UNIVERSITÉ DE NANTES

Master 1 ALMA
2013 – 2014

Stage d'initiation à la recherche
Janvier - Avril 2014

PLUG'CHECK'RUN

Étudiants :
Coraline MARIE, Vincent RAVENEAU et Quentin MORICEAU

Encadrant :
Christian ATTIOGBE

14 avril 2014

Table des matières

Introduction

Afin de faire découvrir aux étudiants l'univers de la recherche en informatique, le master ALMA de l'Université de Nantes dispense un cours d'Introduction à la Recherche. Cette Unité d'Enseignement, effectuée lors du second semestre de master 1, a deux objectifs principaux. Tout d'abord, elle permet aux étudiants de mettre en oeuvre les capacités acquises au cours des différentes années d'étude de l'informatique. Ensuite, elle initie les étudiants au travail de recherche, réalisé en petits groupes encadrés par un enseignant-chercheur.

Le projet sur lequel nous avons travaillé, qui sera présenté dans ce rapport, a pour nom Plug'Check'Run. Il s'agit de contribuer à la construction d'une plateforme d'analyse formelle de composants logiciels et d'objets connectés, qui doit pouvoir répondre à trois objectifs. D'abord, elle doit permettre d'accueillir dynamiquement (Plug) de nouveaux composants et objets (selon son orientation vers des composants logiciels ou physiques). Ensuite, elle doit analyser (Check) leur structure et leur comportement. Enfin, elle doit utiliser (Run) les composants validés par l'analyse pour effectuer diverses tâches.

Notre rapport est structuré de façon à refléter la progression chronologique de nos travaux. Le plan est le suivant :
Nous présenterons dans une première partie l'état de l'art concernant notre projet. Une deuxième partie présentera ensuite les recherches que nous avons effectuées afin de prendre en main et d'explorer le sujet. Puis, dans une troisième partie, nous présenterons la façon dont nous avons cherché à répondre à la problématique du sujet. Ensuite, une quatrième partie présentera une généralisation basée sur les résultats des parties précédentes. Enfin, nous donnerons nos conclusions sur ce projet, ses résultats et ce qu'il nous a apporté.

État de l'art

Le premier objectif d'un travail de recherche est de maîtriser le sujet d'étude. Dans cette optique, nos premières recherches se sont concentrées sur la compréhension des objectifs de Plug'Check'Run et des différents outils utiles pour ce projet. Nous nous sommes également renseignés sur les actuelles avancées techniques de ce projet.

1.1 Études des automates

Plug'Check'Run est un projet visant à systématiser les connexions et la composition de composants représentés par leurs modèles formels. Dans le but de décrire le fonctionnement de cette idée, nous avons étudié différents types de représentations, afin de choisir celui qui illustrera le mieux notre travail de modélisation.

Ainsi, nous avons étudié les modélisations suivantes :
- Les automates à états fini ;
- Les réseaux de Petri ;
- Les automates de Mealy.

1.1.1 Les automates à états fini

Un automate est un graphe orienté ayant un nombre fini d'états, et dont les arcs sont étiquetés par des symboles. Il possède un état courant, ainsi que d'autres états par lesquels il est passé ou par lesquels il peut passer grâce aux changements d'état (que l'on appelle transitions). Un système peut par exemple se trouver dans un état initial et changer d'état en fonction de l'arrivée des signaux qui lui parviennent de son environnement.

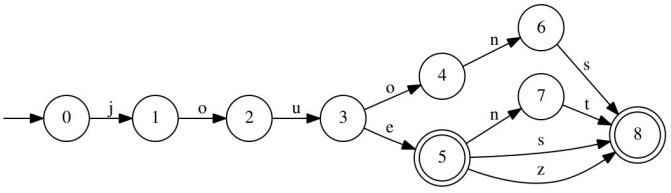

Figure 1.1 – Exemple d'automate à état fini

1.1.2 Les réseaux de Petri

Un réseau de Petri est une modélisation du comportement des systèmes dynamiques à événements discrets. Il est composé d'états, de transitions et d'arcs. Chaque état contient un nombre entier positif ou nul de jetons qui permettent de déterminer si une transition est franchissable ou non. Une transition est franchissable lorsque tous les états d'entrée de la transition contiennent au moins un jeton. Il ne peut jamais y avoir deux états ou deux transitions l'une après l'autre.

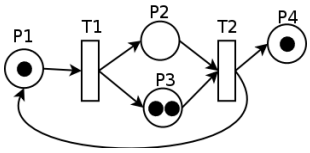

FIGURE 1.2 – Exemple de réseau de Petri

1.1.3 Les automates de Mealy

Un automate de Mealy est un automate fini pour lequel les sorties dépendent à la fois de l'état courant et des évenements d'entrée. Cela signifie que l'étiquette de chaque transition est un couple formé d'une lettre d'entrée et d'une lettre de sortie.

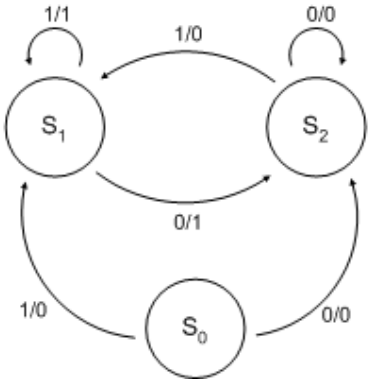

FIGURE 1.3 – Exemple d'automate de Mealy

1.2 Systèmes à composants

Un composant est une entité qui rentre dans la composition d'un système ; il peut être physique (matériel) ou logique (logiciel). Un composant logiciel est une entité (d'exécution) autonome fournissant des fonctionnalités ou des services à travers son interface.

Dans le domaine des composants logiciels exécutables, une définition largement admise est celle de Szyperski (1997)[8]. Cependant, de nombreux travaux et langages considèrent des composants abstraits, formels, et non exécutables directement.

Un système à composants est formé d'un ensemble d'éléments (appelés composants) interagissant entre eux et obéissant à certaines règles. Suivant la nature de leurs composants, ils peuvent être physiques ou logiciels. Un ordinateur est par exemple un système à composants physiques, mais un programme possédant une architecture à plugin est un système à composants logiciels, où les plugins sont les composants.

Aujourd'hui, la plupart des systèmes à composants sont capable de traiter l'ajout (PLUG) et l'utilisation (RUN) d'un nouvel élément. Cependant, l'analyse systématique d'un composant récemment ajouté (CHECK) n'existe pas encore réellement, et tenter d'ajouter un composant incompatible revient généralement à faire planter le système ou à faire échouer le processus en cours.

Le projet PLUG'CHECK'RUN a donc plusieurs objectifs. D'une part, il a pour but de permettre de modéliser des systèmes à composants, de telle sorte qu'ils puissent s'installer, s'analyser et fonctionner automatiquement. D'autre part, il vise à réduire au maximum les erreurs de compatibilité. Le but final étant d'arriver déterminer les besoins d'un environnement favorable à la mise en place d'un système répondant aux caractéristiques définies par le sujet.

Prise en main du sujet

Toujours dans l'optique de maîtriser notre sujet d'étude, l'étape suivante de notre travail s'est révélée être la recherche d'exemples de systèmes qui s'approchent du fonctionnement voulu dans le cadre du projet Plug'Check'Run. Nous avons ensuite illustré ceux de ces modèles qui nous semblaient les plus pertinents.

2.1 Exemples de modélisations

2.1.1 Ordinateur et périphériques externes

Le premier exemple que nous avons trouvé est l'ajout de périphériques externes (vidéoprojecteurs, imprimantes, controleurs, etc ...) à un ordinateur. En effet, les systèmes les plus récents offrent la possibilité d'installer de façon semi-automatique de nouveaux composants. Dans les faits, il suffit qu'un équipement, comme une souris, soit branchée sur l'ordinateur pour qu'elle fonctionne automatiquement, et ce sans installation manuelle ou autre intervention humaine que le branchement du périphérique. Malheureusement, certains périphériques sont encore difficiles à ajouter aussi facilement.

Nous avons choisi cet exemple car il utilise les côtés Plug et Run de notre projet. En effet, le système détecte d'abord la connexion d'un nouveau périphérique (par branchement ou par réception d'un signal) : Plug. L'ordinateur installe le pilote fourni, sans effectuer de vérification, et plante si l'action est impossible. Enfin, il utilise les différentes fonctionnalités du périphérique ajouté : Run.

2.1.2 Un IDE intelligent

En ce qui concerne le second exemple, nous avons choisi celui de l'IDE[1] fonctionnant avec des plugins additionnels. Cet exemple logiciel utilise aussi les côtés Plug (demande d'ajout de plugin) et Run (utilisation du plugin ajouté) de notre projet.

1. Integrated Development Environment (environnement de développement intégré) : ensemble d'outils permettant d'augmenter la productivité des programmeurs qui développent des logiciels

2.1.3 La télécommande universelle

Le dernier exemple que nous avons choisi est la télécommande universelle. En effet, ce modèle utilise encore les côtés PLUG et RUN de notre projet car il ajoute de nouveaux composants, puis les contrôles. L'aspect CHECK est encore généralement absent de la télécommande car elle est incapable d'installer un nouveau composant sans utilisateur et de corriger ses problèmes de connexions. À la place, si elle capte l'équipement, celui-ci est ajouté, sinon aucune action n'est effectuée. En revanche, certaines implémentent des mécanismes de collecte d'information qui peuvent s'apparenter à une ébauche de CHECK.

Cet exemple est vraiment la base de notre réflexion. En effet, nous nous sommes rendu compte qu'ils formaient une bonne base pour le projet PLUG'CHECK-'RUN. C'est pour celà que nous avons détaillé son fonctionnement plus en détails, et notamment les façons dont elle peut acquérir certaines informations sur son environnement et ses composants.

2.2 Étude détaillée : la télécommande universelle

La télécommande universelle est une télécommande capable de contrôler n'importe quel appareil à sa portée. Pour cela, il faut au préalable procéder à une installation de l'équipement à piloter. Cette installation est manuelle, et peut se faire de plusieurs façon différentes.

2.2.1 Bases de données (télécommandes pré-programmées)

La plupart des télécommandes universelles possèdent une liste d'appareils pilotables, classés le plus souvent par marques et par modèles. L'identification se fait grâce à un code inscrit sur l'appareil. Une fois ce code entré, la télécommande récupère les informations dans sa mémoire interne et paramètre ses touches. Si aucun appareil de la base ne correspond à celui à installer, il est presque toujours possible d'engager un mode de recherche.

2.2.2 Le mode recherche

Lors d'une installation en mode recherche, la télécommande teste chaque code de sa base de données (elle peut comprendre jusqu'à 15000 marques différentes). Pour cela, elle émet un signal infrarouge vers l'appareil avec l'ordre de l'éteindre. Si l'appareil ne s'éteint pas, elle recommence avec un autre code. Dès que l'extinction s'effectue, l'appareil est identifié.

2.2.3 L'apprentissage d'une autre télécommande

Devant la multiplication des références d'appareils à piloter, les constructeurs de télécommandes ont ajouté une option permettant à leur télécommande universelle d'apprendre les codes d'une autre télécommande. Pour cela, il faut pointer la télécommande universelle vers la télécommande dont on veut récupérer les codes. Le transmetteur infrarouge envoie une demande d'information. Le récepteur de la télécommande ciblé délivre alors l'ensemble des codes infrarouges attribués à chacune de ses touches. Le récepteur de la télécommande universelle

réceptionne ces codes, les assigne aux touches correspondantes et enregistre le tout dans sa mémoire. Cette opération peut être répétée pour chaque télécommande dont on souhaite recopier les fonctions. La mémoire de la télécommande universelle stocke les informations relatives à chaque modèle de télécommande cloné.

2.2.4 Bases de données évolutives

Pour éviter de devoir ajouter les fonctionnalités d'apprentissage, d'autres constructeurs ont préféré permettre à l'utilisateur d'obtenir la mise à jour de la base de données de la télécommande par Internet, ou directement par téléphone via un modem audio intégré : il suffit de plaquer la télécommande sur le combiné pour récupérer les codes manquants.

Mise en œuvre et modélisation d'un système à composants

3.1 Outils

Pour mener à bien notre projet, nous avons eu besoin de modéliser certains aspects des systèmes sur lesquels nous étions en train de travailler. Pour ce faire, nous avons utilisé différents logiciels, que nous allons présenter dans cette partie.

3.1.1 UPPAAL

UPPAAL[1] est un logiciel développé par les universités d'Uppsala et d'Aalborg. Il permet de modéliser des systèmes composés d'automates, ainsi que de simuler leur fonctionnement. Cette simulation peut être effectuée de deux façons, soit en mode pas à pas avec des entrées manuelles, soit en générant automatiquement des entrées aléatoires parmis les différentes valeurs possibles.

Lors d'une modélisation avec UPPAAL, il faut distinguer deux types d'éléments, les systèmes et les processus. Les processus sont les sous-parties du système à modéliser, qui sont mis en place dans des projets différents. Un système regroupe quand à lui plusieurs processus, qui vont communiquer en s'envoyant des messages à travers des canaux. Un système peut également contenir plusieurs systèmes, si sa complexité le demande.

Cet outil nous a servi à tester le comportement des systèmes que nous avons mis au point. La simulation automatique a notamment permis de mettre en évidence des cas que nous n'avions pas pensé à gérer, ou qui étaient incorrectement traités.

3.1.2 Dia

Dia[2] est un logiciel de création de diagramme, conçu dans le cadre du projet GNOME[3]. Il permet de créer différents types de diagrammes (circuits électriques, diagrammes de flux, ...), et peut servir dans le cas d'un diagramme

1. http://www.uppaal.org/
2. https://wiki.gnome.org/Apps/Dia
3. GNOME (GNU Network Object Model Environment) : environnement de bureau libre convivial dont l'objectif est de rendre accessible l'utilisation du système d'exploitation GNU au plus grand nombre.

UML à générer du code (C, C++, ...) associé.

Pour ce projet, seule la fonction permettant de réaliser des diagrammes de séquence a été utilisée, afin d'illustrer le fonctionnement de diverses parties des systèmes sur lesquels nous travaillions.

3.2 Modélisation du système

Dans cette partie, nous allons présenter la structure et le fonctionnement de systèmes à composants répondant aux objectifs visés par `Plug'Check'Run`. *Cette présentation sera illustrée en annexe par les différentes parties d'un système répondant aux caractéristiques énoncées dans cette partie, que nous avons modélisées à l'aide du logiciel UPPAAL.*

3.2.1 Environnement et portée du système

Un système à composants agit selon une portée qui définit son environnement. Cette contrainte est présente, que le système ait une portée physique (un rayon d'action par exemple, exprimé dans une unité de distance) ou une portée abstraite (la restriction aux logiciels ayant une caractéristique précise par exemple).

La portée est un élément capital du système, puisqu'elle définit si un élément sera capable de communiquer avec le système, indépendamment de toute notion de compatibilité. En effet, toute tentative de communication d'un composant situé hors de l'environnement défini par cette portée sera immédiatement rejetée.

La portée peut être implémentée de deux manières différentes. Tout d'abord, elle peut former une zone physique, par exemple en donnant ses limites, ou un point et un rayon. Elle peut également mettre en place une contrainte logicielle, telle que la position de l'exécutable du composant sur un disque ou le respect d'une interface. Il est important de noter que le choix de la façon dont la portée est implémentée impacte fortement la structure des composants. En effet, ceux-ci devront être capables de fournir au système les informations nécessaires pour déterminer s'ils sont contenus dans son environnement ou non.

3.2.2 Fonctionnement du système

Le système doit être en permanence à l'écoute de son environnement et capable de communiquer avec ses composants. Il doit être capable de déterminer quels composants sont à l'écoute et comment utiliser ceux qui sont connectés. Il doit aussi être attentif à tout signal envoyé par un nouveau composant potentiel afin de déclencher les tests déterminant s'il doit être ajouté ou non à la liste des composants connectés. L'ajout d'un autre équipement pendant la phase de test pouvant compromettre l'intégrité du résultat, il est préférable qu'un seul test ait lieu à la fois. En effet, les tests doivent prendre en compte l'intégralité des composants connectés au système. Par conséquent, des composants testés en parallèle ne peuvent se prendre en compte mutuellement sans faire de supposition sur le résultat du test de l'autre composant. Le système doit également être capable de gérer les composants auxquels il accepte de se connecter, afin

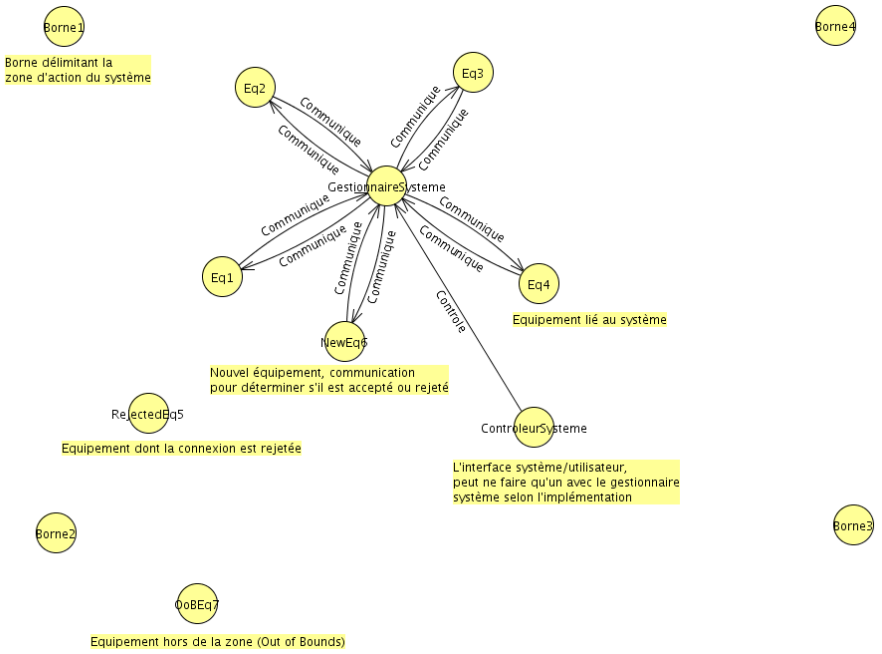

FIGURE 3.1 – Exemple de système à environnement physique

de pouvoir déterminer s'il se trouve dans un état optimal (tous les composants fonctionnent de façon correcte) ou si des erreurs sont présentes.

Détection de composants — PLUG

Le système doit être en mesure de détecter tout composant présent dans son environnement, afin de pouvoir déterminer s'il doit l'ajouter ou non à sa liste de composants connectés. Cette détection peut être implémentée de différentes façons, que l'on peut regrouper en deux catégories :

- Les implémentations où le système cherche en permanence si de nouveaux équipements sont disponibles. Dans ce cas de figure, les composants sont passifs et attendent un signal du système leur demandant les informations dont il a besoin pour lancer la phase d'analyse. Ce signal est envoyé à intervalles réguliers par le système à tous les composants inconnus présents dans l'environnement.
- Les implémentations où le système est passif vis à vis des nouveaux composants. C'est alors à eux d'envoyer au système un signal indiquant qu'ils souhaitent se connecter.

Une fois qu'un nouveau composant est détecté, le système passe en phase de test afin de déterminer s'il peut intégrer ce composant ou non.

Test d'intégration de composants — CHECK

Pour tester si un nouveau composant peut être intégré au système ou non, une série de tests est lancée. Celle-ci se compose de trois étapes successives,

afin de pouvoir gérer le maximum de cas possibles tout en offrant un moyen d'optimiser les cas les plus fréquents.

La première étape est la comparaison du composant avec les équipements connus. Pour ce faire, le système dispose de plusieurs façons de récupérer les équipements déclarés comme compatibles. Celles-ci sont classées par ordre de priorité, afin de déterminer l'ordre dans lequel les utiliser. Comme pour tout autre composant du système, leur implémentation peut prendre plusieurs formes, comme par exemple la consultation d'une base de données ou l'envoi d'une requête à un site web. Dans tous les cas, le résultat de cette étape prend une valeur parmi les trois suivantes :

- *Composant valide* : dans ce cas le composant est reconnu comme compatible avec le système, et est ajouté sans effectuer de tests supplémentaires. Il passe donc directement dans l'état RUN.
- *Composant invalide* : dans ce cas le composant est reconnu comme incompatible avec le système, et est rejeté immédiatement. Selon les systèmes, le composant peut alors être marqué comme ayant été rejeté, afin de lui retirer la capacité à redéclencher un test d'intégration au système.
- *Composant inconnu* : dans ce cas, le système ne dispose pas de suffisamment d'informations sur le composant pour l'accepter ou le rejeter immédiatement. Il passe donc à l'étape suivante du test.

L'objectif du projet étant de mettre au point des systèmes dont les composants interagissent entre eux sans causer de problèmes, ce test devra prendre en compte la combinaison de composants présents. Si ce n'est pas le cas, et que seul le nouveau composant est examiné, la majorité des résultats obtenus auront la valeur *Composant inconnu*. En effet, seuls les composants ne créant strictement aucune interaction avec le système pourront être acceptés sans plus de vérifications.

La seconde étape consiste à tester le composant, et plus particulièrement s'il respecte la définition d'un composant donnée par le système. La façon dont cette étape est réalisée dépend de l'implémentation du système et des contraintes qui y sont liées. Par exemple, si les composants sont caractérisés par l'implémentation d'une interface particulière, c'est sur son existence que portera le test. Si ce test est positif, alors le composant passe à la dernière étape du test, sinon le composant est rejeté.

La dernière étape du test est celle qui caractérise le projet PLUG'CHECK-'RUN, puisqu'elle consiste à tester les interactions entre le nouveau composant et ceux déjà présents dans le système. La façon dont ce test sera effectué dépend une fois de plus des choix faits lors de l'implémentation du système, en particulier des contraintes exprimées sur la gestion des composants. Ainsi, si le système requiert une communication permanente avec ses composants, ceux-ci devront pouvoir répondre en toute circonstance, quel que soit leur état. Une telle vérification peut également prendre plusieurs formes, comme par exemple la modélisation du comportement du composant sous forme d'un automate qui sera ensuite fourni au système. Le système pourrait alors réaliser le produit de cet automate avec celui représentant son comportement, et comparer le nouvel automate ainsi obtenu avec un modèle à respecter. Il pourrait également s'assurer qu'il a de bonnes propriétés, qui correspondent aux contraintes prédéfinies

par le système.

Les tests effectués lors de cette étape peuvent être multiples, et leur nombre est généralement proportionnel au nombre de contraintes imposées au système. Une fois tous les tests effectués et positifs, le composant est ajouté au système, et passe donc dans l'état RUN. Si au moins un test n'est pas validé par le composant, celui-ci est rejeté par le système.

Gestion des composants connectés — RUN

Une fois que le système a accepté la connexion de composants, il lui faut pouvoir interagir efficacement avec eux lorsque nécessaire. Les besoins en terme de dialogue étant variables d'un système à un autre, il est nécessaire de les définir lors de la conception, et d'implémenter des tests ayant pour but d'assurer que tout composant connecté vérifie ces contraintes. Celles-ci peuvent être définies avec des niveaux de priorité différents, afin de permettre au système d'avoir une réaction la plus adaptée possible en cas de défaillance soudaine d'une partie de ses composants. Par exemple, si les contraintes *"Tous les éléments doivent pouvoir communiquer en permanence"* et *"Toute communication doit prendre moins de 50ms"* sont présentes, le non respect de la première doit déclencher une erreur, alors que la seconde peut ne lever qu'un avertissement.

En plus de mémoriser ses composants et de les utiliser lorsque nécessaire, le système doit donc continuellement vérifier chaque contrainte qui lui a été imposée.

Généralisation

Lors de l'étude des différents systèmes décrits précédemment, nous avons pû mettre en évidence les aspects qui nous paraissent importants pour définir un environnement permettant de mettre en place un système répondant aux objectifs de `Plug'Check'Run`.

Pour rappel, ces objectifs sont d'offrir un système permettant d'ajouter et de retirer différents composants, tout en assurant une analyse systématique de ceux-ci. Cette analyse est destinée à éviter les problèmes pouvant intervenir lors des interactions entre les différents composants du système.

Il est donc important pour le système de pouvoir effectuer trois actions particulières.

La première est de pouvoir utiliser les composants qui sont connectés à lui.

La deuxième est de pouvoir déterminer si un nouveau composant est un candidat possible pour être ajouté au système ou non.

La troisième est de pouvoir effectuer les tests sur les interactions entre les nouveaux composants et ceux qui sont déjà connectés au système. Pour ce faire, les composants doivent pouvoir fournir au système un moyen de simuler leur fonctionnement associé à celui de ses autres composants.

Les qualités attendues des composants doivent pouvoir être définies, de même que les contraintes que le système doit respecter dans son fonctionnement. Celles-ci peuvent être ordonnées par priorité, afin de résoudre certains cas conflictuels.

Conclusion

Dans le cadre du cours d'*Initiation à la Recherche*, nous avons réalisé une étude sur la fiabilité et la sûreté des systèmes à composants. Pour ce faire, nous avons travaillé sur une large gamme d'automates (automates à états fini, automates de Mealy, ...), ainsi que déterminé les besoins actuels sur les systèmes à composants.

Ce travail nous a amené à réfléchir sur les systèmes à composants actuels, leur qualités, leur défauts et ce qu'il faudrait améliorer. Nous avons ainsi recherché de nouvelles techniques, qui permettraient de mettre au point un système à composants fiable et sûr. Puis, à partir de nos réflexions, nous avons réalisé avec UPPAAL une modélisation d'un système à composants qui répond à notre analyse. Celui-ci n'est cependant pas exempt de points améliorables, tels que l'impossibilité de tester plus d'un nouveau composant éventuel à la fois.

Ainsi, l'évolution logique du projet Plug'Check'Run serait de permettre de fabriquer un système à composant sûr, évolutif, tant au niveau matériel que logiciel. Il faudrait ensuite étendre cette modélisation à tous les sytèmes à composants existant actuellement, dans le but de rendre le monde plus fiable.

Bibliographie

[1] Tom's Guide. *Le guide Télécommande Universelle*. 2013. http://www.tomsguide.fr/guide/comparatif-Telecommande-Universelle,2-aWRHdWlkZT03Ng==.html.

[2] J. Christian Attiogbé. *Modélisation de la dynamique / Réseaux de Petri*. 2012. http://pagesperso.lina.univ-nantes.fr/info/perso/permanents/attiogbe/mespages/MSFORMEL/IUT/cours_rdp.2x1.pdf.

[3] F. Barthélemy. *Notes de cours sur les automates (NFP108)*. janvier 2012.

[4] Gerd Behrmann, Alexandre David, and Kim G. Larsen. *A Tutorial on Uppaal 4.0*. novembre 2006. http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf.

[5] Kiyomi Hirano and T. Fujikura. *uppaal 4.0 : small tutorial*. novembre 2009. http://www.it.uu.se/research/group/darts/uppaal/small_tutorial.pdf.

[6] Z. Mammeri. *Systèmes de transitions : Automates à états finis*. avril 2011. Université Paul Sabatier.

[7] Marc Pouzet. *Circuits et composition synchrone d'automates*. octobre 2013. http://www.di.ens.fr/~pouzet/cours/mpri/cours4/automates.pdf.

[8] Gough John Smith Glenn and Szyperski Clemens. *Discovery and use of independently developed objects in distributed systems. Eighth IFIP/IEEE International Workshop on Distributed Systems Operations and Management (DSOM'97)*, octobre 1997.

[9] Jacques TICHON, Christian COUWENBERGH, Rudi GIOT, and Salvador GARCIA ACEVEDO. *Communication avec les périphériques* . septembre 2001. http://www.techniques-ingenieur.fr/base-documentaire/electronique-automatique-th13/systemes-d-information-et-de-communication-42397210/communication-avec-les-peripheriques-s8590/.

[10] Stéphane Vialle. *Les réseaux de Petri*. mai 2007. http://www.metz.supelec.fr/metz/personnel/vialle/course/CNAM-ACCOV-NFP103/extern-doc/RdP/Introduction-RdP.pdf.

Table des figures

Annexes

Modélisation d'un système complet

Dans le but de simplifier la compréhension de notre résultat, voici la modélisation d'un système à composants dans un environnement quelconque.

Ce système est composé d'un élément principal, qui fera office de cœur et d'un composant.

1. La partie PLUG du système se compose d'un simple module d'écoute, qui envoie de nouveaux composants à la partie CHECK (demande !, reponse ?)
2. La partie CHECK du système, quant à elle, est réalisée par différents contrôles effectués sur le composant à ajouter. Pour cela, celui-ci est d'abord comparé à la base de données (interne ou en ligne) du système, afin de vérifier sa compatibilité (bddCheck !, composantConnu ?, composantInconnu ?)
3. Si le composant n'est pas identifié, le système passe à une phase de test plus poussée, où il essaye de déduire le comportement de l'élément (éventuellement son automate) (compatibleCheck !, composantCompatible ?, composantIncompatible ?)
4. À partir du comportement récupéré, une autre phase de test simule le fonctionnement des composants et vérifie s'ils sont bien intégrés, et ce quelles que soient les actions effectuées sur le système (integrationCheck !, composantIntegrable ?, composantNonIntegrable ?)

Pour effectuer ce dernier test nous avons pensé à l'utilisation d'un outil semblable à UPPAAL, mais celui ci, bien que pratique pour tester le modèle d'un scénario précis, n'est pas très adapté à la modélisation d'un système en temps réel. Il faudrait donc soit l'adapter pour accepter l'ajout de composants à la volée, soit changer le protocole pour aller vers quelque chose de plus statique (pas d'ajout pendant la phase de test). Il a également manqué un outil qui permettrait d'identifier une potentielle erreur afin de la localiser et de déterminer s'il faut interdire au composant de se connecter au système, ou de lui demander de changer son comportement face à la situation qui pose problème.

Dans la partie RUN, le système effectue les actions demandées par l'utilisateur et reste en permanence au courant de l'état de ses composants(hors zone, en veille,...) via un module de communication. Nous avons modélisé la partie

communication du système. (Figure 4.2)

Dans les figures suivantes, les automates ont été modélisés avec UPPAAL. *Les notations* **xxx!** *correspondent à l'envoi d'un signal* **xxx** *vers un composant ou un autre module du système. Quant aux notations* **yyy?**, *il s'agit de la captation d'un signal* **yyy**. *La transition ne peut s'effectuer que s'il y a émission et reception du signal au même moment.*

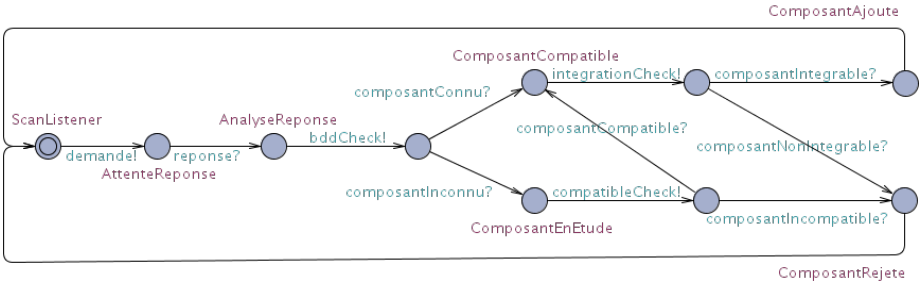

FIGURE 5.1 – Processus d'ajout et de test d'un composant

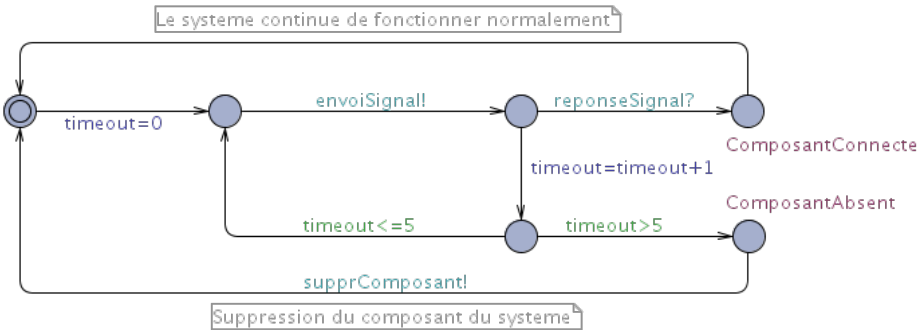

FIGURE 5.2 – Processus d'écoute permanente des composants

Organisation du travail

Le travail effectué se découpe en quatre grands axes. Dans un premier temps nous avons défini l'état de l'art et trouvé des exemples de mise en action du sujet. Puis nous avons commencé à émettre des hypothèses. Enfin, nous sommes passés à la modélisation et avons développé les résultats qui en ont découlé. Nous avons finalement écrit le présent rapport.

Dans la première étape nous nous sommes renseignés sur les différentes méthodes déjà existantes de gestion de systèmes à composants (du type *plug and play*) d'un côté, et de l'autre nous avons cherché la meilleure méthode pour modéliser le comportement d'un composant par un automate. Pour ce faire, nous avons comparé ce qu'offrent les différents types d'automate.

À partir de là nous avons posé des hypothèses sur les différents mécanismes d'ajout de composant, tels que la façon dont ces composants son connus par le système, la forme que doivent prendre les différents tests d'integrations, ou encore le fonctionnement global du système.

Une fois que des hypothèses ont été formulées, il nous a fallu les modéliser afin de déterminer leur validité. Cette phase de modélisation s'est essentiellement effectuée grâce à des outils tels qu'UPPAAL.

Nous avons ensuite fini par la rédaction de ce rapport.

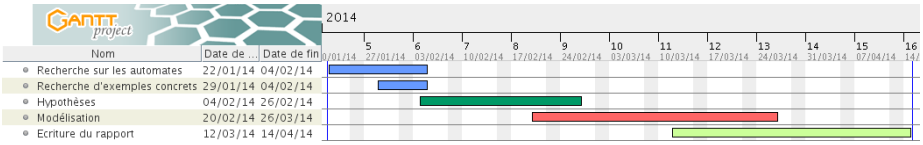

FIGURE 5.3 – Diagramme de Gantt

Tout au long de ce projet, nous avons cherché à nous répartir le travail à effectuer, dans le but de profiter au maximum du temps qui nous était imparti.