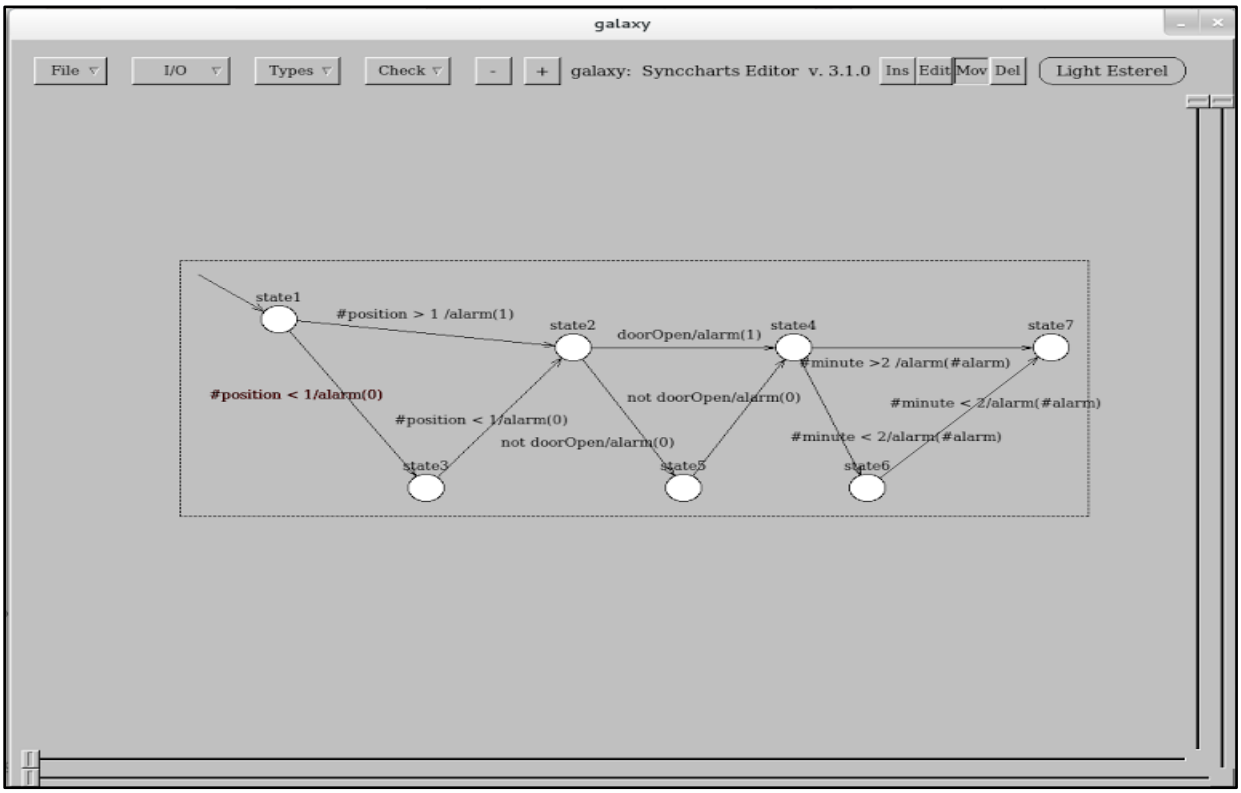

Figure 35 : Automate crée à l'aide de galaxy

Dans cet automate, si la valeur de la position est supérieure à 1, alors on met 1 dans le signal de l'alarme et on passe à l'état suivant, sinon on attribue la valeur 0 à l'alarme et on passe à l'état suivant.
Après, dans l'état suivant, nous vérifions l'état de la porte du réfrigérateur : si elle est ouverte, donc on met la valeur de l'alarme à 1 et on passe à l'état suivant, sinon on la met à 0 et on passe à l'état suivant.
Dans l'état qui suit, nous vérifions la durée d'ouverture de la porte, si elle est supérieure à 2 donc on envoie la valeur calculée de l'alarme sinon on la met à 0.(voir figure 35)

Après avoir créé cet automate, nous avons générer son code de bean c#, nous l'avons compilé et nous avons généré le bean . Nous allons en ce qui suit faire un assemblage de nos composants.

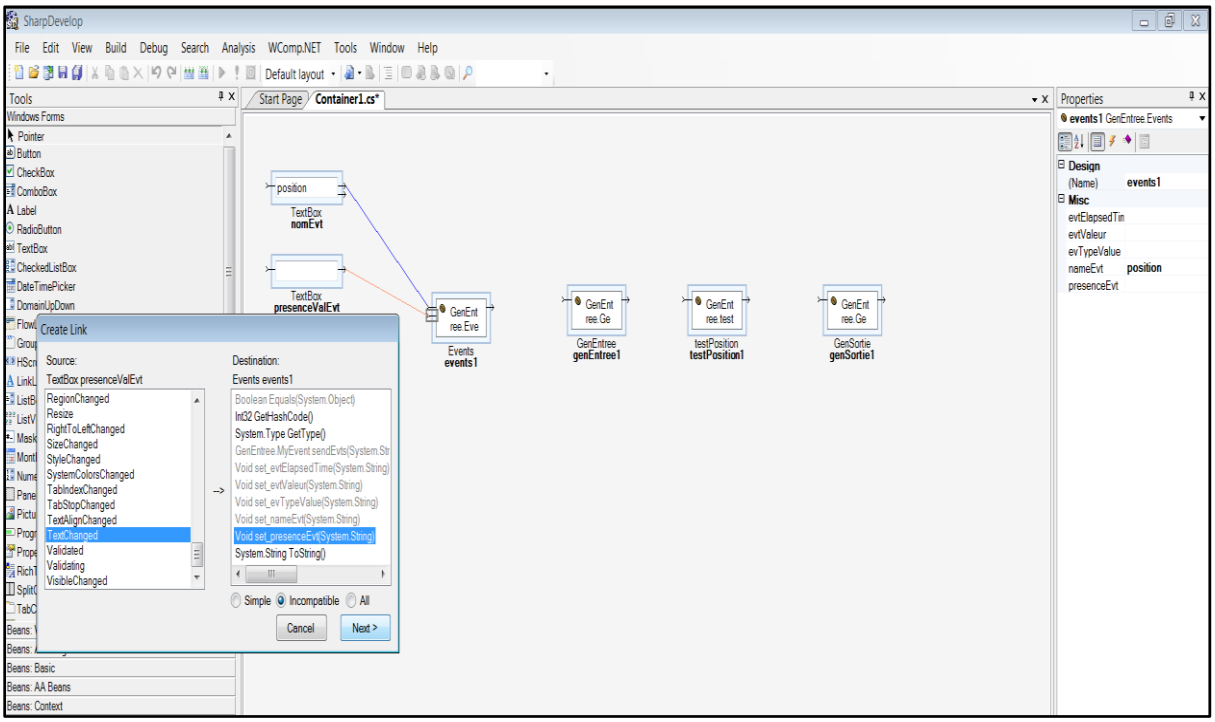

Figure 36: associer des valeurs d'un évènement

Vu que nous n'avons pas de capteur réel nous avons essayé de créer des évènements manuellement. Nous avons ajouté des textBoxs, nous avons fait la liaison nécessaire et nous avons changé les valeurs (voir figure 38).

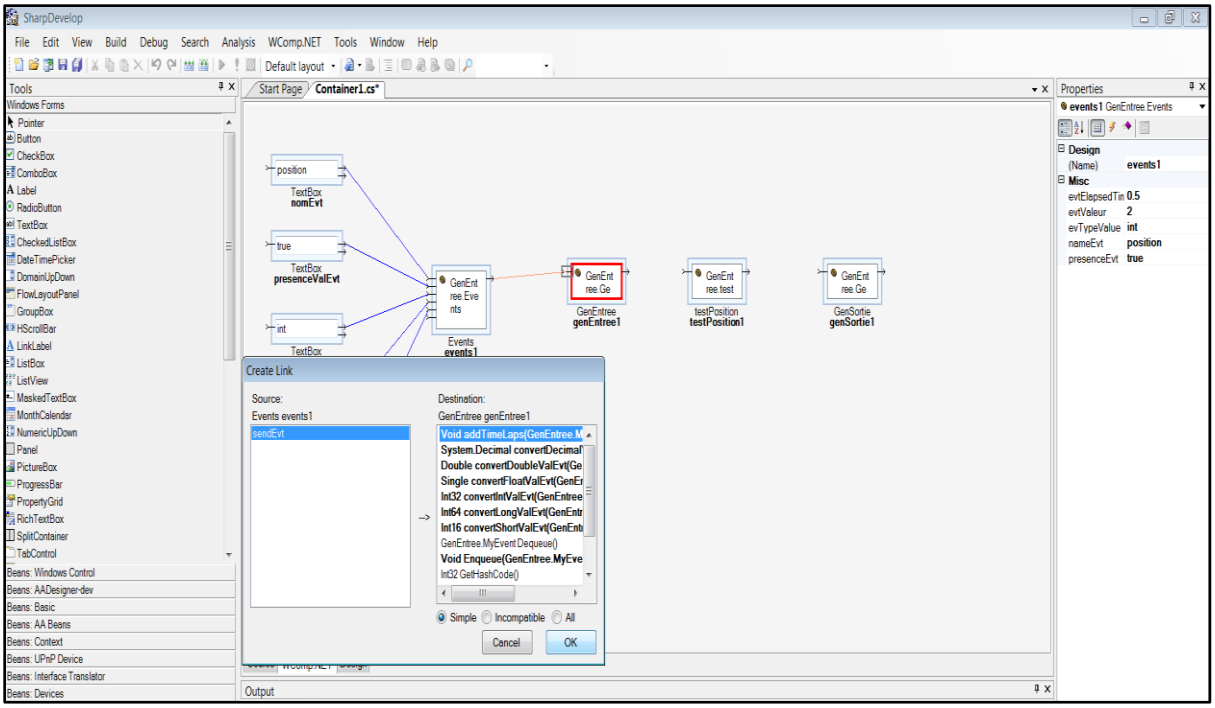

Figure 37: Liaison avec le générateur d'entrée

Après, nous avons lié l'évènement au générateur d'entrée, à l'aide de la méthode addElapsedTime qui va prendre la valeur « elapsedTime » de l'évènement et déclencher la fonction de l'ajout de cet évènement à chaque durée égale à cet « elapsedTime »(voir figure 37).

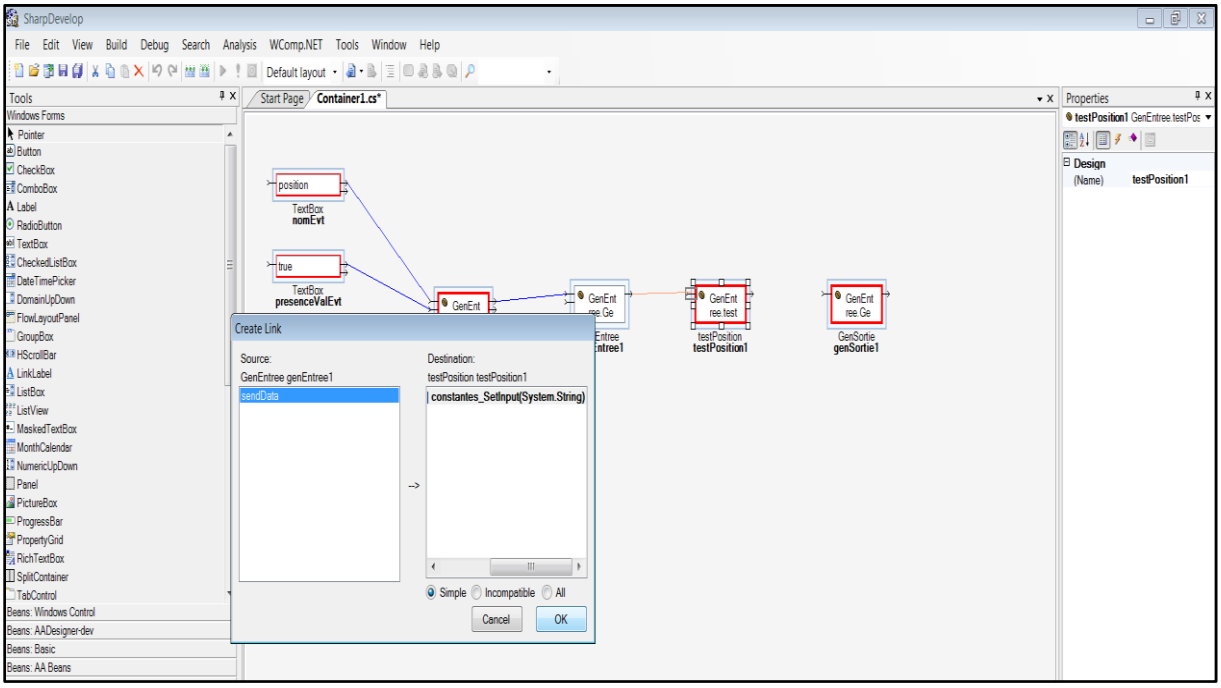

Figure 38: liaison générateur d'entrée/moniteur synchrone

Ensuite nous avons fait la liaison du générateur d'entrées avec notre bean de l'automate (moniteur synchrone), comme affiché, nous avons fait la liaison avec la méthode qui prend en input une chaine de caractère, cette fonction existe dans la catégorie simple, ce qui signifie que nous avons le même format d'entrée et de sortie (grâce à la sérialisation/désérialisation) (voir figure 38).

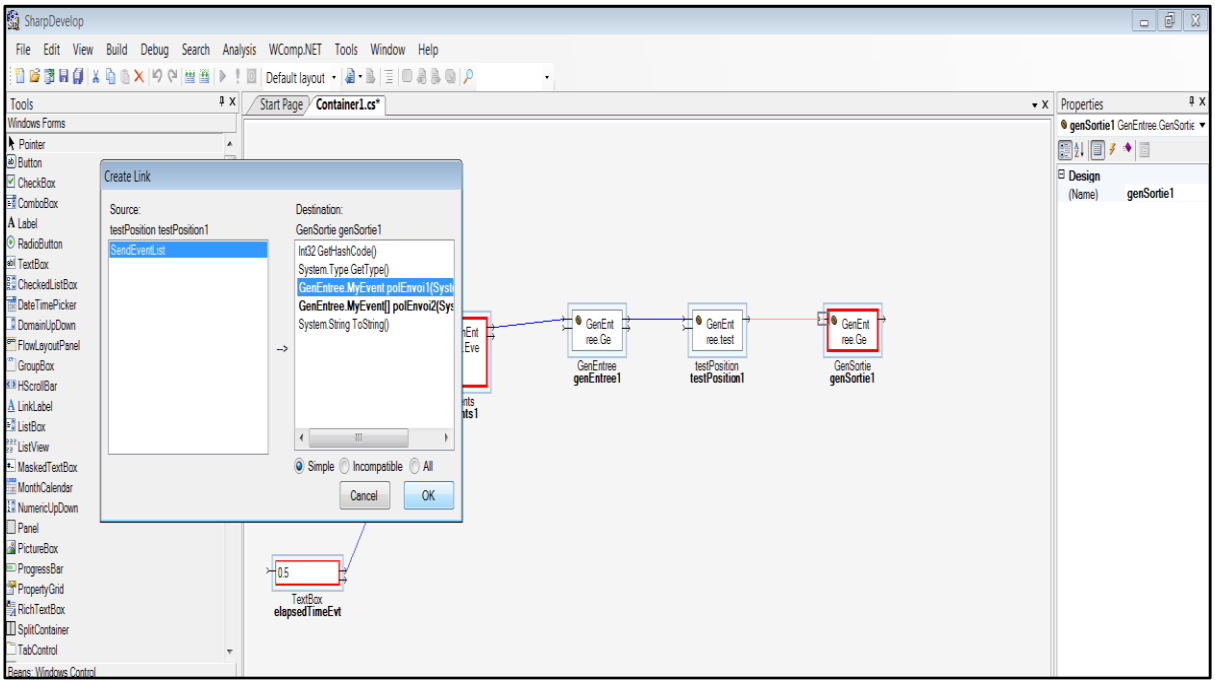

Figure 39: Liaison moniteur synchrone / générateur de sortie

Enfin, nous avons effectué la liaison entre le moniteur synchrone et le générateur de sortie, et nous avons choisi une politique d'envoi asynchrone (voir figure 39).Nous remarquons que le format de sortie du moniteur synchrone est le même celui d'entrée du générateur de sortie.

Vu que nous n'avons pas de composant critique pour lui envoyer l'évènement, pour tester, nous avons affiché le résultat dans un fichier.txt (voir figure 40 et 41)

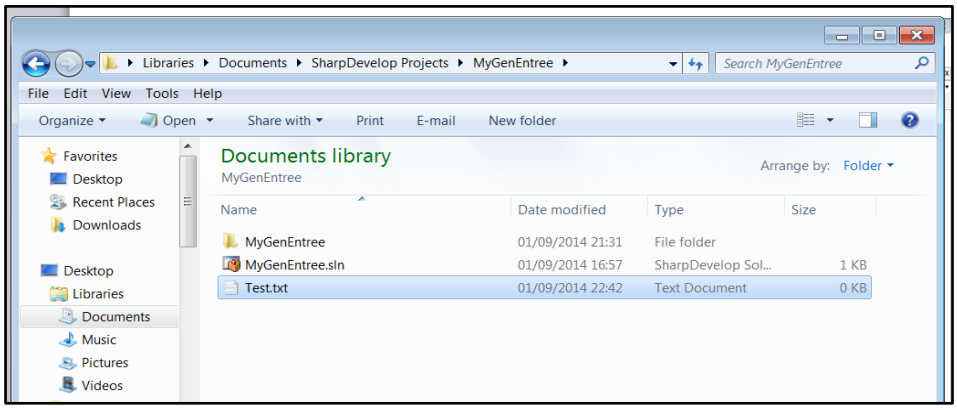

Figure 40: création du fichier de test

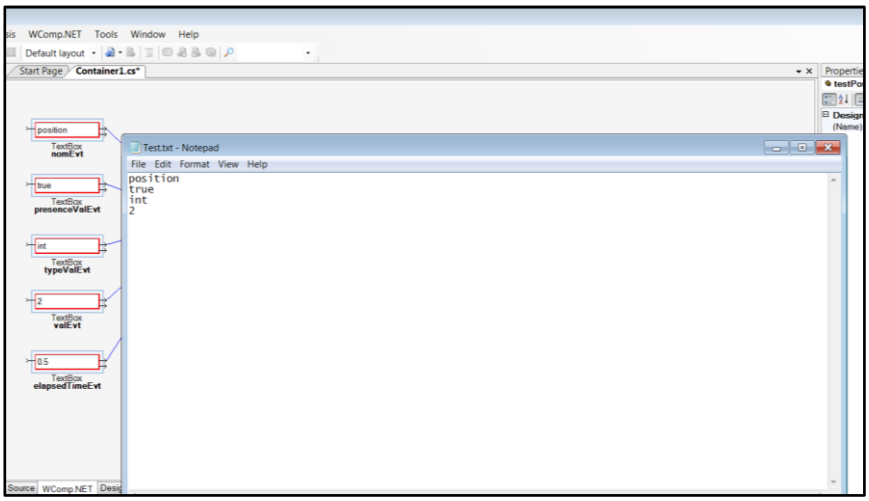

Figure 41: évènement envoyé