الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

Université 20 Août 1955- Skikda
Faculté des Sciences
Département d'Informatique

جامعة 20 أوت 1955
كلية العـلوم
قسم : الإعلام الآلـــي

Thèse

En vue de l'obtention du diplôme de

Doctorat de 3° cycle (LMD) en Informatique

Option : ***Computation et Cognition des Systèmes Informatiques***

Test des Systèmes Multi Agents dans les Environnements d'Intelligence Ambiante

Présentée par :

Sara KERRAOUI

Soutenue publiquement : 22/ 01 / 2019

Devant la Commission du Jury composé de :

JURY

M. Bachir BOUCHEHAM,	Professeur,	Université 20 Août 1955-Skikda,	Président
M. Ramdane MAAMERI,	Professeur,	Université A.Mehri Constantine 2,	Examinateur
M. Mohamed BENMOHAMED,	Professeur,	Université A.Mehri Constantine 2,	Examinateur
M. Smaine MAZOUZI,	MC 'A',	Université 20 Août 1955-Skikda,	Examinateur
M. Mohammed REDJIMI,	Professeur,	Université 20 Août 1955-Skikda,	Directeur de thèse
M. Yacine KISSOUM,	MC 'B',	Université 20 Août 1955-Skikda,	Invité

Remerciements

Je tiens tout d'abord à exprimer ma profonde gratitude à mon co-directeur de thèse M. Yacine KISSOUM, Maître de conférence à l'Université 20 Août 1955-Skikda pour son encadrement et pour l'encouragement et l'intérêt qu'il m'a apporté pour l'accomplissement de ce travail et surtout pour son grande aide et ses qualités humaines.

Je remercie vivement mon directeur de thèse M. Mohammed REDJIMI, Professeur à l'Université 20 Août 1955-Skikda pour son aide et ses conseils judicieux et constructifs, ainsi que sa disponibilité permanente et son soutien moral.

Je tiens également à adresser mes remerciements à Monsieur Bachir BOUCHEHAM, professeur à l'Université 20 Août 1955-Skikda pour avoir accepté de présider le jury de thèse.

Je tiens aussi à exprimer tous mes remerciements à Monsieur Ramdane MAAMERI, Professeur à l'Université A.Mehri Constantine 2, Monsieur Mohamed BENMOHAMED, Professeur à l'Université A.Mehri Constantine 2, Monsieur Smaine MAZOUZI, Maître de Conférence à l'université 20 Août 1955-Skikda, d'avoir bien voulu examiner mon travail en acceptant de faire partie du jury de ma soutenance.

Je remercie mes collègues Moussa SAKER et Med Lamine BOUGHAOUAS pour leur aide à accomplir mon travail.

Je remercie ma collègue et très chère amie Soumia MAMERI Pour son sincère amitié et confiance, et a qui je dois ma reconnaissance et mon attachement.

Un remerciement particulier à Ma Mère qui, de par son encouragement inconditionnel, m'a permis d'atteindre ce but.

Je voudrais remercier vivement mon père, mes beaux parents, mon mari, mes sœurs, mes nièces et neveux, tous mes amis, pour leur soutien moral très précieux et inestimable tout au long de ces années.

Dédicace

A Dieu Le Tout Miséricordieux

Ton amour, Ta miséricorde et Tes grâces à mon endroit m'ont fortifiée dans la persévérance et l'ardeur au travail.

A mon Père

En vous, je voie un père dévoué à sa famille. Ta présence en toute circonstance m'a maintes fois rappelé le sens de la responsabilité.

A ma Mère

En vous, je voie la maman parfaite, toujours prête à se sacrifier pour le bonheur de ses enfants.

Merci pour tout.

A mon Mari

Ton amour et ta sollicitude à mon égard me marqueront à jamais.

A mes deux Anges : YAHIA et AMEL

Que dieu vous protège, je vous aime.

A mes sœurs et beaux frères, mes nièces et mes neveux

Qui je le sais ma réussite est très importante pour vous. Que Dieu vous paye Pour tous vos bienfaits.

SARA

Résumé

Les systèmes ambiants forment un domaine d'application très particulier, car ils sont très dynamiques et en interactions constantes avec des utilisateurs. Les systèmes multi-agents sont utilisés comme abstractions utiles dans ces systèmes, par exemple pour les dispositifs et les fonctionnalités, et comme paradigme pour la mise en œuvre. Cependant, les caractéristiques de tels systèmes rendent leur test un vrai défi qui demande de nouvelles méthodes de test efficaces et adéquates afin d'évaluer les comportements autonomes des agents ambiants et leurs interactions.

Cette thèse aborde le problème des tests basés sur les modèles formels pour les systèmes multi-agents dans les environnements d'intelligence ambiante par le biais d'une approche qui couvre tous les niveaux d'abstraction du paradigme agent. Ces niveaux d'abstraction sont : le niveau agent, le niveau société d'agents et le niveau système. L'approche proposée s'intègre dans l'éthique du test basé sur les modèles en proposant de mettre en œuvre un processus intégrant la gestion des exigences, l'automatisation de la génération des cas de test et leur concrétisation.

En particulier, nous proposons en premier lieu, un système à base d'agents pour la modélisation d'un environnement ambiant destiné aux personnes âgées utilisant les réseaux de références. L'exemple a été modélisé en respectant l'architecture MULAN (multi agent system net) augmentée de JESS (Java Expert System Shell) pour la prise en compte des aspects délibératifs des agents ambiants. De plus, le modèle proposé tient compte de l'aspect temps pour toute décision. Deuxièmement, nous proposons une approche de test basée sur les modèles (réseaux dans les réseaux) qui s'intègre à l'outil JUNIT pour la génération et le test des systèmes multi-agents. Contrairement aux différentes approches de test, où l'instrumentation porte, la plupart du temps, sur le code source de l'application, notre approche laisse intact l'application sous test en faisant l'instrumentation au niveau du modèle, permettant ainsi de préserver son comportement initial (l'aspect fonctionnel ou non-fonctionnel ne sera pas modifié).

Mots-clés : Agent, Systèmes Multi-agents, Intelligence Ambiante, Test basé modèle, Réseaux dans les réseaux, Réseaux de références.

Abstract

Ambient systems form a very particular field of application because they are very dynamic and in constant interaction with users. Multi-agent systems are used as useful abstractions in these systems, for example for devices and features, and as a paradigm for implementation. However, the characteristics of such systems make their testing a real challenge that requires new and effective test methods to assess the autonomous behaviors of ambient agents and their interactions.

This thesis addresses the problem of model-based-testing for multi-agent systems in ambient intelligence environments through an approach which can be considered as an equipped chain integrating requirements management, test cases generation and their concretization.

In particular, we propose as a first step, an agent-based system for modeling an ambient environment for older people using reference net. The example has been modeled according to the MULAN (multi agent system net) architecture augmented by Java Expert System Shell (JESS) to take into account the deliberative aspects of the ambient agents. In addition, the proposed model takes into account the time aspect for any decision. As a second step, we propose a model-based testing approach based on the nets within nets paradigm that integrates JUNIT tool for the generation and testing multi-agent systems. Unlike the different test approaches, where most of the instrumentation is applied to the source code of the application, our approach leaves the application under test intact by doing instrumentation at the model level, thus preserving its initial behavior (the functional or non-functional aspect will not be modified).

Keywords: Agent, Multi agent Systems, Ambient Intelligence, Model based testing, Nets within nets, Reference nets.

ملخص

تشكل الأنظمة المحيطة مجالًا تطبيقيا خاصًا لأنها ديناميكية للغاية وفي تفاعل مستمر مع المستخدمين. تُستخدم الأنظمة متعددة الوكلاء كنماذج تجريدية مفيدة في هذه الأنظمة، سواء بالنسبة للأجهزة، الوظائف، أو حتى للتنفيذ. لكن خصائص هذه الأنظمة تجعل إختبارها تحديًا حقيقيًا يتطلب طرق إختبار جديدة وفعالة لتقييم السلوكيات المستقلة للعوامل المحيطة.

هذه الأطروحة تعالج مشكلة التجارب على أساس النماذج الرسمية للأنظمة متعددة الوكلاء في بيئات الذكاء المحيطي من خلال إقتراح أسلوب تنفيذ يعتبر سلسلة متكاملة المتطلبات لتوليد الاختبارات،إدارتها وتثبيتها.

على وجه الخصوص ، نقترح كخطوة أولى، نظاما قائما على الوكيل لنمذجة بيئة محيطة للمسنين باستخدام الشبكات المرجعية وقد تم تصميم النموذج وفقًا لهيكل (MULAN) الذي تم تعزيزه بواسطة (JESS) لمراعاة الجوانب التداولية للوكلاء المحيطين. بالإضافة إلى ذلك يراعي النموذج المقترح الجانب الزمني لأي قرار.في خطوة ثانية، نقترح نهجا إختباريا قائما على أساس نموذج الشبكات داخل الشبكات يتكامل مع أداة (JUnit) من أجل توليد وإختبار الأنظمة متعددة الوكلاء. على عكس طرق مناهج الاختبار المختلفة أين يتم تطبيق أجهزة القياس على النظام تحت الاختبار، فإن نهجنا يجعل النظام تحت الاختبار سليم من خلال تطبيق أجهزة القياس على مستوى النموذج وبالتالي الحفاظ على سلوكه الأولي.

الكلماتالمفتاحية: وكيل، أنظمة متعددة الوكلاء، الذكاء المحيطي، إختبار على أساس النموذج، الشبكات داخل الشبكات والشبكات المرجعية.

TABLE DES MATIERES

TABLE DES FIGURES

Liste Des Tableaux

Introduction générale

Dans ce qui suit, nous présentons une vision globale sur les travaux développés au cours de cette thèse, en dévoilant le contexte sur lequel porte notre recherche ainsi que la problématique à résoudre et enfin les contributions apportées par nos travaux.

1. Contexte et problématique

L'informatique a connu une évolution assez profonde ces dernières années, ouvrant la voie à l'ère de l'intelligence ambiante. Celle-ci implique un environnement truffé d'une multitude de composants enfouis et intégrés dans des objets divers du monde réel. L'utilisateur a alors accès, inconsciemment, à tout un ensemble de services interconnectés. En intégrant du même coup des systèmes distribués, ouverts, à grande échelle, hétérogènes et intelligents.

En outre, la mise en place du concept de l'intelligence ambiante soulève des défis scientifiques considérables. Certaines réponses à ces défis sont aujourd'hui offertes par des approches de plus en plus flexibles.

Dans ce cadre, il semble assez logique que les systèmes multi-agents (SMA) constituent un bon candidat comme paradigme pour modéliser et implanter ce type de systèmes, notamment aux travers des concepts de communication, d'organisation, de coopération et de coordination entre des entités plus ou moins intelligentes et autonomes. L'ensemble des concepts de l'approche multi-agents font de celle-ci une approche très riche, très fiable et très puissante.

Cependant, L'omniprésence de l'informatique dans notre vie à l'échelle de l'embarqué et de l'intelligence ambiante a rendu les informaticiens de plus en plus exigeants quant à la qualité des applications. Seule une procédure de test efficace permet de répondre à ces exigences. Cette phase incontournable doit être effectué afin d'assurer la fiabilité de tout programme. Dans ce contexte de systèmes ambiants, le test est un vrai défi, du fait de leurs caractéristiques.

Les tests existent depuis longtemps. Après avoir été délaissés par manque d'intérêt de la part des développeurs, il semblerait que ce soit aujourd'hui les directions informatiques qui

prennent conscience de leur véritable utilité. Ils permettent de rassurer et de pallier les erreurs humaines.

C'est ce contexte qui justifie notre tendance actuelle en matière de test des systèmes multi-agents dans les environnements d'intelligence ambiante par le développement d'une approche de test basée sur des modèles formels. L'utilisation des approches formelles, consiste en une parfaite automatisation de la phase de l'oracle pour pouvoir statuer sur la conformité des résultats obtenus par-rapport aux résultats attendus.

2. Contributions

Le travail présenté ici repose sur les tests basés sur les modèles pour les systèmes multi agents dans les environnements d'intelligence ambiante.

Nos investigations ont consisté, tout d'abord, dans la modélisation qui est la première étape du processus de test basé sur les modèles. Il faut noter que cette phase est la plus difficile, du fait de son caractère déclencheur de tout le processus de test. Nous proposons une architecture à base d'agents (maison intelligente pour personnes âgées). L'ensemble du système est modélisé selon l'architecture MULAN (Multi agent Nets) qui est une implémentation du paradigme des réseaux dans les réseaux. Nous insistons, en particulier, sur les agents ambiants cohabitant dans une telle maison. Etant données leurs facultés d'intelligence, d'adaptation, de prédiction et d'auto organisation, nous proposons d'augmenter l'architecture MULAN en dotant ces agents d'un moteur d'inférence. Notre choix s'est orienté vers le système expert JESS (Java expert System Shell). Le modèle obtenu est validé par l'outil Renew.

Ceci fait, nous sommes prêts pour le test, c'est pourquoi notre deuxième contribution consiste en la proposition d'une approche de test qui a pour buts de :

- Minimiser l'effort de test initial par une génération automatique de cas de tests supportée principalement par le modèle utilisé pour la modélisation,
- Automatiser la phase de concrétisation des cas de tests abstraits (tirés à partir de modèles de tests) en des cas de test concrets (très proches du langage de programmation utilisé pour le développement du système sous test).

3. Organisation du manuscrit

Le présent manuscrit est organisé comme suit :

Introduction générale

Le chapitre 1 présente le domaine des systèmes ambiants, ses caractéristiques, ses principes ainsi que la problématique qu'ils soulèvent. Nous nous y intéressons aux liens du paradigme multi-agents avec la notion de l'intelligence ambiante. Enfin nous citerons quelques domaines qui peuvent profiter des avantages offerts par ce nouveau type de système.

Le chapitre 2 est consacré aux tests de logiciels. Nous présentons les définitions et l'historique ainsi qu'un panorama des différentes techniques et stratégies existantes dans la littérature. Nous terminons cette partie en étalant les différentes étapes du test de logiciel.

Le chapitre 3 expose un panorama des travaux existant dans la littérature concernant le test des systèmes ambiants.

Le chapitre 4 concerne l'ingénierie dirigée par les modèles et plus particulièrement la présentation de la technique du test basé sur les modèles, technique que nous avons retenu pour le test de notre système.

Dans le chapitre 5, nous abordons le formalisme des réseaux de Petri. Nous commençons par définir les réseaux de Petri classiques, nous présentons ensuite leurs différentes extensions. Nous explorons à la fin de ce chapitre les intérêts qu'apportent les réseaux de Petri en général et les réseaux dans les réseaux en particulier pour la modélisation des systèmes ambiants.

Le chapitre 6 détaille une approche de modélisation et de validation d'un environnement ambiant utilisant les réseaux dans les réseaux augmenté du système expert JESS (Java Expert System Shell).

Dans le chapitre 7, nous nous attardons sur le test des systèmes ambiants en concentrant sur le test des agents ambiants et tout en expliquant le déroulement de cette approche.

Finalement, nous dressons une conclusion qui résume nos contributions et présente des perspectives de notre travail.

Chapitre 1

Les systèmes ambiants

1 Les systèmes ambiants

1.1 Introduction

L'informatique connait aujourd'hui des évolutions fulgurantes ouvrant la voie à l'ère de l'intelligence ambiante. Celle-ci fait référence à un nouveau paradigme excitant dans lequel nous serons plongés dans un environnement conscient de notre présence, du contexte dans lequel on est situé et qui est sensible, adaptatif et réceptif à nos besoins, nos habitudes, nos gestes voire à nos émotions afin d'améliorer la qualité de vie.

Dans ce chapitre nous allons commencer par introduire un bref historique concernant l'intelligence ambiante. Nous aborderons, notamment, quelques définitions, ses principes ainsi que ses domaines d'applications.

1.2 Evolution historique de l'intelligence ambiante

En 1988, Mark Weiser responsable scientifique de Xerox PARC aux Etats-Unis publie un article dans lequel il décrit ses travaux en cours et propose les bases de l'informatique du 21ème siècle : « l'informatique ubiquitaire » [1, 2]. Weiser constate que les technologies ancrées dans nos activités quotidiennes sont celles qui savent s'y fondre, jusqu'à disparaître. Il illustre son propos avec l'exemple de l'écriture : omniprésente dans nos sociétés modernes, chacun l'utilise au quotidien sans même y prêter attention.
La vision de l'informatique ubiquitaire a été permise par l'utilisation de très nombreux dispositifs miniaturisés, économiques, robustes et connectés par réseau que l'on a pu dès lors installer dans des endroits où les personnes réalisent leurs activités quotidiennes.

En 1996, Mark Weiser et John Sely Brown[3] publient un article intitulé *«the coming age of calm technology »*. Dans cet article, les auteurs expliquent que la technologie doit être « calme » : elle fournit des informations sans demander d'efforts ou d'attention pour les décoder. Autrement dit : une technologie visant à réduire le nombre d'informations reçues par l'utilisateur selon leurs pertinences.
Un exemple d'informatique calme nous est fourni par Natalie Jeremijenko[4]. Elle a créé et installé à Xerox le « Live wire ». Il s'agit d'un fil relié à un moteur pas à pas dont les soubresauts sont directement reliés au trafic réseau. Cela permet donc de visualiser en

périphérie et sans effort particulier une information autrement inconnue : l'intensité du trafic réseau.

En 1998, lors d'une présentation auprès de Philips, Eli Zelkha[5] a proposé le terme intelligence ambiante, ou Ambient Intelligence (AmI) en anglais, dans le but du développement de scenarios pour l'utilisation de dispositifs fournissant des informations et communications ubiquitaires.

À partir de 2005, l'intelligence ambiante à connu un essor suite à l'intégration des processeurs dans les objets de la vie quotidienne (systèmes embarqués) afin de les rendre de plus en plus mobiles et intelligents (smart phone, puce RFID, capteurs de température, etc.).

1.3 L'intelligence ambiante « AmI »

Aujourd'hui on fabrique des objets de plus en plus petits qu'on peut intégrer dans des appareils électroniques. Ceci accroît leurs utilisations et entraîne des nouvelles recherches. De plus, la recherche et le progrès en communication enrichissent le développement et l'utilisation de l'intelligence ambiante.

Le succès de l'AmI dépend en grande partie du développement de la technologie des capteurs, d'actuateurs, des dispositifs mobiles, etc. , mais également des logiciels et de leur intelligence pour la prise de décision.

Definition1 :

Aarts et Ruyter[6] intègrent la notion de mobilité et positionnent l'AMI par rapport à l'informatique ubiquitaire et à l'informatique mobile. Ils la différencient par la prise en compte du contexte ainsi que le mécanisme d'adaptation, la personnalisation selon les préférences de l'utilisateur et la prise en compte des facteurs sociaux ainsi que la perception permanente de l'état de l'utilisateur.

Definition2 :

Pour Bermejo Nieto[7], l'intelligence ambiante est une mise en relation avec des systèmes de services, c'est-à-dire, avec les technologies pour automatiser les actions et avec les dispositifs pour personnaliser et adapter leurs comportements.

Definition3 :

Selon Reignier[8], l'intersection de l'informatique ubiquitaire et l'intelligence artificielle a donné naissance à l'intelligence ambiante. Celle-ci trouve donc son origine dans la vision de l'informatique ubiquitaire dans laquelle les ordinateurs imprègnent l'environnement quotidien

tout en étant transparents à l'utilisateur, mais en y ajoutant la notion d'intelligence, c'est-à-dire la faculté de perception et d'analyse de l'environnement, des utilisateurs et de leurs activités afin de réagir et s'adapter dynamiquement en fonction du contexte.

Donc, L'intelligence ambiante propose de créer un environnement sensible, adaptatif et qui réagit intelligemment à la présence de personnes afin de créer l'ambiance souhaitée. Elle implique un environnement harmonieux composé de réseaux informatiques exploitant une technologie de pointe pour relier nos maisons, nos bureaux et nos voitures. Un environnement conscient de la présence humaine, des personnalités, de leurs besoins et qui est capable de répondre intelligemment à des indications parlées, à des gestes ou des désirs, et même à s'engager dans un dialogue intelligent.

Quatre ingrédients sont nécessaires à l'intelligence ambiante (Figure1-1) :

•**L'informatique ubiquitaire (agents ambiants) :** Signifie l'intégration de microprocesseurs dans des objets du quotidien comme des meubles, des vêtements ou des jouets.

•**La communication ubiquitaire :** Visant à permettre la communication des éléments entre eux et avec les utilisateurs, tout en assurant une bonne sécurité (notamment concernant les données personnelles qui peuvent être acquises).

•**Les interfaces utilisateur intelligentes :** Ce domaine a pour vocation de définir les moyens et les outils à mettre en œuvre, afin qu'une personne puisse interagir (contrôler et communiquer) avec une machine d'une manière intuitive et transparente.

•**La sensibilité au contexte :** C'est la faculté du système à « *sentir* » en permanence la présence et la localisation des objets, des appareils et des personnes pour prendre en compte le contexte d'usage (Le Contexte est l'ensemble des états de l'environnement et les paramètres qui déterminent le comportement d'une application dans laquelle un événement se produit et est intéressant pour l'utilisateur[9]).

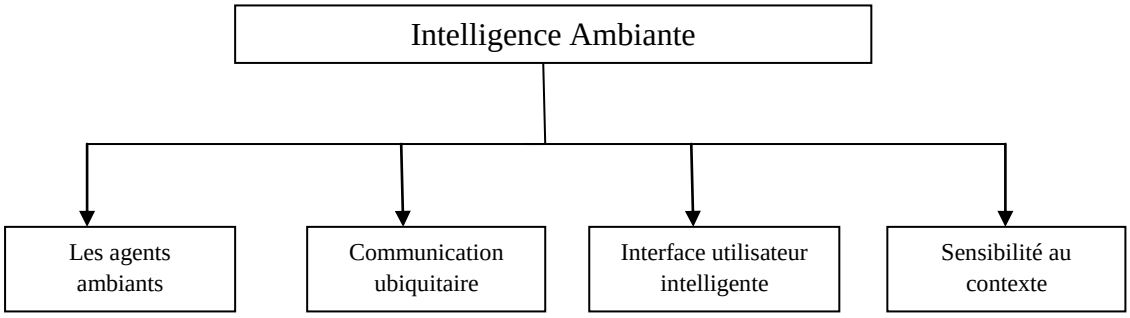


Figure 1-1 : Les ingrédients nécessaires à l'intelligence ambiante.

1.4 Systèmes de l'intelligence Ambiante (Systèmes ambiants)

Il est indispensable d'adopter un système pour le fonctionnement d'un environnement ambiant qui soit capable d'assurer sa gestion d'une façon permanente et transparente. On parle dans ce cas de systèmes ambiants ou simplement systèmes AmI.

Les systèmes de l'AmI peuvent être organisés de différentes façons, mais certaines caractéristiques se retrouvent dans toutes les architectures. Juan Carlos Augusto[10] a proposé une architecture générique pour tous les systèmes de l'AmI.

AmI_Système = <E, IC, I> De telle sorte que :

E : est l'environnement. Par exemple : la maison, le campus, l'hôpital, etc.
I : est un ensemble d'Inter acteurs (il peut s'agir de personnes, d'animaux ou de robots). Ils peuvent interagir avec le système de diverses façons.
IC : est un ensemble de contraintes d'interaction. Il précise les façons possibles pour les éléments de E et de I d'interagir les uns avec les autres.

En outre, pour assurer un contrôle total des espaces intelligents, un système AmI doit être capable de[11] :

- Percevoir les événements générés par des entités. Une entité peut être une personne, un équipement, un lieu ou une application informatique. Les entités sont en général caractérisées par des propriétés : un lieu peut être caractérisé par exemple non seulement par sa topologie mais également par ses niveaux de bruit, de luminosité, de température et d'humidité,
- Collecter et interpréter les informations à partir de sources hétérogènes,
- Interpréter l'intention de l'humain,
- Réagir en conséquence de manière appropriée et transparente.

1.5 Caractéristiques et défis des systèmes ambiants

Les systèmes ambiants présentent les caractéristiques suivantes[11] :

- Hétérogénéité

L'hétérogénéité du système ambiant est due à la diversité des technologies des capteurs et des actionneurs ainsi qu'aux technologies de communication filaires et sans fil.

-Ouverture

L'ouverture de l'environnement à des entités électroniques distinctes et variées dans un contexte évoluant rapidement est aussi une caractéristique importante des environnements ambiants.

-Distribution

L'intelligence ambiante regroupe des systèmes matériels et logiciels embarqués dans lesquels le traitement de l'information est réparti sur les objets du réseau.

Pour cela, La distribution est essentielle pour un système de l'AmI parce que la quantité d'entités et la quantité d'informations échangées dans le système sont très grandes. De plus, elle rend le système plus flexible et robuste.

-Multi-échelle

Les environnements ambiants se caractérisent par un nombre important de ressources (de calcul, de communication et d'interaction, de capteurs et d'actionneurs) à gérer.

- Dynamicité et incertitude

La nature dynamique et incertaine des systèmes ambiants s'incarne dans l'apparition et la disparition des différentes entités ainsi que dans la modification des besoins des utilisateurs.

Ces caractéristiques imposent ainsi des défis scientifiques considérables pour le développement, le déploiement et la maintenance de ces applications et de leurs supports d'exécution. Ces défis se résument dans ce qui suit :

- Interopérabilité

Un système AmI doit assurer l'interopérabilité afin de traiter principalement les problèmes liés à l'hétérogénéité des environnements à intelligence ambiante. Cette dernière pose le problème de partage des données et indirectement des connaissances contextuelles.

- Auto-adaptation, Sensibilité au contexte et Intelligence

Ces trois défis procurent au système AmI la capacité de raisonnement pour la reconnaissance de contextes et la prise de décision, l'adaptation et l'extensibilité afin d'assurer la pérennité.

Face à la dynamique et à l'incertitude de l'environnement ambiant, les mécanismes d'auto-adaptation sont nécessaires afin d'assurer une continuité de service et répondre aux attentes des utilisateurs.

En effet, un système AmI doit percevoir le contexte de l'environnement pour interagir plus «naturellement » avec l'utilisateur. La perception du contexte est réalisée par des capteurs de l'environnement physique.

Enfin, L'intelligence du système réside dans sa capacité d'analyse et d'agrégation du contexte.

Pour accroître cette intelligence et la capacité du système AmI, il y'a lieu de rester attentif pour répondre à tout moment et en tous lieux aux besoins des utilisateurs. Afin de pouvoir, ainsi, évaluer une situation courante ou possible, comparer deux situations et juger de la meilleure. Des mécanismes de raisonnement et de pro-action « *all the time everywhere* » sont utilisés à cet effet.

-Passage à l'échelle

Pour gérer la distribution et l'ouverture de l'environnement AmI, le système intelligent doit être doté de mécanismes lui permettant d'assurer un passage à l'échelle (Scalabilité). Pour ce faire, le cœur de ce système doit être développé indépendamment du volume des utilisateurs et des appareils.

-Invisibilité et transparence

Les objectifs de l'intelligence ambiante sont :

-de rendre l'informatique invisible en la diluant dans l'environnement quotidien.

-de rendre l'accès à l'information plus transparent vis à vis du lieu et du contexte actuel.

-Sécurité et respect de la vie privée

Les environnements d'intelligence ambiante peuvent comporter de multiples sources d'informations qu'il faut contrôler et en protéger l'accès pour promouvoir la sécurité et enrichir la vie des utilisateurs tout en respectant les aspects éthiques et juridiques.

1.6 L'AMI et les systèmes multi-agents (SMA)

L'intelligence ambiante s'impose de plus en plus comme une réelle alternative aux systèmes d'aujourd'hui. Elle propose de nouveaux modes d'interactions faciles, très efficaces et contrôlables entre les personnes et la technologie afin d'améliorer la qualité de vie.

Cependant, ces environnements sont associés à des aspects complexes comme la contextualisation des données, la mobilité de l'information sur les réseaux, les interactions avec les utilisateurs, les dispositifs qui sont très variés et de nature distribués et hétérogènes, la dynamicité, etc. Ainsi, la mise en place du concept de l'intelligence ambiante soulève des défis scientifiques considérables. Certaines réponses à ces défis sont aujourd'hui offertes par des approches de plus en plus flexibles.

Dans ce cadre, les systèmes multi-agents (SMA) se présentent comme des solutions adéquates pour implémenter de telles applications car ils sont destinés à opérer dans des

contextes dynamiques, distribués, imprévisibles pour lesquels les systèmes traditionnels ne sont souvent pas très performants.

De plus, les agents peuvent être utilisés comme des abstractions utiles dans les systèmes AmI. Par exemple, pour les appareils et les fonctionnalités, et en tant que paradigmes pour la mise en œuvre. Ils peuvent être utilisés à différents niveaux : Pour modéliser les dispositifs présents dans l'environnement AmI, ils peuvent être exploités au niveau intergiciel pour coordonner les activités des entités du niveau inférieur ou ils peuvent être utilisés au niveau supérieur afin de former l'interface pour les humains[12].

Pour conclure, on peut dire qu'avec l'avènement des systèmes de l'intelligence ambiante **«** systèmes constitués de nombreux composants en interaction dynamique entre eux et avec le monde extérieur », les systèmes multi-agents sont devenus un paradigme de conception à part entière. Ils permettent de travailler sur le fonctionnement global d'un système en s'intéressant aux entités qui le composent et à leurs interactions.

Les agents offrent des caractéristiques indéniables pour la conception et le développement de systèmes d'AmI[13] :

- **Réactivité** : Répondre aux évènements ou aux besoins explicites de l'utilisateur.
- **Pro-activité** : Agir sans demande explicite pour anticiper les besoins de l'utilisateur.
- **Raisonnement** : Agir de manière intelligente pour prendre la décision correcte convenant à la situation.
- **Autonomie** : Un agent peut collaborer avec d'autres agents mais peut agir seul dans n'importe quelle situation.
- **Mobilité :** La mobilité qualifie la faculté de déplacement (migration) de l'agent d'un hôte à un autre pour effectuer ses différentes tâches et où il peut poursuivre ou reprendre son exécution.

Réaliser un environnement d'intelligence ambiante à l'aide d'un système multi-agents nous amène à l'utilisation d'un typage d'agent particulier nommé agent ambiant.

1.6.1 Définition d'un agent ambiant

Un agent ambiant peut être conçu pour être embarqué sur un appareil électronique formant ainsi un Ambient Intelligence Device (AmID)[14]. Cet AmID est destiné à intégrer un environnement contenant des dizaines, voire des centaines d'autre AmIDs.

Afin que cet ensemble interagisse et que ses constituants ne restent pas inertes les uns par rapport aux autres, il est nécessaire qu'un ensemble de capacités et de comportements soient fournis aux agents.

A la différence de la plupart des agents coopératifs, les agents ambiants doivent posséder des caractéristiques et des capacités indispensables au domaine de l'intelligence ambiante. Très souvent, les agents classiques sont intégrés dès le départ dans un même système. Ils communiquent avec un même langage et les mêmes protocoles de communication. Généralement ils ne sont pas destinés à fonctionner sur des appareils mobiles.

Contrairement aux agents classiques, les agents ambiants coopératifs n'ont généralement aucune connaissance initiale sur leurs voisins, ni de leur existence, ni de leurs caractéristiques. Ils doivent donc avoir la capacité de s'intégrer dans un système, lui-même en changement. Ces agents peuvent avoir différents langages et protocoles de communication. Ils sont conçus pour fonctionner sur des appareils mobiles avec les contraintes qui en découlent.

1.6.2 Capacité d'un agent ambiant

Pour répondre à son environnement, un agent ambiant nécessite plusieurs capacités :

- Il offre et réalise des services : En utilisant les ressources de l'appareil dans lequel il est embarqué (Stockage, traitement, etc.), ou grâce à ses propres capacités (représentation du contexte).
- Il réagit aux capacités de l'appareil dans lequel il est embarqué : Il doit prendre en compte les caractéristiques spécifiques de l'appareil qui l'intègre (autonomie, capacité de calcul).
- Il peut percevoir son environnement : Cette capacité permet à l'agent d'agir, il peut percevoir son environnement localement (grâce à des capteurs), ou grâce aux informations reçus d'autres agents.
- Il a la capacité de collaborer et d'interagir avec les autres agents : Un agent ambiant n'est pas un simple système embarqué dédié à une tâche bien précise. Il peut communiquer, coopérer avec d'autres appareils, etc.
- Il doit pouvoir respecter les règles qu'on lui impose : Le facteur humain étant l'élément primordial dans un système d'AmI. La satisfaction des besoins des utilisateurs, le respect de leurs intérêts et de leurs vies privées sont des aspects primordiaux pour un agent ambiant.

1.7 Quelques exemples d'applications

L'intelligence ambiante assiste l'humain dans son quotidien afin de le rendre plus confortable, plus sécurisé et plus agréable à vivre. De plus, elle permet d'économiser du temps et améliore le fonctionnement et les fonctionnalités des systèmes. C'est une technologie de pointe qui s'améliore tous les jours.

De nombreux domaines peuvent profiter des avantages offerts par ce nouveau type de système. Nous détaillerons dans un premier temps quelques exemples de services proposés dans le domaine de la domotique pour poursuivre sur d'autres exemples dans d'autres domaines.

La domotique peut être vue comme la discipline qui étudie et réalise l'automatisation des tâches au sein d'une maison (tâches domestiques). Elle est l'un des domaines privilégiés de l'intelligence ambiante, car la maison est le lieu que l'on aimerait qu'il soit le plus confortable que possible.

En effet, ce domaine d'application à retenu le plus grand intérêt de la part des chercheurs où la construction des maisons intelligentes est le champ le plus exploré.

Qu'est-ce qu'une maison intelligente ?

La maison intelligente (Smart Home) est un espace sensible où les sujets humains et les objets domestiques inter-communiquent via des agents intelligents et des connexions internet. Voici de quoi auront l'air nos maisons dans un futur proche (figure 1-2).

Figure 1-2 : Maisons futuristes.

Chapitre 1 Les systèmes ambiants

De nombreuses recherches effectuées donnent des exemples de maisons intelligentes, ces recherches ont démontré qu'ils pouvaient engendrer des économies énergétiques, assurer le confort et la sécurité des personnes et des biens matériels.

Pour nous faciliter la vie, l'éclairage suit nos déplacements sans avoir besoin d'actionner les interrupteurs. La maison diffuse des odeurs et teinte ses murs selon nos humeurs. Le réfrigérateur « intelligent » connaît son contenu et prévient l'utilisateur des articles à renouveler. On peut mentionner l'hôpital Danderyd en suède, qui s'équipe d'une maison domotique pour faciliter la vie de ses patients. La cuisine de cette maison est entièrement équipée d'appareils domotiques avec un contrôle des commandes sur écran tactile. Sa salle de bain est aussi adaptée aux besoins des patients.

On ce qui concerne les économies d'énergie le but est d'éviter le gaspillage en supprimant les dépenses inutiles. On peut citer à titre d'exemple le projet (e-House 2000)[1] qui est un bâtiment expérimental avancé permettant le contrôle de nombreux dispositifs à partir d'un site web spécifique. e-House 2000 permet de contrôler la température, le niveau de lumière, le chauffage, l'humidité. En ajustant au mieux le chauffage et les sources de lumière ; et en déclenchant les appareils les plus consommateurs d'énergie au moment où la tarification est basse.

En matière de sécurité domestique, les maisons intelligentes sont dotées d'alarmes, de détecteurs de mouvement ou d'intrusion, d'interphones et portiers vidéo, simulateurs de présence pour dissuader les visiteurs indésirables ou malintentionnés. D'autres systèmes de détection sont prévus pour surveiller les enfants, prévenir les risques d'accident (incendie, fuite de gaz…), signaler des pannes (inondation, coupure de courant…). De plus, les maisons intelligente ont le potentiel de permettre aux personnes âgées et handicapées à mener une vie autonome dans leurs propres maisons pour améliorer l'autonomie, la confiance en soi et la qualité de la vie. On peut citer le projet HIS (Habitat Intelligent pour la Santé)[15]. Cet habitat est destiné aux personnes souffrant de diverses pathologies (chroniques) ou insuffisances (handicaps, dépendance). Ses objectifs sont :

- Favoriser la vie et l'autonomie au domicile, diminuer les risques, détecter les chutes et situations critiques, répondre à des besoins divers d'assistance d'information, de relation…

[1] http://michaelmcdonough.com/e-House/ehouse.htm

- Favoriser les réseaux de santé. Tout en exploitant le meilleur des technologies actuelles d'une manière non invasive, non intrusive et dans le respect de la vie privée et du secret médical.

L'habitat est truffé de capteurs et actionneurs afin de Détecter des situations critiques (chutes, malaises, appels vocaux,...). Il permet aussi l'interprétation automatique des données des capteurs (paramétrée, sécurisée, optionnelle - si nécessaire, souhaitée).ces données sont transmises via diverses voies sécurisées : réseau commuté, Internet, GSM…vers un centre d'appel (téléassistance /télésurveillance) avec des services médico-sociaux.

On adopte aussi l'AmI dans les environnements d'apprentissage et éducatif afin d'encourager les étudiants à collaborer de différentes façons, de faire évoluer et de mieux orienter leur compétences, ainsi que la façon dont les enseignants peuvent introduire du matériel dans une variété de travaux multidisciplinaires. De même, on l'utilise pour la détection des difficultés et des problèmes et pour offrir une aide personnalisée. On peut l'exploiter aussi pour soutenir les apprenants et les enseignants dans les activités de classe et d'étendre l'apprentissage au-delà de la salle de classe. Automatiser des tâches en rendant l'information disponible n'importe où et n'importe quand.

On peut citer le projet iClassroom qui permet la création d'un environnement intelligent comme réalisé à l'Université d'Essex. Son but est de soutenir une gamme de scénarios d'enseignement et d'apprentissage. Cela inclut le e-Learning, les conférences et les tutoriels, mais aussi des activités plus collaboratives de résolution de problèmes d'apprentissage.

Pour cela, l'intelligence ambiante représente un outil intéressant quant à l'augmentation de la productivité et des résultats de ceux-ci.

Dans le secteur du transport, on cherche à rendre le trafic plus fluide, le transport plus efficace, plus sûr grâce aux technologies GPS de recherche du meilleur chemin, en évitant les problèmes routiers (travaux, accidents, …), en obtenant des informations sur l'itinéraire (route, temps, etc.), en utilisant le paiement électronique sur l'autoroute[16] et des techniques de stationnement automatisé.

Une réponse pertinente à la gestion du trafic saturé des métropoles asiatiques ainsi qu'au respect de l'environnement est : les bus suspendus de Shenzhen Hashi Future Parking[2].

Un modèle innovant de recharge de bus électrique, permettant un transport plus écologique est le projet TOSA[3].

[2] http://en.wikipedia.org/wiki/3D_Express_Coach

On utilise l'AmI dans la gestion des crises pour mieux les coordonner et les administrer. Les services de pompiers ont des besoins d'outils informatiques pour la découverte et la gestion des incidents par tout intervenant, même non expert, présent sur le terrain et améliorer le temps de réaction, trouver les lieux plus vite, etc. le projet AMIRA[17] (Advanced Multimodal Intelligence for Remote Assistance) porte sur l'assistance aux pompiers en cas d'incendie et dans le projet TNO[18] L'opérateur de tunnels surveille et régule le flux du trafic dans et proche d'un tunnel. Le système décide des actions à entreprendre en cas d'incident.

L'AmI est aussi employée dans le secteur du tourisme pour la personnalisation et l'adaptation des voyages pour différents touristes.

Dans les usines[16], on l'applique pour le contrôle des chaînes de production dans le but d'augmenter la production en fonction de la demande et pour l'organisation et la sécurité du personnel. De même, on peut l'utiliser pour la prise de décision qui est une charge importante au travail. En effet, elle est basée sur différents points de vue, d'échange d'idées, d'arguments et elle demande de négocier, coopérer, collaborer et discuter sur le sujet.

Enfin, on emploie l'intelligence ambiante dans le domaine de la vidéosurveillance pour augmenter la sécurité, la sûreté et les statistiques. On dissémine des systèmes de surveillance vidéo dans les lieux publics (en particulier les métros, les gares et les aéroports) afin de prévenir les crimes, le vandalisme ou même des attaques terroristes. De même, la vidéosurveillance est de plus en plus populaire pour un usage privé dans les maisons, bureaux et banques, visant à garantir la sécurité personnelle des citoyens et des travailleurs.

1.8 Approche par système multi-agents dans les maisons intelligentes

Plusieurs travaux concernant les maisons intelligentes qui utilisent des systèmes multi-agents vont être présentés. Il est à noter que la description sur les agents dans ces travaux n'est pas très détaillée.

Dans[19], les auteurs présentent un système multi-agent capable de détecter les chutes à travers des capteurs dans un périphérique mobile et agir en conséquence lors de l'exécution.

Aussi, Lim et al.[20] ont créé une maison contextuelle qui utilise un système multi-agent pour surveiller et exécuter les actions appropriées en fonction de l'état actuel de la maison.

[3] http://www.tosa2013.com/fr

Dans[21], les auteurs proposent un AmIHomCare basé sur une architecture multi-agents pour un système d'acquisition de services. Ce système d'acquisition de services est spécifié et validé en utilisant une méthode de spécification formelle baptisée Evénement B.

Egalement, Sun et al.[22] ont proposé un cadre de conception de systèmes multi-agents (SMA) pour réaliser une automatisation intelligente de la maison. Les nouveautés de ce travail incluent le développement de : 1) modèles de comportement d'agent de croyance, de désir et d'intention (BDI), 2) un mécanisme de collaboration multi-agents basé sur une politique de régulation et 3) un ensemble de métriques pour l'évaluation de la performance SMA.

Enfin, une architecture hybride multi-agents qui facilite les services de surveillance et de soins à distance pour les patients handicapés à leur domicile à été proposé dans[23]. Elle combine des systèmes multi-agents et des services Web pour faciliter la communication et l'intégration avec plusieurs systèmes de soins de santé. En outre, cette architecture se concentre sur la conception d'agents réactifs capables d'interagir avec différents capteurs présents dans l'environnement et intègre un système d'alertes via les technologies mobiles SMS et MMS. Elle utilise aussi les technologies RFID et JavaCard pour fournir des systèmes de localisation et d'identification avancés ainsi que des fonctions de contrôle d'accès automatiques.

1.9 Nouvelles tendances

Récemment tous les travaux de recherche ont montré que les dispositifs portables (wearable devices) basés sur la détection inertielle sont devenus une solution populaire pour la reconnaissance des gestes et les applications de localisation à la maison.

Hsu et al.[24] ont développé un système de maison intelligente basé sur la technologie de fusion de données multi-capteurs en intégrant la technologie intelligente portative (wearable intelligent technologie), l'intelligence artificielle et la technologie de fusion de capteurs pour la reconnaissance de geste et les services de localisation.

Egalement, Hong et al.[25] ont proposé un système de reconnaissance gestuelle de mouvement basé sur des accélérations pour classer les ensembles de confusion et les gestes faciles à définir, qui extraient les caractéristiques basées sur le domaine temporel, le domaine fréquentiel et la décomposition des valeurs singulières (SVD); gestes classifiés en utilisant le classificateur machine à vecteur de support (SVM). Selon ces auteurs, la précision pour

catégoriser l'ensemble de confusion et les gestes d'ensemble faciles sont de 89,92% et de 95,40%, respectivement.

1.10 Conclusion

Grâce à sa popularité grandissante dans les divers laboratoires de recherche, l'intelligence ambiante constitue une alternative aux technologies utilisées aujourd'hui.

Elle affirme la volonté de rendre l'informatique invisible en la diluant dans l'environnement quotidien et couvre un large spectre de domaines incluant des applications extrêmement diverses.

Aussi, l'application des principes utilisés dans ce domaine requiert l'utilisation de méthodologies spécifiques pour qu'on puisse répondre à certains défis et appliquer cette vision à tous les domaines de notre quotidien.

Par ailleurs, dans l'optique des systèmes ambiants, tous les objets de notre quotidien fonctionnent avec des programmes informatiques communicants plus ou moins complexes, seront amenés à interagir avec l'utilisateur et avec le monde extérieur. Cependant, ces interactions peuvent parfois être critiques. Pour cela, les systèmes ambiants restent toutefois soumis à de fortes contraintes et exigences, parmi lesquelles la nécessite d'assurer une certaine qualité dans leur développement et surtout une fiabilité dans leur utilisation.

Cette fiabilité est assurée d'une part par un processus de spécification et de développement très rigoureux, d'autre part par une phase de test logiciel dont le but est de détecter les erreurs de conception et de réalisation des programmes pour tendre vers des systèmes avec zéro défaut.

Chapitre 2

Le test des logiciels

2 Le test des logiciels

2.1 Introduction

Les logiciels sont devenus indispensables et omniprésents dans notre quotidien et ils sont très souvent utilisés pour accomplir des tâches critiques.

Malheureusement, ces systèmes ont maintes fois montré des défaillances conduisant souvent à des dégâts importants et des pertes en vies humaines, des exemples tristement célèbres existent :

- **En 1992 :** les ambulances de Londres sont mal orientés par le logiciel, des pertes humaines sont à déplorer.
- **En 1996 :** l'explosion de la fusée Ariane 5 au bout de 30 secondes de vol suite à une erreur de conversion de données numérique.
- **En 2004 :** une défaillance du système d'alarme d'une centrale qui produisit une coupure d'électricité aux États-Unis et au Canada.
- **En 2007(**armée sud-africaine) : un canon anti aérien Oerlikon, quatre cracheurs de balles de 35 millimètres, s'est retourné, tout en tirant, au hasard. Un bug informatique serait responsable.
- **En 2010 :** Le problème de freins de Toyota en raison d'une défaillance dans le logiciel de freinage antiblocage (ABS).
- **En2015 :** une panne informatique à l'aéroport d'Orly a provoqué l'interruption du trafic durant une demi-heure. La panne a été provoquée par une erreur dans le système de Diffusion des données d'Environnement Contrôle d'Orly et de Roissy (DECOR) qui délivre les informations métrologiques aux pilotes.
- **Enfin en 2017**, Une panne du logiciel Amadeus Altéa, utilisé par 125 compagnies aériennes, empêche l'enregistrement des voyageurs. Cette panne a provoqué un grand désordre dans plusieurs grands aéroports.

Ces pannes majeures et répétées ont sérieusement entravé la confiance du public dans la sûreté du logiciel.

Afin d'éviter ce genre de catastrophe, Il est important d'étudier la sûreté de fonctionnement des logiciels. On peut estimer la confiance que l'on peut avoir en ces

systèmes en effectuant des tests avec des méthodes de conception et de validation rigoureuses en vue d'atteindre la qualité voulue.

Ceci dit, nous sommes maintenant prêts à présenter les concepts de base du test de logiciel.

2.2 Définitions

2.2.1 Qu'est-ce qu'un produit logiciel ?

Un produit logiciel est :

- des documents de gestion de projet ;
- une spécification décrivant :

-la liste des fonctions à remplir par le logiciel

- les facteurs qualité du logiciel : sa portabilité, son évolutivité, sa robustesse, . . .

- les contraintes : performances temporelles et spatiales, . . .

-les interfaces avec son environnement

- une conception décrivant :

- la découpe de la spécification en modules (ou objets)

-la description des interfaces entre ces modules (ou objets)

- la description des algorithmes mis en place.

- un code source,
- un exécutable.

2.2.2 Qu'est-ce qu'un test de logiciel ?

Il existe une pléthore de définitions du terme test de logiciel, nous en présentons les suivantes :

- *Le test consiste en la vérification dynamique du comportement d'un programme sur un nombre fini de cas de test, sélectionnées convenablement à partir du domaine d'exécution (généralement infini) vis-à-vis du comportement attendu[26].*
- *Le test est le processus d'évaluation systématique d'un système en observant son exécution[27].*
- *Le test est une technique de contrôle consistant à s'assurer, au moyen de son exécution que le comportement d'un programme est conforme à des données préétablies[28].*

- *Le test est l'exécution ou l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus*[4].
- Enfin, *Le test d'un logiciel est une activité qui fait partie du processus de développement. Il est mené selon les règles de l'assurance de la qualité et débute une fois que l'activité de programmation est terminée. Il s'intéresse aussi bien au code source qu'au comportement du logiciel. Son objectif consiste à minimiser les chances d'apparition d'une anomalie avec des moyens automatiques ou manuels qui visent à détecter aussi bien les diverses anomalies possibles que les éventuels défauts qui les provoqueraient[29]*.

On pourrait hâtivement conclure que le test est une dimension incontournable pour augmenter la qualité du logiciel et diminuer les probabilités de ses défaillances.

2.3 Terminologies spécifiques aux tests des logiciels

Pour cerner la notion du test, nous avons jugé nécessaire de présenter les terminologies ci-dessous :

- **Spécification :** c'est la description d'un système avec un modèle formel ainsi que l'ensemble des propriétés qu'il doit vérifier. Autrement définit : une spécification exprime ce qu'on attend du système.
- **Vérification :** elle permet de s'assurer que le système est conforme à sa spécification. Elle permet de répondre à la question : « est-ce que le système fonctionne correctement ? ». Autrement définie : la vérification c'est prouver formellement une propriété (exigence) sur un logiciel[30]. En effet, le test est la méthode de vérification la plus répandue.
- **Validation** : elle permet de s'assurer que le système satisfait les besoins de l'utilisateur. Elle consiste à répondre à la question : « est-ce que nous avons bien construit le système ? ».Ou bien : « Est-ce que le logiciel offre les services attendues ? ». La figure 2-1[5] illustre la notion de la vérification et la validation.
- **Qualification :** c'est émettre un avis sur le logiciel.

[4]IEEE standard glossary of software engineering terminology - IEEE Std 610.12-1990 available: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=49C6B9AB6A35DD79A7D7E87A1BD7273A?

[5] http://mrbool.com/software-testing-verification-and-validation/29609

- **Cas de test :** « *c'est un chemin fonctionnel à mettre en œuvre pour atteindre un objectif de test. Un cas de test se définit par le jeu de test mis en œuvre, le scénario de test à exécuter et les résultats attendus* »[6]. Ou tout simplement : un cas de test est une exécution du programme déclenchée sur des données de test (jeux de test).

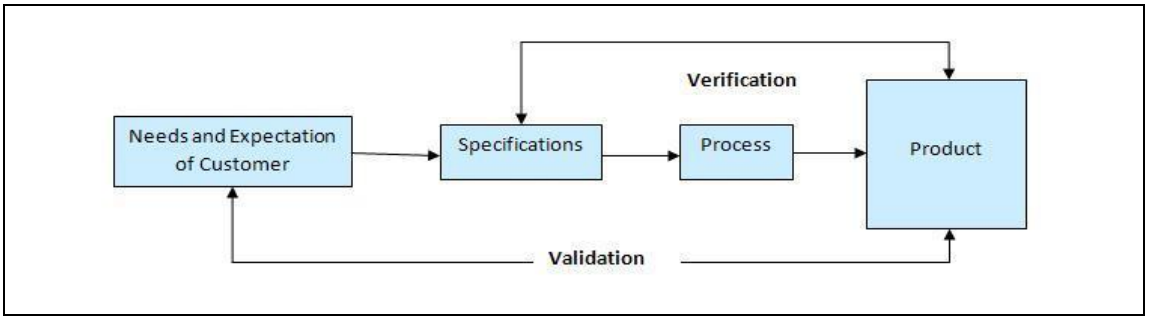


Figure 2-1: Notion de la V&V (vérification et validation).

- **jeux de test :** concerne les données en entrée d'un cas de test : valeurs à saisir, données réelles (base de données existante ou de test), génération automatique (aléatoire ou à partir de spécifications). Le même jeu de test peut servir à plusieurs cas de test[7].
- **Scénario de test :** c'est l'ensemble des actions à effectuer avant de soumettre le jeu de test ; Le scénario de test produit un résultat. Ce dernier doit être évalué de manière manuelle ou automatique pour produire un oracle[8].
- **Oracle :** Un oracle est un artefact qui comprend les connaissances sur le comportement attendu du système sous test (SUT)[31].

Selon la Norme IEEE (Software Engineering Terminology):

- **Erreur ou faute** (fault) **:** commise par le développeur, entraîne un défaut,
- **Défaut :** imperfection dans le logiciel, pouvant amener à une panne,
- **Panne ou défaillance** (failure) **:** comportement anormal d'un logiciel.

Ces trois dernières terminologies sont liées par une relation de causalité (figure2-2).

- **Plan de test :** Un plan de test se représente par un tableau contenant : une description du test, les valeurs en entrées du module et les résultats attendus. On numérotera chaque test (phases).

[6] http://dico.developpez.com/html/1111-Gestion-de-projet-cas-de-test.php
[7] http://dico.developpez.com/html/1118-Gestion-de-projet-jeu-de-test.php
[8] http://dept-info.labri.fr/~felix/Annee2008-09/S4/McInfo4_ASR%20Tests/1.pdf (last accessed on March 2017).

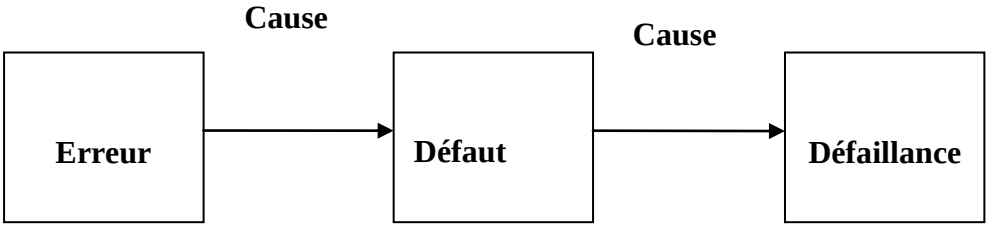


Figure 2-2 : Relation entre défaillance, défaut et erreur.

2.4 Qu'est-ce que les tests et qu'est-ce qui ne l'est pas ?

Le test est : 1) exécuter le programme, 2) connaître les résultats attendus (notion d'oracle), 3) on échoue lorsqu'on ne trouve aucune erreur.

Le test n'est pas : 1) ne donne pas d'explication d'une erreur, 2) ce n'est pas une validation (pas d'erreur détectée ne signifie pas que le programme soit correct).

2.5 Evolution historique du test de logiciel

Examinons brièvement l'évolution du test de logiciel proposé par Gelperin et Hetzel [32-34] (voir tableau 2-1 ci-dessous) :

	Evolution du test
▪ - 1956	Période orientée débogage
1957 - 1978	Période orientée démonstration
1979 - 1982	Période orientée destruction
1983 - 1987	Période orientée évaluation
1988 -	Période orientée prévention

Tableau 2-1 : Historique du test du logiciel.

2.5.1 Test orienté débogage

Ces tests étaient effectués durant les premières années de l'informatique. A cette époque, les activités d'identification et de correction étaient basées sur l'expérience des programmeurs et leur capacité à comprendre les programmes écrits. Toutes ou une partie de ces activités étaient communément appelées test.

C'est ainsi que les programmeurs écrivaient et vérifiaient les programmes jusqu'à ce qu'ils soient satisfaits que toutes les erreurs aient été identifiées et corrigées.

2.5.2 Test orienté démonstration

Les activités de test et de débogage étaient identifiées comme étant des activités séparées, car à cette époque ce qui était important c'était de prouver que le logiciel soit à la hauteur de toute attente.

Les activités de test et de débogage consistaient en :

- **Activité de débogage :** qui consiste à s'assurer que le programme s'exécute (ne crache pas).
- **Activité de test :** consiste à s'assurer que le programme fait ce qu'il est censé faire.

2.5.3 Test orienté destruction

Avec la publication du livre *"The Art of Software Testing"* par Myers[35], la définition des termes test et débogage a une fois encore changé devenant ainsi :

- ***Débogage* :** concerne la localisation et la correction des erreurs.
- ***Test* :** concerne la détection de la présence des erreurs dans un programme.

Les critères de sélection des cas de test étaient choisis de façon à révéler des erreurs particulières dans le logiciel.

2.5.4 Test orienté évaluation

Après la publication d'un rapport par *The National Bureau of Standards[36]* en 1983, qui a mis en évidence le principe attestant que plutôt une erreur est détectée moins sera le coût de sa correction, l'ère de prouver la qualité du logiciel et la mesurer est née. C'est ainsi que le test a été considéré comme une évaluation nécessaire à la fin de n'importe quel cycle de vie du logiciel.

2.5.5 Test orienté prévention

En fait, ce qui est important et primordial c'est de mettre sur le marché un produit logiciel sans défauts, répondant aux spécifications « Quels tests devons-nous utiliser ? ». Hetzel et Gelperin[33] ont commencé à généraliser les méthodes du test unitaire et à développer une méthodologie compréhensive pour la gestion pratique des tests. Ces derniers, selon leur point de vue, doivent prévenir des erreurs à chaque phase du cycle de vie. Les erreurs peuvent être introduites dans des parties du code et les critères sont maintenant orientés vers leur détection.

2.6 Quelques principes de base

Nous présentons ci-dessous les principes de test qui nous semblent être les plus importants[35]:

Principe 1 : Un programmeur ne doit pas tester ses propres programmes. Tout simplement pour ne pas être juge et partie.

Principe 2 : Ne pas effectuer des tests avec l'hypothèse de base qu'aucune erreur ne va être trouvée.

Principe 3 : La définition des sortie ou résultats attendus doit être effectuée avant l'exécution d'un test.

Ceci est une erreur fréquente du test. Ce n'est pas lorsque l'on effectue le test d'une procédure qu'il faut se poser la question de la validité des résultats, car il y a des chances non négligeables de prendre un résultat erroné mais semblant être cohérent pour un résultat correct.

Principe 4 : Inspecter minutieusement les résultats (trace) de chaque test. Pour une raison similaire au principe 2, il faut séparer l'exécution (action) et l'analyse des résultats.

Principe 5 : Les jeux de tests doivent être écrits pour des entrées invalides ou incohérentes aussi bien que pour des entrées valides. Tout simplement, afin de découvrir les anomalies.

Principe 6 : Vérifier un logiciel pour détecter qu'il ne réalise pas ce qu'il est supposé faire n'est que la moitié du travail. Il faut aussi vérifier ce que fait le programme lorsqu'il n'est pas supposé le faire. Afin d'évaluer la robustesse du logiciel.

2.7 Les différentes méthodes de test

On distingue essentiellement deux catégories de méthodes de tests : les tests statiques et les tests dynamiques. Les méthodes de tests statiques traitent le code du logiciel sans l'exécuter sur des données réelles, par contre les méthodes de tests dynamiques requièrent l'exécution effective du logiciel sur un sous ensemble bien choisi du domaine de ses entrées possibles.

2.7.1 Test Statique (Analyse statique)

L'analyse statique regroupe toutes les méthodes qui ne nécessitent pas l'exécution du code, l'idée générale est d'évaluer manuellement le code source afin de fournir des informations globales concernant la structure du programme, ces méthodes donnent des indications directes sur les défauts plutôt que sur l'identification des échecs (défaillances)[30]. Elle peut, par exemple, détecter :

- La dépendance dans des modèles,
- La violation de règles de sécurité et de programmation,
- La difficulté de maintenance,
- La présence de codes morts (des morceaux du programme qui ne seront jamais exécutés),
- Des variables jamais utilisées et donc inutiles
- La présence de boucles infinies.

En effet, 60% à 95% des erreurs sont détectées lors de contrôles statiques, cette approche est efficace et peu coûteuse. De plus, elle peut fournir des informations utiles pour la planification des tests, réduire le nombre de cas de test requis et augmenter la confiance dans les résultats des tests[37]. Mais, son inconvénient majeur réside dans le fait que l'aspect pratique, concernant l'exécution réelle du programme avec des vraies données de test, n'est pas considéré. Parmi les procédés qu'elle englobe :

2.7.1.1 Revue de code et lecture croisée

Le succès de la revue de code[30] dépend des participants et de leurs interactions. Typiquement, le nombre de participants par revue est entre quatre et dix participants. Ces derniers prennent des rôles différents et généralement l'un d'entre eux est le modérateur. Par contre, la lecture croisée[38] impliquant deux personnes, l'une examinant et critiquant le code source de l'autre et vice-versa. Les deux optiques partagent un objectif commun ; celui de s'assurer du respect de certains standards de codage et de l'identification de certaines pratiques de programmation suspecte suite à un examen détaillé.

2.7.1.2 L'inspection et la procédure pas à pas[39]

Le but est de détecter (to test), et non pas de corriger (to debug). Ces techniques sont menées par une équipe de trois à quatre développeurs, un seul ayant participé à l'écriture du code. Une erreur détectée ainsi correspond souvent à la correction de potentiels échecs de cas de test. En effet, ces techniques permettent de détecter essentiellement les erreurs de

conception logique ou de codage (30% à 70%), mais rarement les erreurs de conception de haut niveau. L'inspection et le pas à pas sont mieux adaptés au test de modifications de programmes.

Malgré leur utilisation fréquente, les inconvénients résident dans leurs mises en place qui sont lourdes et dans la nécessité de liens transversaux entre équipes. Ce sont, donc, des tâches plutôt fastidieuses.

En conclusion, Les méthodes de tests statiques sont nécessaires, mais non suffisantes.

2.7.2 Test Dynamique :

Les méthodes de tests dynamiques consistent à exécuter le programme sur un ensemble fini de données d'entrées (jeux de test). Elles visent à détecter des erreurs en confrontant les résultats obtenus à ceux attendus par la spécification.

Principalement deux techniques de test dynamique sont à révéler :

2.7.2.1 Test structurel (test boîte blanche ou approche boîte de verre) :

Le test structurel permet d'examiner la structure interne d'un logiciel. Entre autre, la sélection du jeu de test s'appuie sur une analyse du code source du logiciel dans le but de détecter les fautes d'implémentation. Par conséquent, de vérifier que le logiciel n'en fait pas plus que sa spécification et qu'il n'existe pas de cas de plantage (overflow, non initialisation, etc.). Cette technique établit les tests en fonction de critères de couverture en se basant sur le graphe de contrôle et le flot de données.

2.7.2.1.1 Le graphe de contrôle :

C'est un graphe orienté, constitué de nœuds représentant des blocs d'instructions et un ensemble d'arcs qui symbolise la possibilité de transfert de l'exécution d'un nœud à un autre. Les arcs peuvent être étiquetés avec un attribut (prédicat) représentant la condition du transfert de contrôle. Ce graphe possède une seul entrée (nœud d'entrée à partir duquel on peut visiter tous les autres) et une seule sortie (nœud de sortie). En voici un exemple (figure2-3) :

Un critère de couverture est une condition caractérisant un ensemble de chemins.

A chaque critère de couverture du graphe de contrôle est associée une valeur que l'on appelle taux de couverture ou mesure de complétude (Test Effectiveness Ratio(TER)). Cette valeur exprime d'une manière quantitative le degré de satisfaction du critère en question.

```
1 - Open fichier1;
    Read (x, fichier1);
    Read (y, fichier1);
    Z:=0;
2 - While x>=y do
3 – Begin
       X:=x-y;
       Z:=z+1;
    End
4 - Print(y, fichier2);
   Close fichier1 ;
   Close fichier2 ;
```

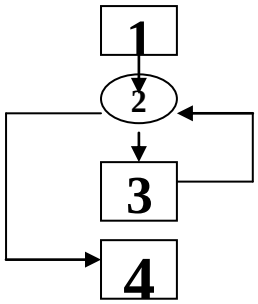


Figure 2-3: Graphe de contrôle d'un programme donné.

Elle est calculée en divisant les objets couverts par le nombre total des objets que le critère impose de couvrir (que l'on évalue par simple analyse du code). Les critères de couverture les plus utilisés sont :

a-couverture de toutes les instructions (tous les nœuds) : Chaque nœud (chaque bloc d'instructions) est atteint par au moins l'un des chemins parmi les chemins qui constituent le jeu de test. Malheureusement, le critère de chaque instruction est avéré insuffisant parce qu'il ne permet pas la détection d'un grand nombre d'anomalies.

b-couverture de toutes les branches (tous les arcs ou décisions) : Chaque arc est couvert par au moins l'un des chemins parmi les chemins qui constituent le jeu de test. Ce sont les décisions qui sont l'objet des tests. La couverture de toutes les branches équivaut à la couverture de toutes les valeurs de vérités pour chaque nœud de décision[32]. Donc, La valeur de vérité de chaque nœud de décision a été au moins une fois vraie et une fois fausse. Cependant, le critère de toutes les branches est incapable de détecter des erreurs en cas de non-exécution d'une boucle.

c-couverture de tous les chemins : selon le critère de couverture de tous les chemins, il faut tester tous les chemins exécutables menant du nœud d'entrée jusqu'au nœud de sortie. Néanmoins, ce critère est impraticable lorsque le programme contient une boucle impliquant ainsi l'existence d'une infinité de chemins à tester. Deux restrictions possibles sont à ajouter : tous les chemins limite-intérieure (ou i-chemins) et tous les chemins de taille bornée[40].

2.7.2.1.2Le flot de données :

Un graphe de flot de données s'acquiert en ajoutant au graphe de contrôle les informations liant définition (def) et utilisation (use) de variables. Une définition (def) est un emplacement où une valeur d'une variable est stockée dans la mémoire (affectation, entrée, etc.). Tandis

qu'une utilisation est un emplacement où la valeur d'une variable est accédée[26]. On distingue deux classes d'utilisations : si la variable est utilisée dans le prédicat d'une instruction de décision, il s'agit d'une p-utilisation (predicate uses), sinon il s'agit d'une c-utilisation (computation uses).

Les critères de test de flux de données utilisent le fait que les valeurs sont transférées des définitions aux utilisations. Nous appelons ceci une paire DU[26]. Dans ce cas, les critères de couverture pourront être :

a-couverture de toutes les définitions : Pour chaque définition, il existe au moins un chemin qui la couvre dans un test.

b-couverture de toutes les utilisations : consiste à couvrir au moins un chemin pour chaque paire DU.

c-couverture de tous les def-use (DU) chemins : il s'agit de couvrir tous les chemins possibles de chaque paire DU, en se limitant aux chemins sans cycle.

2.7.2.2 Test Fonctionnel (test boite noire) :

Le test fonctionnel ne considère pas la structure interne du système sous test, mais les points de contrôle et les observations externes. Il vise à vérifier la conformité du système face à ses spécifications (correction, facteurs qualité spécifiés, les performances, interfaces avec les équipements externes, etc.). Autrement dit : la sélection du jeu de test s'appuie sur les spécifications. Parmi les techniques du test boîte noire ; on retrouve les suivantes :

2.7.2.2.1Analyse partitionnelle :

La méthode de l'analyse partitionnelle est généralement dérivée de la spécification des exigences pour les attributs d'entrée qui influencent le traitement de l'objet de test. Elle consiste à établir des classes équivalentes (partitions) à partir desquelles des cas de tests peuvent être dérivés. Les données de test se baseront sur le choix d'un représentant quelconque de chaque classe.

L'analyse partitionnelle n'est pas une méthode autonome pour déterminer les cas de test. Elle est généralement complétée par l'analyse de la valeur limite (test aux limites).

2.7.2.2.2 Test aux limites :

Cette méthode s'intéresse aux bornes des intervalles partitionnant les domaines des variables d'entrées. Le test aux limites couvre un large substantif d'erreurs, car la plupart des défauts se nichent aux limites. Leur principal défaut est surtout la difficulté (et peut être l'impossibilité) de formalisation de la notion de limite et de marginalité, ce qui explique sans doute leur caractère souvent intuitif et heuristique.

2.7.2.2.3 Graphes cause-effet :

C'est un graphe orienté établissant la relation entre l'effet et ses causes. Où "cause" est une condition d'entrée et "effet" est une séquence de calculs à effectuer. Chaque condition forme un nœud, la construction du graphe se base sur quatre types de symboles usuels exprimant les interdépendances des effets aux causes (identité, négation, OU logique et ET logique)[41]. Une fois que le graphe cause-effet est établi, on le transforme en une table de décision pour qu'ensuite chaque colonne de la table de décision représente un cas de test. La méthode de test par graphes cause-effet peut s'avérer une méthode de test fonctionnel très complète et précise. Le fait qu'elle utilise un langage de représentation très formalisé rend possible l'automatisation de certaines de ses phases. Cependant, elle devient impraticable lorsque les graphes deviennent très complexes quand une fonction fait intervenir un grand nombre de causes.

2.7.2.2.4Test aléatoire (statistique) :

Lors de l'approche statistique[42], la sélection des données de test se base sur le choix aléatoire dans le domaine de la donnée d'entrée. En effet, lorsqu'une loi statistique sur le domaine est connue elle peut guider la sélection. De plus, elle permet de définir un modèle de croissance de fiabilité afin de fournir un critère d'arrêt du test. Dans ce cas, la génération se fait d'une manière probabiliste contrairement aux approches présentées précédemment (approches déterministes). L'intérêt d'une telle technique s'incarne dans le fait que la procédure de génération de jeux de tests est simplifiée et peut être facilement automatisée. Un autre intérêt consiste en l'objectivité des cas de tests produits. Cependant, son fonctionnement en aveugle (production par hasard des entrées) sensibilise des comportements très spécifiques surtout pour des programmes de taille industrielle.

Après avoir présenté les techniques de test dynamique, on insiste sur le fait que les tests fonctionnels et les tests structurels sont complémentaires et permettent de mettre en évidence différents défauts dans un programme. Les tests fonctionnels permettent principalement de détecter des défauts dus à une mauvaise compréhension des spécifications du logiciel. Les tests structurels permettent de détecter des défauts liés à la programmation.

2.8 Test par phases

Le test est un ensemble d'activités à dérouler principalement en trois phases effectuées conjointement avec l'implantation du logiciel (figure2-4).

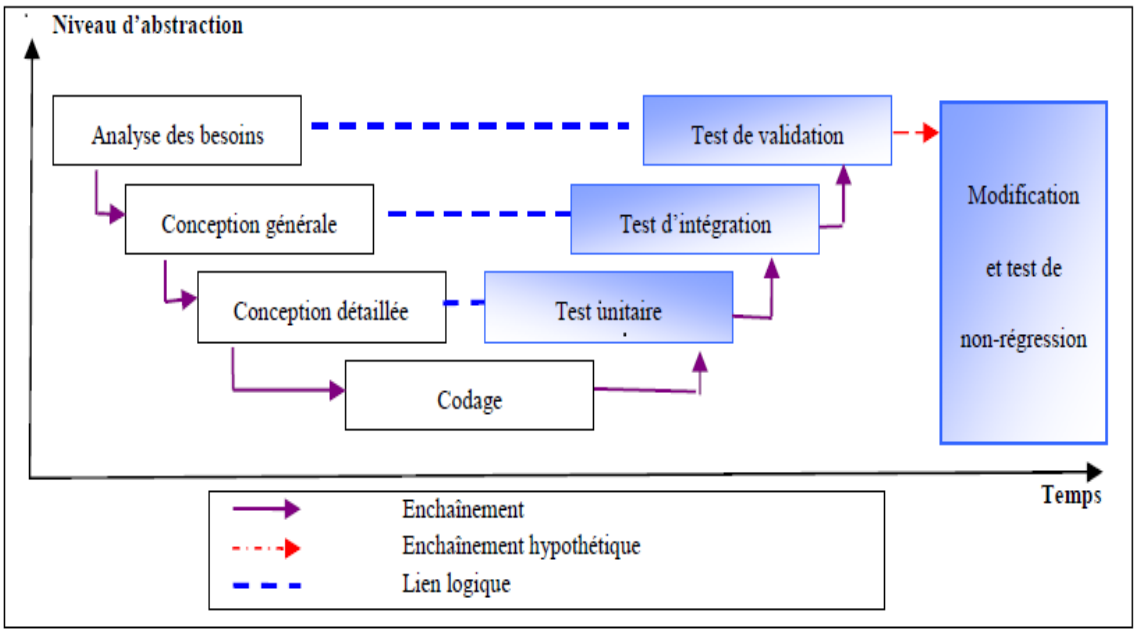


Figure 2-4 : Cycle de vie en V avec les phases de test[32].

2.8.1 Test unitaire :

Cette phase débute dès qu'un composant a été codé. Elle vise à valider la conformité de chaque composant logiciel pris unitairement par rapport à sa spécification détaillée (tester les composants un par un). Elle peut être réalisée par l'équipe de développement (mais pas par le développeur ayant codé le composant) pour les logiciels de faible criticité sur machine hôte, généralement sans banc de tests[9].

2.8.2 Test d'intégration :

Le test d'intégration est mené dès qu'un sous-système fonctionnel (module, objet) est entièrement testé unitairement. Cette phase consiste à examiner la façon dont les parties d'un système (Modules) opèrent ensemble, en particulier au niveau des interfaces. Ainsi, ce processus veille à ce que les différents composants d'un système : (1) interagissent correctement, (2) effectuent des passages de données correctement, (3) fonctionnent en cohésion. Pour effectuer un tel test, il faut adopter des stratégies spécifiques tels que : la stratégie Top-down, la stratégie Bottom-up ou la stratégie Big-bang[10].

[9] https://www.irif.fr/~eleph/Enseignement/2013-14/Tests/Cours_2.pdf (last accessed on April 2018)
[10] https://www2.informatik.hu-berlin.de/~hs/Lehre/2004-WS_SWQS/20041126_Ex_Integration-testing.pdf (last accessed on September 2017)

2.8.3 Test de validation (test système) :

Cette phase suppose que les tests unitaires et d'intégration ont été réalisés. Le test au niveau système se concentre sur les tests des fonctionnalités de plusieurs programmes, d'interfaces externes, de sécurité, de récupération et de performance. Par ailleurs, ce processus d'évaluation permet de confirmer que tous les modules fonctionnent comme prévu et que le système dans son ensemble s'exécute de manière adéquate sur la plate-forme sur laquelle il va être déployé. Enfin, il est réalisé sur un système complet et intégré pour évaluer la conformité avec les exigences spécifiées.

À côté de ces phases, il y a encore un autre test dans le cycle, c'est le **test de non-régression**[11]. Ce type de test est demandé lors d'une modification du logiciel ou de son environnement (on ne s'intéresse qu'aux parties ayant subi l'impact de la modification). Son objectif est de s'assurer que les corrections et les évolutions du code n'ont pas introduit de nouveaux défauts.

2.9 Difficultés liées aux tests

Même si les tests font l'objet de méthodes, de planning tel un véritable projet informatique, il n'en reste pas moins que certains paramètres viennent perturber leurs exécutions :

- Il est impossible de réaliser un test exhaustif.
- La qualité des tests dépend des données utilisées (données de test).

Il existe des difficultés dites formelles : il n'existe à ce jour aucun algorithme capable de prouver l'exactitude totale d'un programme.

2.10 Conclusion

Le test constitue aujourd'hui le vecteur principal de l'amélioration de la qualité du logiciel. En effet, c'est un métier à part entière. Dans l'industrie, c'est la seule activité dans le cycle de développement où l'on peut voir toutes les fonctionnalités d'un produit logiciel. Il constitue l'étape la plus coûteuse car il représente 30 à 40% des coûts de développement d'un logiciel suivant son niveau de criticité (généralement 1/3 dans le planning). Son importance nous pousse à lui octroyer un intérêt particulier et à utiliser et même à développer des outils

[11] https://www.inf.ed.ac.uk/teaching/courses/st/2011-12/Resource-folder/11_regression.pdf (last accessed on March 2017)

permettant, lorsque cela est possible, d'automatiser ou d'aider à la poursuite de ses différentes phases.

Les méthodes de test peuvent être statiques ou dynamiques, dans les méthodes dynamiques on peut trouver des techniques fonctionnelles et structurelles d'une façon complémentaire rendant ainsi le test plus efficace. Les techniques de tests logiciels peuvent être appliquées pour faire des tests unitaires, tests d'intégration, tests de sous-système, pour l'assurance qualité logicielle et aussi pour le test de système. Dans le cadre de cette thèse nous nous intéressons au test des systèmes multi-agents dans les environnements d'intelligence ambiante.

Chapitre 3

Etat de l'art sur le test des systèmes ambiants

3 Etat de l’art sur le test des systèmes ambiants

3.1 Introduction

L'omniprésence de l'informatique dans notre vie à l’échelle de l’embarqué et de l’intelligence ambiante a rendu les informaticiens de plus en plus exigeants quant à la qualité des applications. Seule une procédure de test efficace permet de répondre à ces exigences. Cette phase incontournable doit être effectuée afin d’assurer la fiabilité de tout programme. Dans ce contexte de systèmes ambiants, le test est un vrai défi, du fait de leurs caractéristiques.

A travers ce chapitre, nous allons faire un tour d’horizon de quelques travaux, qui d’une manière ou d’une autre, apportent leurs contributions aux tests des systèmes ambiants.

3.2 Test des systèmes ambiants

La technologie multi-agents est couramment utilisée dans les applications AMI. Ce n'est pas surprenant, car la nature omniprésente de l'AMI nécessite une information distribuée et une résolution de problèmes. Les systèmes multi-agents (SMA) sont connus pour faciliter de telles architectures. Les agents peuvent être utilisés comme abstractions utiles dans les systèmes AmI, par exemple pour les dispositifs et les fonctionnalités, et comme paradigmes pour la mise en œuvre[12].

Favela et al.[43] et Rodriguez et al.[44, 45], Par exemple, décrivent une architecture appelée SALSA, pour les soins de santé, qui utilise des agents comme des abstractions, pour agir au nom des utilisateurs, représenter des services et pour cacher des fonctionnalités complexes à l'utilisateur. Amigoni et al.[46] décrivent une approche mixte centralisée et distribuée de la planification dans un environnement AmI où les périphériques sont représentés par des agents, et entrent et sortent de l'environnement.

La figure 3-1 illustre la multidisciplinarité de L’intelligence ambiante.

Donc, vu l’importance des SMA dans le cadre de l’intelligence ambiante, il est convivial de dire qu’on s’intéresse au test des systèmes multi-agents dans les environnements

d'intelligence ambiante et plus particulièrement aux agents ambiants à travers toutes ses étapes.

Dans ce qui suit, nous présentons les niveaux de test des SMAs.

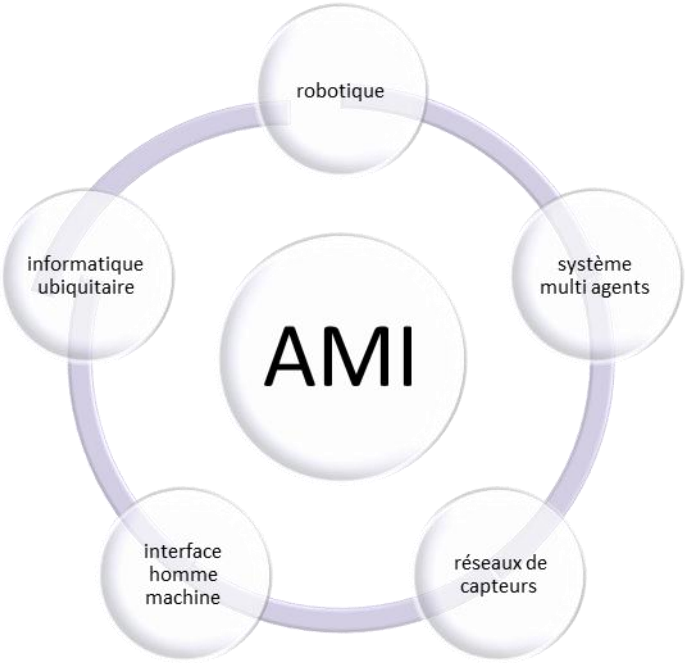


Figure 3-1: Les axes de l'AMI[47].

3.3 Niveaux de test des systèmes multi-agents (SMA)

Le test des systèmes multi-agents peut être classé en différents niveaux de test[48], les sujets à tester et les activités de chaque niveau sont décrites comme suit :

3.3.1 Niveau unité

A ce niveau, on teste tous les composants d'un agent (base de connaissance, moteur de raisonnement, les plans, les objectifs ... etc.). Et s'assurer qu'elles fonctionnent comme prévu.

3.3.2 Niveau agent

Le niveau agent considère les interactions entre un groupe de classes coopérants. Tester un agent consiste à tester ses fonctionnalités internes et celles qu'il met à la disposition des autres agents, en prenant en considération, ses propres objectifs. Lors du test de ce niveau d'abstraction, un certain nombre d'erreurs susceptibles d'être détectées peuvent être résumées dans les points suivants[49] :

- Transfert d'un message incorrect vers un agent,

- Placement d'une requête incorrecte dans un message qui ne sera pas reconnu par l'agent interceptant,
- Analyse incorrecte des messages entrants,
- Détection d'une faute performative au niveau d'un message entrant.

3.3.3 Niveau société d'agents

Le test de ce niveau est une sorte de test d'intégration et la stratégie d'intégration dépendra de l'architecture du système multi-agents où les dépendances entre agents sont exprimées en termes de communications avec parfois des interactions de médiations environnementales.

3.3.4 Niveau système

Les agents peuvent opérer correctement s'ils sont exécutés séparément mais ils peuvent manifester des défauts s'ils sont mis en commun. Le test du niveau système implique l'assurance que tous les agents dans le système opèrent selon les spécifications. On peut également au niveau du test système vérifier les principaux types d'interactions agents (coopération, négociation et coordination) ou faire appel à d'autres types de tests tels les tests de performance, de conformité, les tests de chargement et/ou de stress, etc.

3.3.5 Niveau d'acceptation

A ce niveau, on teste les SMA dans l'environnement d'exécution du client et on vérifie qu'il répond aux objectifs des parties prenantes, avec la participation de ces dernières.

3.4 Panorama des approches de test des systèmes multi-agents

Cette section présente une classification des divers travaux sur le test des SMA en respectant les catégories précédentes.

3.4.1 Niveau unitaire

- Zhang et al.[50], ont proposé un Framework de test basé sur un modèle, en utilisant les modèles de conception de la méthodologie de développement de l'agent.
- Ekinci et al.[51], affirment que les objectifs de l'agent sont les plus petites unités testables en SMA et ont proposé de tester ces unités par le biais des objectifs de test.

3.4.2 Niveaux agent

- Tiryaki et al.[52] proposent une approche test-conduite de développement de SMA qui soutient la construction itérative et par accroissement de SMA. Un cadre de test appelé

SUnit, qui a été construit sur JUnit[12] et Seagent[53], a été développé pour soutenir l'approche. Ce cadre permet des écritures de tests pour des comportements d'agent et des interactions entre les agents.

- Coelho et al.[54], proposent un outil JAT similaire à SUnit mais qui est basé sur l'utilisation d'agents simulés (Mock Agent).
- Lam et Barber[55], proposent un processus semi-automatisé pour comprendre les comportements d'agents logiciels. L'approche imite ce qu'un utilisateur humain fait dans la compréhension du logiciel : construction et raffinement d'une base de connaissance au sujet des comportements des agents puis emploi pour vérifier et expliquer les comportements des agents et leur temps d'exécution.
- Nunez et al.[56], présentent un cadre formel pour spécifier le comportement des agents autonomes de commerce électronique. Les comportements désirés des agents sous test sont présentés au moyen d'un nouveau formalisme, appelé machine d'état de service qui incarne les préférences des utilisateurs dans ces états. Deux méthodologies de test sont proposées pour vérifier si une exécution d'agent spécifique se comporte comme prévu (c.à.d. test de conformité) ; un actif et l'autre passif. Dans leur approche de test actif, ils emploient pour chaque agent sous test un agent spécial (agent test) qui prend les spécifications formelles de l'agent pour lui faciliter l'atteinte d'un état spécifique. La trace opérationnelle de l'agent est alors comparée aux spécifications afin de détecter des défauts. D'autre part, les auteurs proposent également d'employer le test passif, dans lequel les agents sous test sont seulement observés et non simulés comme dans le test actif. Les traces invalides, si elles existent, sont alors identifiées grâce aux spécifications formelles des agents.
- Enfin, Nguyen et al.[57], présentent et évaluent une approche pour tester les agents autonomes. Cette méthode prône l'optimisation évolutionnaire pour produire des cas de test exigeants. Dans ce travail, les auteurs ont proposé une manière systématique d'évaluer la qualité des agents autonomes. D'abord, des exigences de dépositaire sont représentées comme qualité des mesures puis les seuils correspondants sont employés comme critères de test. Les agents autonomes doivent répondre à ces critères afin d'être fiables. Des fonctions de forme physique qui représentent des objectifs de test sont définies en conséquence et guident cette technique de génération de test évolutionnaire pour produire des cas de test automatiquement.

[12] E. Gamma, and K. Beck. "JUnit: A Regression Testing Framework", 2000, available: http://www.junit.org

3.4.3 Niveau société d'agents

- Padgham et al.[58] ont utilisé des outils de conception (Prometheus design) pour tester les protocoles d'interaction et les spécifications du plan pour fournir une identification automatique de la source des erreurs détectées lors de l'exécution.
- Rodrigues et al.[59] ont proposé d'exploiter les conventions sociales, à savoir les normes et les règles, qui prescrivent des autorisations, obligations, et / ou des interdictions d'agents dans un SMA ouvert à des tests d'intégration.
- Finalement, Serrano et Batia[60] ont utilisé l'ACL analyser : équipement exécuté dans la plateforme JADE[13]. Il intercepte tous les messages échangés entre les agents et les stocke dans une base de données relationnelle.

3.4.4 Niveaux système

- L'approche d'analyse empirique De Wolf et al.[61], combine la simulation multi-agents et les algorithmes numériques pour analyser le comportement global d'un système d'auto-organisation.
- Sudeikat et Renz[62], ont proposé d'utiliser la dynamique du système de modélisation comme notion pour la validation du SMA. Ceux-ci permettent de décrire les comportements observables, destinés macroscopiques qui proviennent de structures de victimes cycliques. La simulation des systèmes est ensuite utilisée pour mesurer les valeurs de l'état du système afin d'examiner si les causalités sont observables.
- Enfin, Houhamdi et Athamena[63], proposent une approche qui définit un processus de suite de dérivation de tests d'intégration structuré et complet pour les agents de logiciels d'ingénierie en fournissant un moyen systématique de dériver des cas de test à partir de l'analyse des objectifs.

3.4.5 Niveaux d'acceptation

Au meilleur de notre connaissance, il n'y a pas de travail traitant explicitement le test des SMAs au niveau de l'acceptation pour le moment. En fait, l'agent, l'intégration et le niveau système peuvent être réutilisés dans le test d'acceptation. Cependant, comme les objectifs de test d'acceptation diffèrent de ceux des niveaux inférieurs par les paramètres d'évaluation comme les paramètres pour l'ouverture, la tolérance aux pannes et l'adaptabilité, la demande pour de nouvelles recherches est indéniablement nécessaire.

[13] http://jade.tilab.com/

Chapitre 3 Etat de l'art sur les systèmes ambiants

3.5 Panorama des travaux existants de test des systèmes multi-agents dans les environnements d'intelligence ambiante

Malgré la popularité des systèmes ambiants, les techniques de test dédiées sont encore peu nombreuses.

Dans les applications d'apprentissage en intelligence ambiante, un framework nommé AmITest a été proposé dans[64], il permet de tester et de valider les programmes comportementaux écrits par les utilisateurs de manière simple, directe et efficace. AmITest fait partie d'une suite complète de développement d'utilisateurs finaux appelé AmIClass, qui permet la programmation efficace des environnements éducatifs AmI par des professionnels non-informaticiens. La figure 3-2 montre l'architecture générale du framework AmITest. Un telle framework est constitué de :

- (1) les agents de test ClassScript (ClassScript Testing Agent : CTA), qui sont installés sur chaque artefact et qui sont responsables de l'installation et de l'exécution locale des scripts de test,

-et (2) une suite Web maître, la suite de gestion et de déploiement des tests (Management and Deployment Suite : TMDS), responsable de la création, de la supervision et de la gestion de la procédure de test.

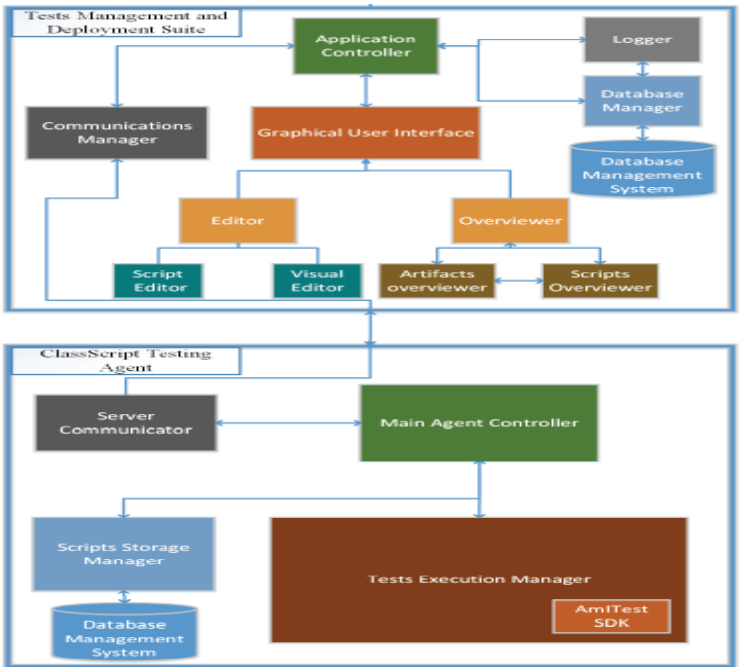


Figure 3-2: Architecture du framework AmITest[64].

En outre, une approche dirigée par les modèles, ou model-driven development (MDD) en anglais, pour réutiliser les tests dans les systèmes domestiques intelligents a été proposé dans[65], cette dernière se concentre sur la génération systématique de cas de test pour les systèmes de maisons intelligentes (Smart Home : SH). Concrètement, elle permet de définir des modèles de test en dehors de la spécification du système SH et de les intégrer en incorporant des cas de test au système final au moyen d'un processus de tissage générique. La figure 3-3 montre les différentes activités et les produits générés impliqués dans ce processus.

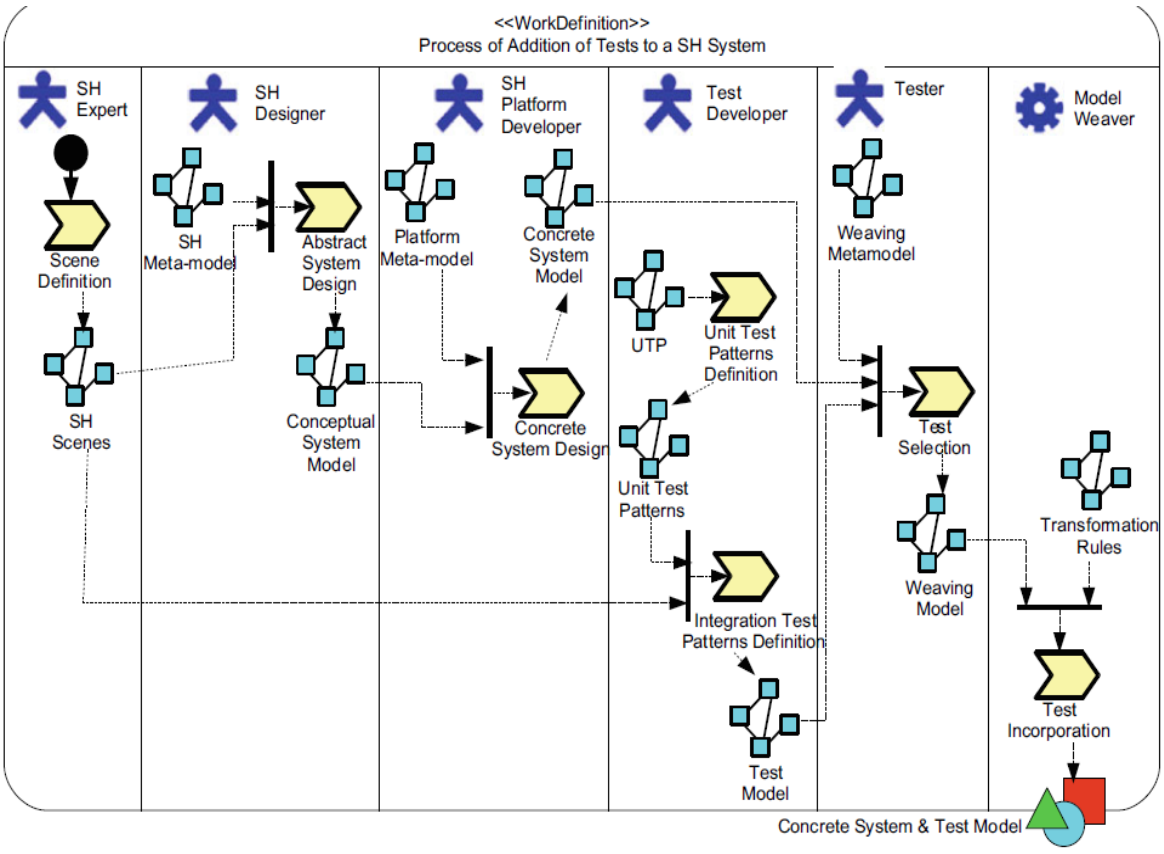


Figure 3-3: Processus de spécification et de test du système de maison intelligente MDD[65].

Par ailleurs, la plupart des applications d'intelligence ambiante nécessitent des stratégies et des solutions d'accès et de gestion des données adaptées au contexte. C'est pour cela qu'on trouve dans la littérature des approches de test des systèmes ambiants traitant précisément cet aspect-là : la sensibilité au contexte.

A titre d'exemple, Xu et al.[66], ont proposé des modèles d'erreurs de surveillance pour suivre les défauts responsables dans l'adaptation au contexte. Tse et al.[67], se sont appuyés

sur des relations métamorphiques pour décider si les contextes et leurs applications de couche supérieure se comportent anormalement. Ramires et al[68], ont proposé de découvrir des combinaisons spécifiques des conditions environnementales qui produisent des comportements violés dans les systèmes adaptatifs.

Ces travaux utilisaient différentes observations, mais en général, ils reposaient encore sur des tests aléatoires. Cela implique qu'ils ne garantissent pas l'exploration systématique de l'espace d'une application.

Certaines approches de test d'applications contextuelles prennent en compte l'interaction entre les applications et leurs environnements. Griebe et al.[69], ont proposé une approche de test basée sur un modèle qui génère des cas de test en utilisant la transformation de modèle sur des modèles de système au moment de la conception enrichis par le contexte. Amalfitano et al.[70], ont utilisé des modèles d'événements prédéfinis pour générer des traces d'événements de contexte afin d'explorer différents comportements pour les applications contextuelles. Jiang et al.[71], ont proposé un framework pour simuler à la fois la production de contextes de données et l'exécution de services pour tester des applications contextuelles. Le framework est également capable de générer des données de contexte simulées basées sur des composants opérationnels spécifiés par l'utilisateur, tels que des capteurs virtuels, des dispositifs, et même des êtres humains. Wang et al.[72], ont amélioré la couverture de test pour les applications contextuelles en utilisant des points de changement de contexte dans les applications. Lu et al.[73], ont proposé une famille de critères d'adéquation des tests pour couvrir les nouveaux flux de données induits par les interactions entre une application et son intergiciel sous-jacent. Ils ont ensuite étendu le travail pour prendre en charge les applications de test avec des services de résolution d'incohérences externes[74], qui nécessitent de nouveaux critères de couverture.

En revanche, une nouvelle approche baptisé SIT (Sampling Interactive Test : Test Interactif à Base d'Échantillons) pour tester efficacement et de manière légère les applications auto-adaptatives a été proposé dans[75]. Cette dernière explore l'espace d'entrée d'une application et son comportement correspondant d'une manière systématique et guidée. En outre, elle considère l'incertitude dans les interactions, qui incluent à la fois la détection environnementale et l'adaptation comportementale.

3.6 Conclusion

Les systèmes ambiants forment un domaine d’application très particulier, car ils sont très dynamiques et en interactions constantes avec des utilisateurs. Les systèmes multi-agents sont utilisés comme abstractions utiles dans ces systèmes, par exemple pour les dispositifs et les fonctionnalités, et comme paradigme pour la mise en œuvre.

Cependant, les caractéristiques de tels systèmes rendent leur test un vrai défi qui demande de nouvelles méthodes de test efficaces et adéquates pour évaluer les comportements autonomes des agents ambiants et leurs interactions. En outre, les méthodes doivent être soutenues par des modèles de bases appropriés à leurs fondements théoriques, outillés pour une utilisation pratique et des mesures pour leur gestion et leur évaluation. Le test basé sur les modèles, ou model based testing (MBT) en anglais, est un bon exemple de type de processus à adopter, en particulier avec l’avènement d’un nouveau paradigme du génie logiciel baptisé : ingénierie dirigée par les modéles, qui rend le développement et l’évolution des logiciels plus faciles.

Le chapitre suivant présente l'ingénierie dirigée par les modéles. Nous nous intéressons Particulièrement au test basé sur les modéles.

Chapitre 4

L'ingénierie dirigée par les modéles

4 L'ingénierie dirigée par les modèles

4.1 Introduction

L'ingénierie dirigée par les modèles (IDM ou en anglais MDE pour Model Driven Engineering), est une approche de plus en plus présente dans le monde industriel et universitaire. Cette présence a fortement contribué à faire émerger de nouveaux concepts et un ensemble d'outils permettant de changer la façon dont nous construisons, nous exploitons et nous maintenons nos systèmes à logiciel prépondérant. L'intérêt de cette approche, est de considérer le modèle comme un point de référence tout au long du cycle de développement du logiciel. L'IDM apporte aussi une grande part d'automatisation des processus de développement, essentiellement grâce aux transformations de modèles.

Le présent chapitre a pour ambition de présenter les principes clés de l'IDM, un intérêt particulier sera accorder à la technique du test basé sur les modèles.

4.2 Qu'est ce qu'un modèle?

Le concept central de l'IDM est la notion de modèle, pour laquelle plusieurs définitions ont été évoquées [76]. Pour résumer nous retenons celle-ci : Un modèle est la simplification d'un système dans l'objectif de répondre à certaines questions à la place du système qu'il représente.

4.3 Pourquoi modéliser ?

La modélisation des systèmes informatiques est devenue de plus en plus une nécessité méthodologique, notamment pour le développement et la validation des systèmes complexes. Elle est parfois vue comme le moyen d'exprimer une solution à un plus haut niveau d'abstraction que le code.

En effet, Modéliser un système avant sa réalisation nous aide à mieux comprendre le fonctionnement du système et à prédire son comportement. C'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence.

Dans le domaine de l'IDM, le modèle permet de mieux répartir les tâches et d'automatiser certaines d'entre elles. C'est également un facteur indispensable pour améliorer la qualité des logiciels avec des réductions de coûts importantes sur la production et sur la maintenance.

4.4 L'exemple MDA

L'ingénierie dirigée par les modèles s'appuie principalement sur l'initiative Model-Driven Architecture (MDA), menée en 2000 par l'OMG[14]. L'idée astucieuse de cette initiative est de séparer les besoins applicatifs d'un système des détails de la plateforme utilisée. Pour cela, MDA définit une architecture de spécification structurée en plusieurs types de modèles.

Le MDA préconise l'élaboration de modèles en se basant sur le standard UML, Plus précisément, le MDA définit deux principaux modèles :

- d'analyse et de conception (Platform Independent Model – PIM),
- de code (Platform Specific Model – PSM).

L'objectif majeur du MDA est l'élaboration de modèles pérennes (PIM), indépendants des détails techniques des plateformes d'exécution (J2EE, .Net, PHP, etc.), afin de permettre la génération automatique de la totalité des modèles de code (PSM) [77]. Le passage de PIM à PSM est le résultat de transformations automatiques.

4.5 Le test et l'IDM

L'IDM permet de couvrir les différentes phases du cycle de développement logiciel (conception, génération de code. . .). Le test ne fait pas exception : en effet l'IDM couvre aussi cette phase. Dans[78] l'intérêt de l'IDM à travers plusieurs exemples pris à différents niveaux du cycle de développement du logiciel est donnée. Ces exemples ont permis d'illustrer les points forts de l'IDM pour le génie logiciel à savoir : l'utilisation intensive, la réutilisation et la transformation de modèles.

Le test consiste à détecter des erreurs dans un système, ce qui se traduit par une série d'activités dont le but est d'observer un comportement du système différent de celui attendu. Dans la plupart des cas, le comportement attendu est décrit dans un document de spécification et le système sous test est une implantation qui doit être conforme à cette spécification.

Dans le cas de l'ingénierie dirigée par les modéles, la spécification et le système à tester peuvent être tous les deux des modéles.

[14] Object Management Group (OMG), OMG Model Driven Architecture, 2016 http://www.omg.org/mda/

Un modèle de test, comme un modèle de conception, doit être conforme à un langage de modélisation. Le choix de ce langage dépend des besoins du testeur et des objectifs du test lui-même [79]. De manière naturelle, la définition d'un langage de modélisation a pris la forme d'un modèle, appelé méta modèle.

Les activités pour le test sont nombreuses et font toutes l'objet d'importants travaux pour les mettre au point et les rendre efficaces. Les activités principales sont : la génération de cas de test, l'exécution de ces tests sur le système. Dans la suite nous nous intéressons surtout à la technique largement étudiée pour ces activités appelée test basé sur les modèles.

4.6 Test basé sur les modèles

Le test basé sur les modèles, ou Model Based Testing (MBT) en anglais, constituent une innovation que l'on pourrait qualifier de révolutionnaire dans le domaine des tests de logiciels car il automatise complètement le processus de test.

Le MBT est une technique qui s'appuie sur un modèle du système afin d'en produire des cas de test fonctionnels à partir d'exigences relatives au niveau des tests souhaités. La figure 4-1 exhibe le principe d'une telle technique.

Un processus MBT inclut généralement les étapes suivantes [80] :

- Construction d'un modèle abstrait du système sous test,
- Validation du modèle,
- Génération de cas de test abstraits à partir du modèle,
- Raffinement de ces tests abstraits en tests concrets et exécutables.

Ceci fait, les tests concrets peuvent être exécutés sur le système testé, afin de détecter les défaillances (lorsque les sorties du système testé sont différentes de celles prédites par les tests).

Les solutions MBT existantes sont nombreuses et se distinguent notamment par le type de modèle, la méthode de génération des cas de test ou encore le type d'exécution de ces tests.

Ces étapes chronologiques sont décrites dans ce qui suit.

4.6.1 Modélisation du système sous test

La production d'un modèle dédié au test est l'activité principale d'un utilisateur de la technique de test basée modèle. Il s'agit de concevoir un modèle duquel seront produits les cas de test à exécuter sur le système sous test, ou system under test en anglais (SUT). Ce modèle représente les comportements du système à un certain niveau d'abstraction. L'objectif du

modèle de test est double : D'une part, il spécifie les comportements attendus du système sous test et simplifie ainsi la phase verdict de test. Cette phase consiste à comparer les résultats obtenus suite à l'exécution du système avec les résultats attendus. D'autre part, sa structure est exploitée lors de la génération des cas de test pour garantir une certaine couverture du modèle et donc de la spécification fonctionnelle dont il est issu. Typiquement, les différentes familles de critères de couverture structurelle (citées dans le chapitre 2) peuvent être employées pour garantir une couverture de modèle donnée.

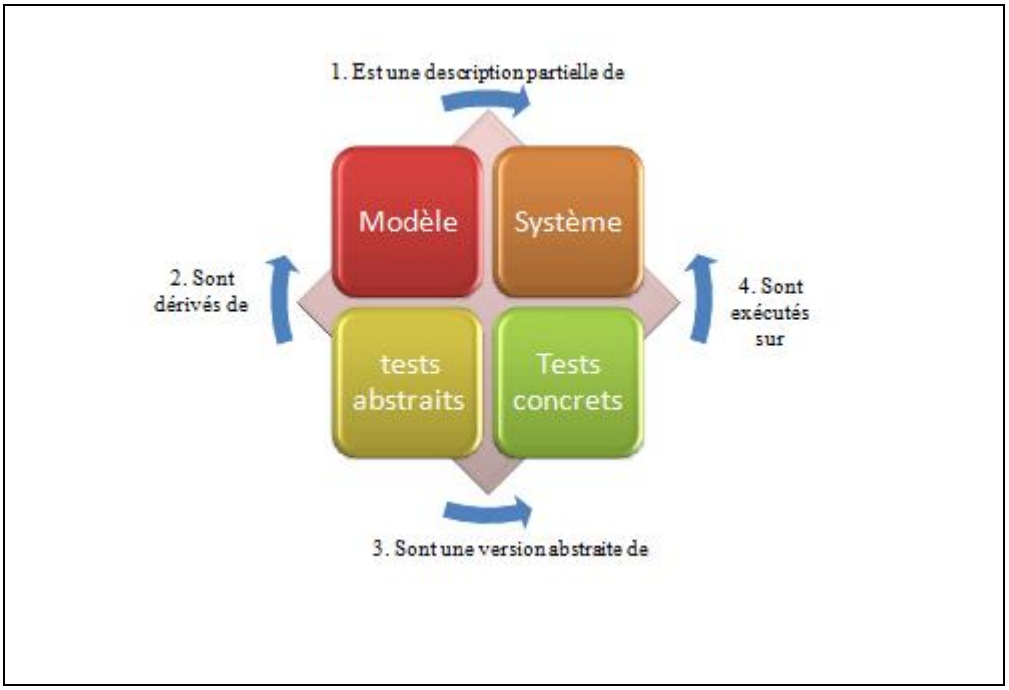


Figure 4-1: Principe d'une technique MBT.

4.6.2 Validation du modèle

Dans le but de déceler d'éventuelles anomalies pouvant se glisser dans le modèle obtenu, sa validation vis-à-vis de la spécification de départ est plus qu'obligatoire. Ce processus de validation est certes incomplet, mais les retombées des erreurs omises sont moins cruciales dans ce contexte que dans le contexte de raffinement des modèles en code. Pour assurer cette validation, plusieurs solutions sont possibles, comme la simulation, l'exploration, l'animation, l'analyse statique et les preuves. Avec le test basé sur les modèles, si quelques erreurs n'ont pas été détectées lors de la validation du modèle, il le seront sûrement lors de la soumission des cas de tests à l'application sous test.

4.6.3 Génération ou la production de cas de test abstraits à partir du modèle

Dans un processus MBT, la production des cas de test est la plupart du temps automatisée par des outils de génération automatique de tests. Cette génération s'appuie sur les

comportements issus du modèle de test et sur des critères de sélection de test pour décider quels tests nous voulons générer à partir du modèle, car il y a généralement un nombre infini de tests possibles. La figure 4-2 montre un processus générique de génération de cas de test.

Une suite de test peut être générée automatiquement au moyen d'algorithmes de génération basés sur différentes techniques : algorithmes de recherche par graphes, model-checking, exécution symbolique ou encore par preuve déductive.

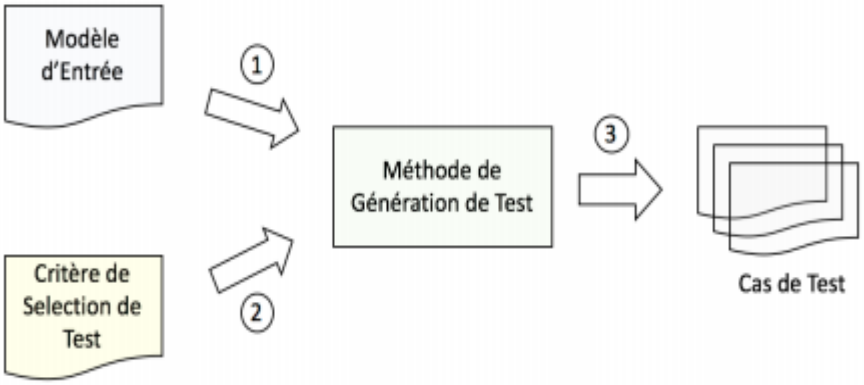


Figure 4-2 : Processus générique de génération de cas de test[81].

Un résumé de quelques techniques de génération de cas de test ainsi que leurs forces et faiblesses est présenté dans le tableau 4-1 ci-dessous.

	Présentation	**Atouts**	**Faiblesses**
Méthode Aléatoire	Sélection aléatoire de tests à partir des domaines d'entrée	Implémentation facile et pas coûteuse	Pas souvent efficace en termes de couverture (ex, MC/DC)
Vérification de Modèle	Vérification de propriétés sur un modèle	Vérification formelle des propriétés sur un modèle	Technique coûteuse ; Difficulté dans la définition des propriétés
Algorithmes de graphes	Techniques de parcours de graphes	Diversité d'algorithmes ; Convient aux graphes	Complexité de certains algorithmes
Exécution Symbolique	Exécution d'un modèle avec des valeurs génériques (symboliques)	Exploration chemins du modèle ; Détection chemins faisables, infaisables	Résolution de certains types de contraintes ; Explosion des chemins à explorer

Tableau 4-1 : Comparaison des techniques de génération de cas de test[81].

4.6.4 Raffinement de ces tests abstraits en tests concrets et exécutable ou concrétisation

La concrétisation est l'étape la plus délicate dans le processus de test basé sur les modèles. Cette étape agit comme un traducteur qui transforme les cas de tests abstraits en cas de tests concrets. Par la suite, ces cas de tests concrets peuvent être soumis à l'application sous test. Le translateur ayant pour but de réduire le gap entre le modèle abstrait et le système concret en ajoutant des informations manquantes et en transformant les entités abstraites en des structures du langage de la plateforme de test.

Après une telle étape, on peut dire que le système est prêt à être exécuté, cette exécution peut être manuelle à l'aide d'une personne physique, ou automatique via un environnement dédié (banc de test), qui fournit des facilités pour exécuter automatiquement les cas de test et pour enregistrer les verdicts. Plus précisément, les tests concrets retenus lors de l'étape précédente sont exécutés sur le système sous test et une décision est prise sur leur conformité par rapport à ceux prévues par le modèle (notion d'oracle). Cependant, la décision n'est pas toujours simple, ni même possible. Si l'oracle a détecté une défaillance, il faut la corriger et réexécuter le même cas de test pour vérifier que la correction a effectivement éliminé la faute. Si l'oracle n'a pas détecté de défaillance, on à deux directions soit le critère d'arrêt est satisfait et la phase de test du logiciel est terminée ; soit un nouveau cas de test sera considéré.

4.7 Avantages du processus MBT

Le test basé sur les modèles a plusieurs avantages : D'abord, le modèle de test est habituellement tout à fait petit, facile à comprendre et facile à maintenir. En outre, le MBT permet d'améliorer la détection des défauts du SUT et réduire le coût de la phase de tests. De plus, il peut amener à réduire le temps et les efforts consacrés à tester. Dans ce cas le processus du model-based testing est rentable, car la génération des scripts de test est automatisée et il rend plus facile la gestion de l'évolution des exigences en ne modifiant que le modèle et en régénérant les cas de test, plutôt qu'en maintenant la suite de test en soi. Cela permet de réduire considérablement le coût de la maintenance de test.

4.8 Limitations du processus MBT

Une limitation pratique du MBT est qu'il nécessite des testeurs maîtrisant la modélisation. Par ailleurs, une autre limitation réside dans le fait qu'il est une mauvaise métrique dans le sens qu'il ne pourra jamais détecter tous les défauts ainsi que l'explosion combinatoire de cas

de test. Enfin, l'inconvénient majeur est le coût de construction du modèle de test abstrait du système sous test.

4.9 Outils MBT

Il existe actuellement un large spectre d'outils de test basés sur des modèles commerciaux et académiques manipulant toute une variété de méthodes et de notations. La plupart des outils permettent à l'ingénieur de test de guider le processus de génération de tests pour contrôler le nombre de tests produits ou pour concentrer les efforts de test sur certaines zones du modèle. Nous présentons ici quelques outils récents (voir Tableau 4-2) et bien connus dans le milieu industriel.

Outil	**Modèle d'entrée**	**Description**	**Type**
MaTeLo	chaînes de Markov	MaTeLo est l'acronyme de Markov Test Logic développé par ALL4TEC. C'est un outil commercial de génération automatique de tests fonctionnels et de validation, leur Stratégies : génération aléatoire orientée par profils, couverture de toutes les transitions. Peut être connecté à de nombreuses plateformes de test.	Commercial
TEMPPO	Modèle de flux de tâches	Cet outil a été initialement développé pour modéliser les interfaces graphiques, mais il a ensuite été étendu aux tests API. Il supporte plusieurs Framework d'exécution de tests externes. Il a été utilisé par l'Agence spatiale européenne (ESA).	Commercial
ModelJUnit[15]	machine à	ModelJUnit vous permet d'écrire	Open source

[15] http://sourceforge.net/projects/modeljunit/

	états finis étendue	des modèles de machine à états finis (FSM) ou des modèles EFSM (Extended Finite State Machine) en classes Java, puis de générer des tests à partir de ces modèles et de mesurer différentes métriques de couverture de modèle.	
SmartTesting CertifyIt	UML /BPMN + OCL	CertifyIt est un outil complet de MBT utilisant des modèles comportementaux (en UML / BPMN avec des contraintes OCL) et un large éventail de critères de couverture (couverture des exigences, couverture structurelle du modèle, couverture basée sur les patterns).	Commercial
MISTA	Réseaux de Petri	MISTA génère des cas de test à partir de réseaux de Petri de haut niveau, il peut générer du code de test exécutable pour diverses plates-formes (JUnit, NUnit, Selenium ...). Il peut être utilisé pour des tests fonctionnels ou de sécurité.	Académique

Tableau 4-2: Quelques outils MBT.

Dans le cadre de cette thèse, nous avons développé un outil académique MATT (Multi Agent Testing Tool) basé sur un modèle nommé les réseaux dans les réseaux (nets within nets en anglais). Ce dernier permet d'automatiser la phase de génération et de concrétisation des cas de test tout en faisant l'instrumentation au niveau du modèle. Contrairement aux différentes approches de test, où l'instrumentation porte la plupart du temps sur le code source

de l'application. Aussi, l'exécution se fait d'une manière automatique en se basant sur Renew [82] et en couvrant tous les niveaux de test d'un système multi-agents.

4.10 Conclusion

Au cours de ce chapitre nous avons introduit les principes généraux de l'IDM et nous nous sommes principalement intéressés à la technique de test particulière MBT.

L'idée de base du MBT est de disposer d'un modèle du système et de l'utiliser pour générer des séquences d'entrées et des séquences de sorties escomptées. L'entrée est soumise au système sous test et la réponse du système est comparée à celle issue du modèle.

En revanche, Les méthodes formelles fournissent des outils permettant de garantir l'absence de certaines erreurs. Ces méthodes sont indispensables pour assurer les plus hauts niveaux de sûreté. De plus, L'utilisation des approches formelles, consiste en une parfaite automatisation de la phase de l'oracle pour pouvoir statuer sur la conformité des résultats obtenus par-rapport aux résultats attendus. En outre, Ces méthodes sont basées sur des concepts mathématiques permettant à la fois de spécifier de manière non-ambigue le comportement d'un système en le décrivant avec un modèle formel et de vérifier de manière rigoureuse son bon fonctionnement.

Plusieurs formalismes ont été utilisés pour décrire un modèle, on s'intéresse plus particulièrement aux réseaux de Petri.

Chapitre 5

Les réseaux de Petri

5 Les réseaux de Petri

5.1 Introduction

L'utilisation des méthodes formelles s'est accrue ces dernières années, surtout dans le développement des logiciels les plus critiques à travers les différentes étapes du cycle de vie : la phase d'analyse, de conception et de validation. Ainsi, ces méthodes permettent d'obtenir une très forte assurance de l'absence de bugs dans les logiciels en repérant et en corrigeant les erreurs assez tôt durant le processus de développement tout en respectant les exigences du système et en évitant les erreurs possibles.

L'application des méthodes formelles, implique une phase de spécification du système qui consiste à décrire ce dernier à l'aide d'un modèle théorique. Cette phase repose sur un langage formel basé sur une syntaxe claire et précise avec une sémantique bien définie. Le choix de ce langage dépend du type de système à spécifier (fini, infini, séquentiel, ou concurrent).

Étant donné le nombre important de formalismes existants (automates, logique temporelles, réseaux de Petri…..etc.), nous nous sommes intéressés plus particulièrement aux réseaux de Petri vue la puissance, la précision et l'expressivité de ces outils. Une description des réseaux de Petri est donnée dans ce qui suit.

5.2 Les réseaux de Petri

Aujourd'hui, les réseaux de Petri qu'on abrège par RdP constituent l'un des modèles formels les plus avancés et les plus complets vus qu'ils offrent un outil à la fois graphique et mathématique permettant de modéliser le comportement dynamique des systèmes à évènements discrets, notamment les systèmes concurrents et parallèles. L'intérêt primordial de ces réseaux réside dans leur possibilité d'analyser les systèmes modélisés.

Pour mieux comprendre ce formalisme, différents concepts régissant les fondements de cette théorie sont présentés dans ce qui suit.

5.2.1 Définitions

Les réseaux de Petri ont été introduits par Carl Adam Petri[83], dans sa thèse "Communication avec des Automates" à l'université Darmstadt en Allemagne, en 1962. Nous présentons leur définition d'une manière formelle et informelle.

5.2.1.1 Définition informelle

Intuitivement, un réseau de Petri est un graphe biparti qui comporte des places, représentées par des cercles modélisant des conditions représentant les états possible des systèmes et des transitions, représentées par des rectangles ou des traits modélisant des évènements qui sont des actions qui se déroulent dans le système.

Les arcs relient les places et les transitions. Un arc à l'entrée d'une transition désigne une pré- condition et un arc en sortie d'une transition désigne une post-condition. Une pré ou post-condition signifie une condition qui doit être vérifiée pour franchir la transition en question. Chaque place contient un nombre positif ou nul de jetons.

A chaque arc est associé un nombre entier strictement positif appelé poids de l'arc. Le réseau de Petri dont tous ses arcs sont de poids "1" est appelé réseau de Petri ordinaire. Dans le cas où les arcs peuvent avoir des poids supérieurs à "1", il s'agit d'un réseau de Petri généralisé.

Un marquage est la répartition d'un nombre de jetons dans chaque place. Ce marquage définit un état du réseau.

La figure 5-1, nous montre un exemple d'une représentation graphique d'un réseau de Petri.

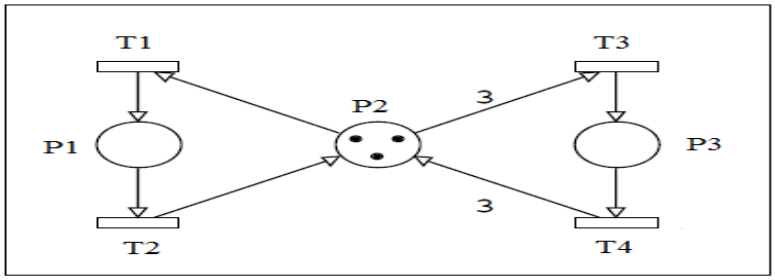


Figure 5-1 : Exemple d'un réseau de Petri[84].

5.2.1.2 Définition formelle

Formellement, un réseau de Petri est défini par un quadruplet R[85] tel que : R = (P, T, Pré, Post) avec :

- P : ensemble des places du réseau ;
- T : ensemble des transitions du réseau ;

- Pré : application d'incidence avant, de P × T dans N. ∀p ∈ P, ∀t ∈ T, Pré (p, t) = v (p, t) si (p, t) ∈ G, 0 sinon.
- Post : application d'incidence arrière, de P × T dans N. ∀p ∈ P, ∀t ∈ T, Post (p, t) = v(t, p) si (t, p) ∈ G, 0 sinon.

A chaque application d'incidence est associée une matrice |P| × |T|.

Ceci dit, on peut ainsi définir :

- la matrice d'incidence du réseau C= Post – Pré.
- Le marquage M qui est une application de P →N, ce marquage nous permet de connaitre le nombre de jetons dans chaque place. M0 est en général appelé le marquage initial.

Pour le réseau ci-dessus (figure1), P = {p1, p2, p3} T = {t1, t2, t3, t4}, la représentation matricielle est donnée ci-dessous (figure5-2).

$$\bullet Pré = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \bullet Post = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\bullet C = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 3 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \qquad M_0 = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}$$

Figure 5-2: Matrice d'incidence et vecteur de marquage M0 du réseau de Petri de la figure 1.

5.3 Modélisation des systèmes complexes par les réseaux de Petri

Grâce aux réseaux de Petri, il est possible de modéliser des comportements très divers dans les systèmes complexes tels que : la synchronisation, le parallélisme et le partage de ressources.

5.3.1 Synchronisation

Deux formes de synchronisation peuvent être modélisées :

5.3.1.1 Synchronisation mutuelle (Par rendez-vous)

La synchronisation mutuelle ou par rendez-vous permet de synchroniser les opérations de plusieurs processus qui se retrouvent au niveau d'une transition donnée. La figure 5-3 montre un exemple de deux processus. Le franchissement de la transition T7 ne peut se faire que si la

place P12 du processus1 et la place P6 du processus 2 contiennent chacune au moins un jeton. Si ce n'est pas le cas, par exemple la place P12 ne contient pas de jetons, le processus 2 est bloqué sur la place P6 ; il attend que l'évolution du processus 1 soit telle qu'au moins un jeton apparaisse dans la place P12.

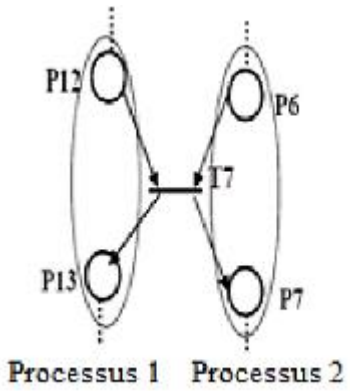


Figure 5-3 : Synchronisation par rendez-vous[86].

5.3.1.2 Synchronisation par signal (sémaphore)

Dans ce type de synchronisation, l'évolution d'un processus est conditionnée par celle d'un autre. Dans l'exemple de la figure 5-4, les opérations du processus 2 ne peuvent se poursuivre que si le processus 1 a atteint un certain niveau dans la suite de ses opérations. Par contre, L'avancement des opérations du processus 1 ne dépend pas de l'avancement des opérations du processus 2. Si la place "Signal" est marquée et la place "Attente" ne l'est pas, cela signifie que le processus 1 a envoyé le signal mais le processus 2 ne l'a pas encore reçu. Si, par contre, la place "Signal" n'est pas marquée et que la place "Attente" est marquée, cela signifie que le processus 2 est en attente du signal.

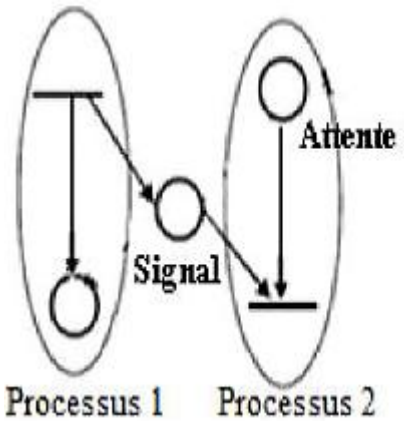


Figure 5-4: Synchronisation par signal[86].

5.3.2 Parallélisme

Le parallélisme représente la possibilité que plusieurs processus évoluent simultanément au sein du même système. Une transition ayant plusieurs places en sortie, peut provoquer le

départ simultané de l'évolution de plusieurs processus. Par exemple sur la figure 5-5, le franchissement de la transition T1 met un jeton dans la place P2 (ce qui marque le déclenchement du processus 1) et un jeton dans la place P3 (ce qui marque le déclenchement du processus 2).

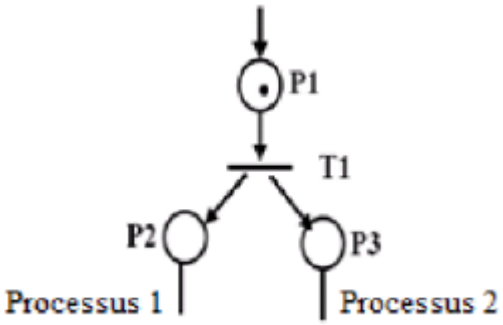


Figure 5-5 : Parallélisme[86].

5.3.3 Partage de ressources

C'est un type de modélisation lié d'un système au sein duquel plusieurs processus partagent une même ressource en utilisant le principe de l'exclusion mutuelle. Sur la figure 5-6, le jeton dans la place P0 présente une ressource mise en commun entre le processus 1 et le processus 2. Le franchissement de la transition T17, lors de l'évolution du processus 1 entraîne la consommation du jeton présenté dans la place P0. La ressource que constitue ce jeton n'est alors plus disponible pour l´évolution du processus 2. Lorsque la transition T18 est franchie, un jeton est alors placé dans la place P0 : la ressource devient alors disponible pour l'évolution des deux processus.

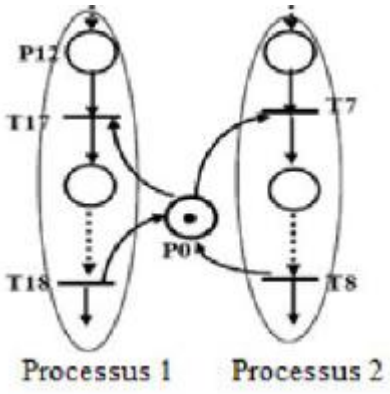


Figure 5-6: Partage de ressource[86].

5.4 Principales extensions des réseaux de Petri

Au départ, il s'agissait essentiellement de représenter les systèmes à événements discrets. Seuls les systèmes qui changeaient de manière discrète pouvaient être représentés dans le

cadre des réseaux de Petri. Puis, beaucoup d'extensions ont vu le jour pour que d'autres aspects soient prises en compte, tels que ; la modularité, la colorisation, la temporisation, la mobilité et autres. Dans ce qui suit nous explorons les extensions les plus répandues.

5.4.1 Réseaux de Petri colorés

Les réseaux de Petri colorés[87] qu'on abrège par RdPC, sont des réseaux de Petri dans lesquels les jetons portent des couleurs, une couleur est une information attachée à un jeton. Cette information permet de distinguer des jetons entre eux et peut être de type quelconque.

Dans la terminologie des RdPCs, les types associés aux places et aux transitions sont appelés « domaines de couleur », et les valeurs des jetons sont associées aux couleurs des jetons. Un marquage d'un réseau de Petri coloré associe à toute place du réseau un multi-ensemble de jetons typés par le domaine de couleurs de la place correspondante.

Les arcs joignant les places aux transitions sont étiquetés par des constantes, des variables ou des fonctions. Les valeurs affectées à ces derniers déterminent les jetons typés retirés ou ajoutés à une place pour le franchissement d'une transition.

Enfin, les transitions du réseau peuvent être gardées (on leur associe des expressions booléennes). Cette dernière ne sera franchissable que si son expression de garde est vraie (voir figure 5-7).

En conclusion, Les réseaux de Petri colorés n'apportent pas de puissance de description supplémentaire par rapport aux réseaux de Petri ordinaires, ils permettent juste une condensation de l'information.

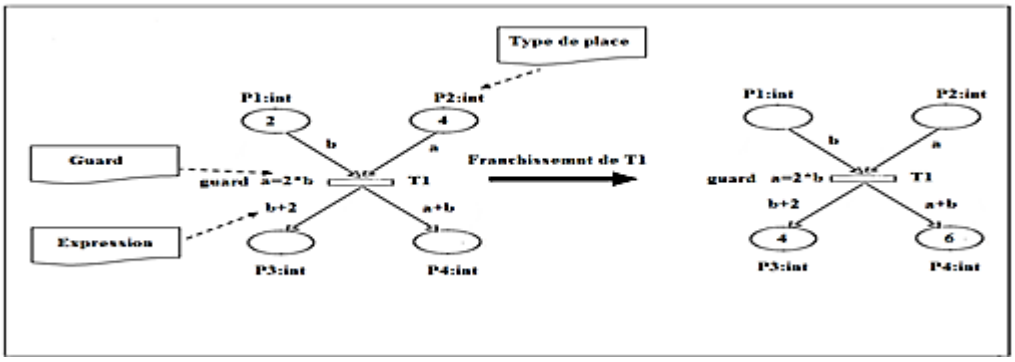


Figure 5-7 : Exemple d'un RdPC.

5.4.2 Réseaux de Petri temporels

Ce type de modèle[88], expriment nativement des spécifications «en délais». En explicitant les débuts et fins d'actions, ils peuvent aussi exprimer des spécifications «en

durées». Les réseaux de Petri temporels s'obtiennent en associant deux dates *min* et *max* à chaque transition d'un réseau de Petri classique.

5.4.3 Réseaux de Petri temporisés

L'utilisation des réseaux de Petri temporisé[89] est préconisée dans la modélisation des problèmes d'ordonnancement d'un système dynamique où lors de l'évaluation des performances. Dans ce type de réseau, la durée d'une activité est explicitement intégrée. La temporisation peut concerner les places (réseaux de Petri P-temporisés) ou bien les transitions (réseaux de Petri T-temporisés) selon les événements modélisés.

5.4.4 Paradigme des réseaux dans les réseaux

Les réseaux de Petri classique où étendus tel que les réseaux de Petri colorés est désormais insuffisante pour modéliser les applications mobiles, mais récemment Köhler et al.[90], ont proposé un modèle qui capture la mobilité d'une manière élégante et intuitive. Un tel modèle est fondé sur un formalisme baptisé réseaux dans les réseaux (Nets within Nets en anglais)[91, 92] dont la sémantique est formelle.

L'idée astucieuse d'un tel formalisme est qu'il donne aux jetons mêmes du réseau la structure d'un réseau de Petri.

Le paradigme des réseaux dans les réseaux permet de modéliser les problèmes de grande taille en les structurant selon deux niveaux d'abstraction permettant ainsi la modélisation modulaire (notion d'hiérarchie). Ces deux niveaux d'abstraction sont les suivants :

- Un niveau supérieur appelé *réseau système* où les jetons dans ces réseaux sont eux même des réseaux de Petri (le réseau composé des places P1et P2 et les transitions T1 et T2 dans la figure 5-8).
- Un niveau inférieur appelé *réseau objet.* A ce niveau, le réseau est considéré comme un jeton dans un réseau système, les places d'un réseau objet contiennent des jetons simples (la place P1 de la figure 5-8contient un réseau objet).

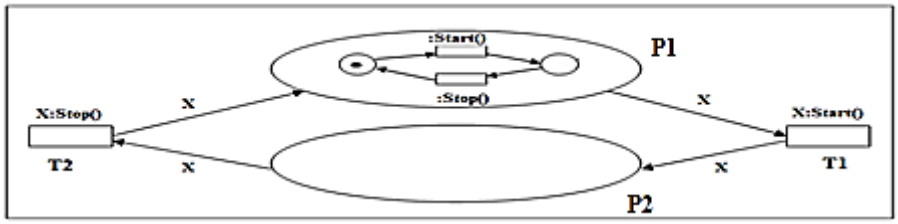


Figure 5-8 : Exemple d'un réseau de Petri nets-within-nets.

Le franchissement d'une transition dans un réseau système se fait par le déplacement de tout un réseau objet d'une place à une autre. Une transition dans un réseau objet peut être synchronisée avec une transition d'un réseau système. Si l'on considère l'exemple de la figure 5-8, on remarque que le réseau objet contenant dans la place P1 est lié à l'arc avec l'inscription *X* ce qui rend la transition T1 active. Cette transition est inscrite avec un canal synchrone (X : Start()) , cela signifie que la transition T1 ne peut être franchissable que si la transition du réseau objet marqué avec l'inscription (: Start()) est franchissable. Cette pré condition permet de faire la liaison entre le réseau objet et la transition T1. Après franchissement de la transition T1 (la synchronisation fait que la transition dans le réseau objet est franchie en même temps que la transition T1) on obtient le réseau présenté dans la figure 5-9. On remarque que la transition du réseau objet avec l'inscription (: Stop()) est synchronisée avec la transition T2 du réseau système, et le franchissement de celle-ci nous fait revenir à la situation du réseau de la figure 5-8.

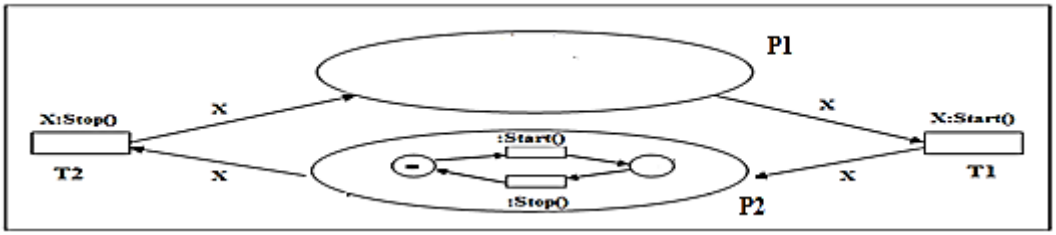


Figure 5-9 : Réseau de Petri Nets-within-nets après franchissement.

L'interaction entre les *réseaux objets* et les *réseaux systèmes* induit quatre possibilités pour contrôler le type de déplacement et par conséquent la mobilité des *réseaux objets.* Voir tableau 5-1.

Réseau Objet	Réseau système	Types de mobilité	Commentaires
Non impliqué	Non impliqué	Mouvement spontané	Aucune influence du réseau objet et du réseau système sur le mouvement. Aucune pré ou post condition n'est nécessaire.
Non impliqué	impliqué	Mouvement Objectif (transportation)	Le réseau système contrôle le mouvement, le réseau objet est transporté d'une position à une autre.
Impliqué	Non impliqué	Mouvement Subjectif	Difficile à imaginer dans la pratique, mais théoriquement un réseau objet peut contrôler le mouvement.
Impliqué	impliqué	Mouvement Consensuel	Les deux réseaux ont une influence sur le mouvement

Tableau 5-1 : Types de mobilité[32].

Ceci dit, nous allons donner une brève introduction d'une implémentation de certains aspects des réseaux dans les réseaux appelés les réseaux de référence

5.4.4.1 Réseaux de référence

Les réseaux de référence[93] sont une notation graphique très appropriée à la description et à l'exécution des processus concurrents et complexes. Telle que la majorité des autres formalismes Petri, il existe un outil de simulation de ces réseaux appelé **Renew** (***Re****ference* ***ne****t* ***w****orkshop*)[82]. Les réseaux de référence étendent les réseaux de Petri classiques et les réseaux de Petri colorés par l'introduction de nouvelles notions telles que : les instances réseaux, les réseaux considérés comme jetons objets, la communication via des canaux synchrones et l'utilisation de différents types d'arcs. Les définitions de ces extensions sont données dans ce qui suit[94, 95] :

***Les instances de réseaux* :** Selon le même principe d'instanciation d'un objet à partir d'une classe dans les langages de programmation objet, les instances de réseaux sont une copie instanciée à partir d'un réseau gabarit (un réseau moule). Notons que différentes instances d'un même réseau peuvent prendre des états différents à un instant donné et sont indépendantes les unes des autres dans tous leurs aspects.

***Le réseau est considéré comme jeton objet* :** les réseaux de référence implémentent le paradigme des réseaux dans les réseaux. Les places dans ces réseaux (appelés aussi réseaux systèmes) peuvent contenir des jetons schématisant un autre réseau (également appelé réseau objet). Partant de ce principe, on obtient facilement une hiérarchie de réseaux : un réseau système contenant un jeton schématisant un réseau objet peut lui-même être un jeton schématisant un réseau objet dans un autre réseau système.

Les canaux synchrones : les canaux synchrones permettent de synchroniser deux transitions pour les tirer automatiquement en même temps. Pour que cette synchronisation ait lieu, les deux transitions doivent porter le même nom de canal et le même nombre de paramètres. Notons qu'il est possible de synchroniser plus de deux transitions à un instant donné en les inscrivant, tout simplement, dans plusieurs canaux synchrones.

***Les types d'arc* :** En plus des arcs usuels, les réseaux de référence offrent deux autres types d'arcs : les arcs de réservation et les arcs test. Ces deux types d'arcs sont similaires dans le sens où ils ne changent pas le marquage associé à une place. Les arcs de réservation sont en réalité une autre manière de représenter deux arcs classiques dont les directions sont

opposées. Ils permettent de réserver un jeton lors du tirage d'une transition. Les arcs test sont dénués d'orientation et permettent d'accéder (tester) à un jeton donné.

5.4.4.2 L'architecture MULAN (multi agent net)

L'architecture MULAN[96], est une implémentation du paradigme des réseaux dans les réseaux. Mulan structure le système selon quatre niveaux d'abstraction (voir figure 5-10). Le premier niveau (le réseau en haut à gauche) décrit la structure d'un système d'agents où chaque place représente une plateforme qui peut contenir elle-même l'ensemble des agents se trouvant sur cette plateforme. Chaque plateforme offre aux agents un certain nombre de services. Comme on peut le remarquer après avoir effectué un zoom sur le réseau en haut à droite, la transition (new) sert à créer un agent, la transition (destroy) sert à le détruire et la transition (external communication) relie, par message, deux agents qui se trouvent sur deux plateformes distinctes. L'agent peut accéder à la plateforme par la transition (receive agent), et la quitter via la transition (send agent). Un agent est intelligent puisqu'il a accès à la place dite ''knowledge base'' qui décrit les connaissances de l'agent sur son environnement. Le comportement de l'agent est décrit en termes de protocoles qui se trouvent sous forme de modèles dans la place ''protocols''. Une partie de ces protocoles est instanciée et ensuite placée dans la place ''conversation''. La principale activité chez un tel agent est la sélection d'un protocole et par conséquent le commencement d'une conversation. L'arrivée d'un message et le commencement d'une nouvelle conversation sont tous deux influencés par la place ''knowledge base''.

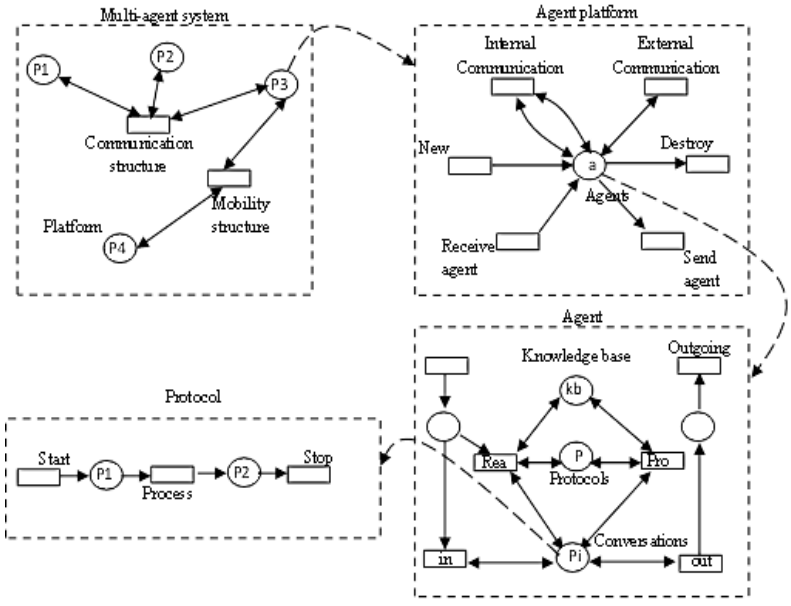


Figure 5-10 : Architecture MULAN[90].

5.5 Outils d'analyse et de simulation des réseaux de Petri

L'aspect formel des réseaux de Petri a motivé les développeurs à mettre au point plusieurs outils permettant la modélisation, la simulation et la vérification des réseaux de Petri (CPNTools[16], JARP[17], MARIA[18], LOLA[19], PROD[20], RENEW[82], etc.). La plupart de ces outils présente un environnement graphique d'édition des réseaux de Petri avec la possibilité de simuler le modèle et d'analyser des propriétés génériques des réseaux de Petri.

5.6 Les réseaux de Petri et les systèmes ambiants

Les réseaux de Petri et leurs extensions présentent un outil de modélisation graphique aisé pour l'expression et la compréhension des systèmes complexes, adaptatifs et mobiles qui est le cas des systèmes ambiants. Par ailleurs, Ils représentent un cadre formel de spécification des systèmes grâce à leur sémantique. De plus, Ils permettent d'exprimer très simplement les concepts premiers des fonctionnements communicants, en incluant les phénomènes d'attente et de synchronisation, et en prenant en considération leurs caractéristiques et paramètres temporels et stochastiques. Ainsi leur principal attrait par rapport à la grande majorité des autres représentations est leur aptitude à donner des informations concernant l'exécution du système.

Dans ce cadre, il semble très logique que les réseaux de Petri constituent un bon formalisme pour modéliser les systèmes ambiants.

5.7 Conclusion

Nous avons présenté dans ce chapitre le formalisme RdP qui est par leur représentation graphique compacte, constitue un modèle facile à comprendre, à manipuler et donc à analyser.

Nous avons abordé dans un cadre général la modélisation par les réseaux de Petri. Par la suite, nous avons étudié quelques types de RdP avec leurs variantes (Extensions). On a constaté que les propriétés mathématiques des réseaux de Petri et leur aspect formel ont donné naissance à une multitude d'outils de simulation et de vérification.

---

[16] http://wiki.daimi.au.dk/cpntools/cpntools.wiki.

[17] http://sourceforge.net/jarp.

[18] http://www.tcs.hut.fi/maria.

[19] http://www.informatik.hu-berlin.de/~kschmidt/lola.html.

[20] J.F. Pradat-Peyre, « Une introduction à Prod », 20 Juin 2003, http://l1.lamsade.dauphine.fr/~moreaux/outilsréseau de Petri200603/prodmaria.pdf

Chapitre 5 Les réseaux de Petri

Nous porterons dans ce qui suit une attention particulière au réseau de Petri étendu : les réseaux de référence pour modéliser, simuler et même tester les systèmes ambiants à base d'agents en faisant appel à l'architecture MULAN utilisée pour décrire l'aspect hiérarchique des systèmes multi-agents.

Chapitre 6

La modélisation et la validation d'un système ambiant

6 La modélisation et la validation d'un système ambiant

6.1 Introduction

Comme on s'intéresse aux solutions de test basé sur les modèles dans les systèmes ambiants, notre première contribution consiste à proposer une approche de modélisation d'environnement ambiant à base d'agents. Cette dernière va être expliquée en détail durant ce chapitre.

6.2 Présentation de l'approche

L'intelligence ambiante est un nouveau domaine de recherche qui assiste l'humain dans son quotidien ; c'est une application importante des systèmes multi-agents (SMA). L'un de ses domaines privilégiés est la domotique. Dernièrement, un phénomène social qui a motivé la recherche dans la domotique est le vieillissement rapide de la population.

En l'absence de statistiques locales, nous nous sommes basés sur des statistiques fournies par le très connu US Census Bureau. La figure 6-1 illustre ces statistiques. On y remarque que le nombre de personnes âgés et à mobilités réduites est en constante augmentation. Entre 2010 et 2050 en particulier, les personnes âgées de plus de 65 ans et celles âgées de plus de 85 ans atteindront les chiffres de 80 millions et de 20 millions respectivement.

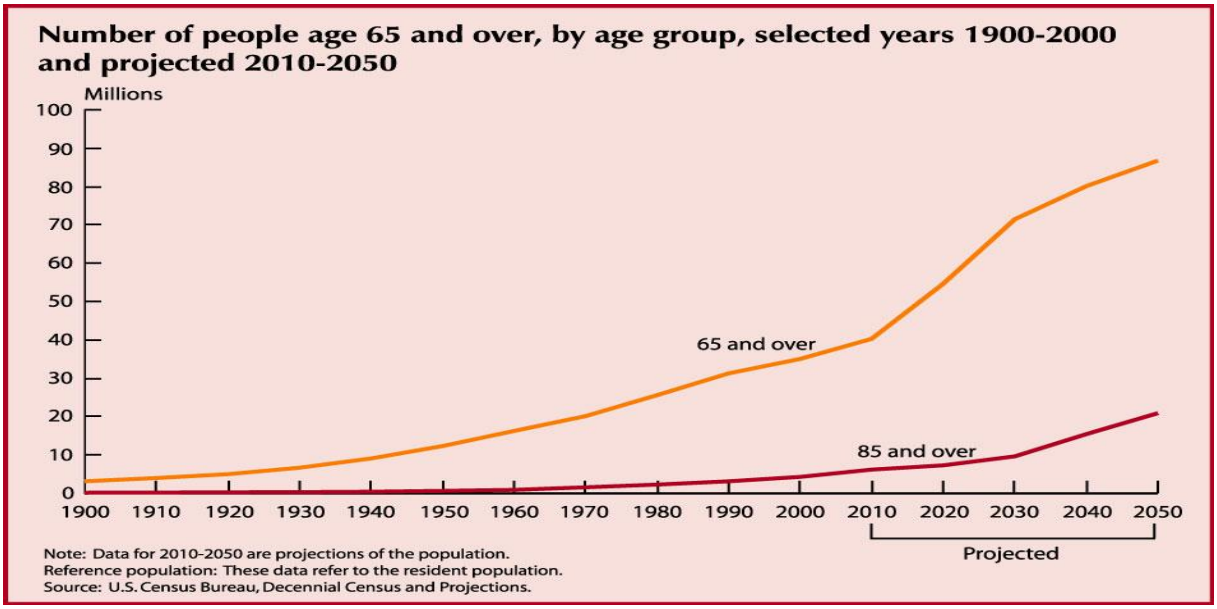


Figure 6-1: Tendances démographiques américaines.

Et ce n'est pas tout !

- Un citoyen sur 5 (environ 18,5%de la population américaine) sera âgé (vieux) en 2030.

- Entre 1990 et 2020, la population dont l'âge varie entre 65 et 74 ans augmentera de 74 %. Celle âgée de moins de 65 ans accroîtra de 24 % seulement.

- cette croissance devrait se produire entre 2010 et 2030, lorsque la génération jeune ou ce que l'on appelle la génération "baby-boom» atteint la vieillisse. Durant cette période, le nombre de personnes âgées augmentera en moyenne de 2,8 % par an.

- De plus, les personnes âgées de plus de 65 ans souffrent le plus souvent de déficiences motrices, sensorielles ou organiques : maladies cardio-vasculaires, respiratoires, intellectuelles ou mentales.

- Cette population dépense pour les soins médicaux quatre fois de plus que pour la population jeune.

Face à ces contraintes, nous pensons concevoir des maisons intelligentes pour ce type de personnes, contrairement à les placer dans des maisons de retraites (considéré dans notre société comme un acte immoral) devrait

- Contribuer en une meilleure qualité de vie en augmentant la maîtrise de soi, l'estime de soi, et l'épanouissement,
- Permettre à ces personnes de rester indépendant chez soi en rendant leur quotidien aisé.

Il est donc nécessaire:

- D'adapter la maison aux capacités fonctionnelles de l'utilisateur,
- De surveiller leur état de santé pour la prévention et les interventions précoces,
- D'accroître l'efficacité des services de soins via l'utilisation des nouvelles technologies afin de les dispenser à ces personnes dans leur environnement local,
- Et enfin de rendre attrayant leur environnement social.

Cependant et avant de pouvoir bénéficier de cette technologie, plusieurs défis devraient être relevés. Notamment :

Chapitre 6 La modélisation et la validation d'un système ambiant

- La plupart des systèmes sont conçus autour d'architectures fermées rendant très difficile toute modification
- Il faut respecter les problèmes de confidentialité, de sécurité et protection de la vie privée.
- la Création d'un environnement sûr et sécurisé afin de pouvoir réduire les chutes, les handicaps, le stress, la peur ou l'isolement social.
- ainsi il est fortement recommandé pour le développement des maisons intelligentes de commencer par un modèle de haut niveau et d'adopter un processus de raffinement, de simulation, de vérification, de mise en œuvre et de test.

L'approche que nous proposons permet :

- De créer un système modulaire qui rend facile l'ajout/la suppression de composants, de les maintenir et de les mettre à jour.
- et ce en utilisant un modèle formel capable de gérer à la fois la modélisation et la vérification des maisons intelligentes.
- Par ailleurs, Il très clair que la modélisation des systèmes complexes et mobiles telle les systèmes ambiants utilisant les réseaux de Petri ordinaires est insuffisante à cause des particularités des applications mobiles (migration, clonage, etc.). Il en est même pour les réseaux de Petri colorés. L'introduction de nouvelles extensions aux réseaux de Petri est nécessaire. Pour cela notre approche est basée sur l'utilisation du paradigme des réseaux dans les réseaux. Ce formalisme donne aux jetons d'un réseau la structure d'un autre réseau de Petri. Ainsi, contrairement aux autres types de réseaux de Petri où les jetons sont passifs, les jetons dans ce paradigme sont actifs. de plus ce dernier est renommé pour son aisance à capturer différents types de mobilité et par sa justesse à se mouler parfaitement au principe de composition des systèmes multi-agents (architecture MULAN discuté dans le chapitre précédent). Ce paradigme permet de détecter au plus tôt les ambiguïtés ou les contradictions au sein du modèle, avec un bénéfice immédiat pour le test de validation. De plus, en utilisant les réseaux dans les réseaux le nombre d'erreurs est réduit suite à l'utilisation d'un seul modèle formel.

Dans ce qui suit, nous présentons une étude de cas montrant la puissance de ce paradigme dans la modélisation des systèmes ambiant mobiles.

Chapitre 6 La modélisation et la validation d'un système ambiant

6.3 Cas d'étude : Maison intelligente pour personnes âgées[97]

Imaginons une maison constituée d'une cuisine, une salle séjour, un hall, deux chambres et une salle de bain. Etant intelligente signifie qu'une telle maison doit contenir un grand nombre de services qui coopèrent pour simplifier la vie du propriétaire, de faire des économies d'énergie et de fournir des solutions de confort et de sécurité. La figure 6- 2 montre un exemple d'une telle maison où divers appareils électroménagers sont éparpillés dans les chambres pour constituer un environnement collaboratif fournissant des services intelligents et assurant la sécurité des habitants. Pour cela, plusieurs détecteurs y sont installés : Par exemple, des détecteurs du niveau et du taux de température (T_l) et (T_r). Des détecteurs de fumée (S) pour détecter les incendies, un détecteur de gaz (G) pour détecter le risque de fuite de gaz. Il y'a des capteurs d'image (I) et des détecteurs de mouvement (M) dans toutes les chambres afin d'identifier tout visiteur et vérifier s'il est autorisé ou non. Des détecteurs de bris de verre (B) et des détecteurs de contact (C) sont également installés sur toutes les fenêtres/portes pour vérifier leur innocuité. En outre, la maison intelligente fournit des solutions d'économie d'énergie grâce à l'installation de détecteurs de présence (O) dans toutes les chambres. Le système d'éclairage (L) utilisera les conditions environnementales pour aider à économiser plus d'énergie.

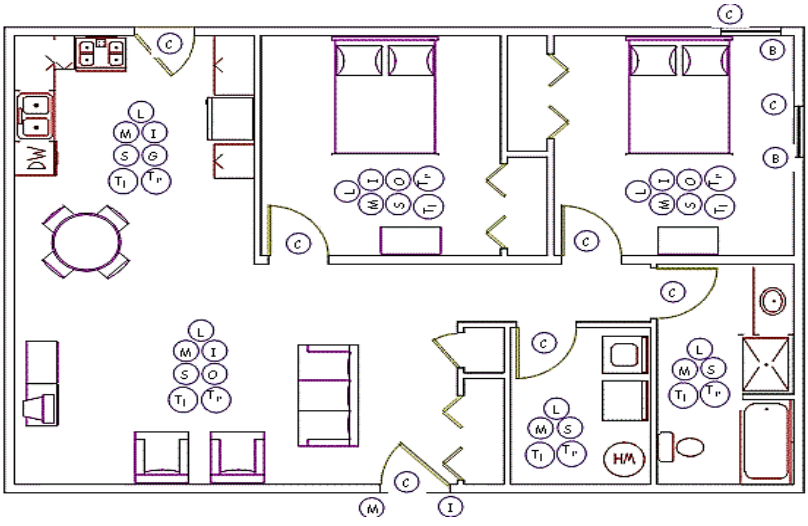


Figure 6-2: exemple d'une maison intelligente.

6.4 Modélisation des composants de la maison intelligente

Ceci dit, notre maison intelligente destinée aux personnes âgées sera conçue selon l'architecture MULAN[96].

6.4.1 Modélisation de l'environnement

Il est souvent indésirable, lors de la modélisation d'un système complexe, de voir toute sa complexité au moment de la modélisation et/ou l'exécution. Par conséquent, la notion de vue est introduite. Plusieurs vues d'un système multi-agent sont possibles. Par exemple, le système d'agents, l'ensemble des plates-formes, l'agent lui-même ou tout simplement son comportement.

Par ailleurs, lorsqu'il s'agit de modéliser et de simuler des systèmes du monde réel, tels les maisons intelligentes, on aura besoin d'ajouter la notion temps à ce formalisme. Dans les réseaux temporisés, une estampille temporelle est jointe à chaque jeton. Elle désigne le moment où le jeton est disponible.

Prenons un exemple de la simulation de deux feux de circulation dans lequel les feux jaunes ont été omis. La notation a@30 signifie que le passage d'un feu à l'autre prendra environ 30 unités de temps. Notons aussi que ce passage est parfaitement synchronisé par l'utilisation des canaux synchrones (notés : go1 et : go2). Dans cet exemple les voitures ont été modélisées par des agents mobiles (new car1 () et new car2 (). Le franchissement du carrefour est autorisé si le feu est vert. C'est ce qui est modélisé par les arcs de test (n'ayant pas de flèches à leurs extrémités) (figure 6- 3).

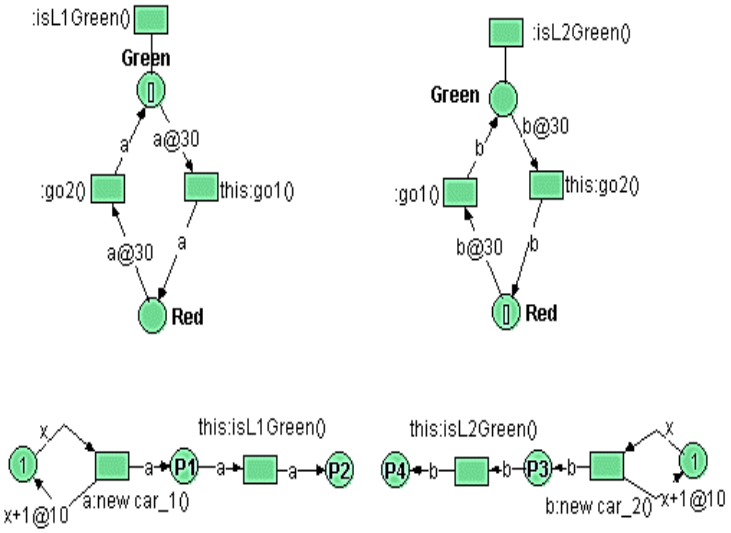


Figure 6-3 : Exemple de feux de circulation.

L'utilisation des réseaux dans les réseaux comme paradigme de modélisation permet d'exploiter ceci comme suit :

L'ensemble du système sera conçu comme un réseau avec les places définissant les chambres : cuisine, séjour, les deux chambres, salle de bain et le hall. Les transitions modélisent les mouvements possibles entre ces pièces sont schématisés par les transitions vertes (figure 6-4). Etant intelligente, les chambres devraient être sensibles à la présence humaine. Être sensible implique de connaître les exigences des utilisateurs, apprendre ou connaître leurs préférences, c'est pourquoi nous avons installé dans chaque chambre un agent cognitif stationnaire qui servira les habitants de cette maisons (transitions bleus). Ces agents sont : agent hall, agent cuisine, agent salle de bain, agent chambre 1, agent chambre 2 et agent séjour. Par ailleurs, parmi tous les dispositifs dispersés dans la maison nous avons sélectionné ceux de la chambre séjour et de la cuisine qui sont modélisés en tant qu'agents : l'agent télévision, l'agent lecteur de music, l'agent rideaux électriques, l'agent lumière, l'agent cuisinière et l'agent détecteur de fuites de gaz et enfin l'agent machine à café (transition marrons). Il existe également un agent particulier (transition rouge) responsable de la vérification de la pression artérielle et / ou le taux de glycémie de la personne âgée à des instants aléatoires de la journée. Il peut prendre la forme d'un bracelet sur le poignet de la personne âgée. Nous terminons par les transitions schématisant les occupants de cette maison (transition jaunes) à savoir le propriétaire (la personne âgée), son médecin traitant et un proche (disons son fils).

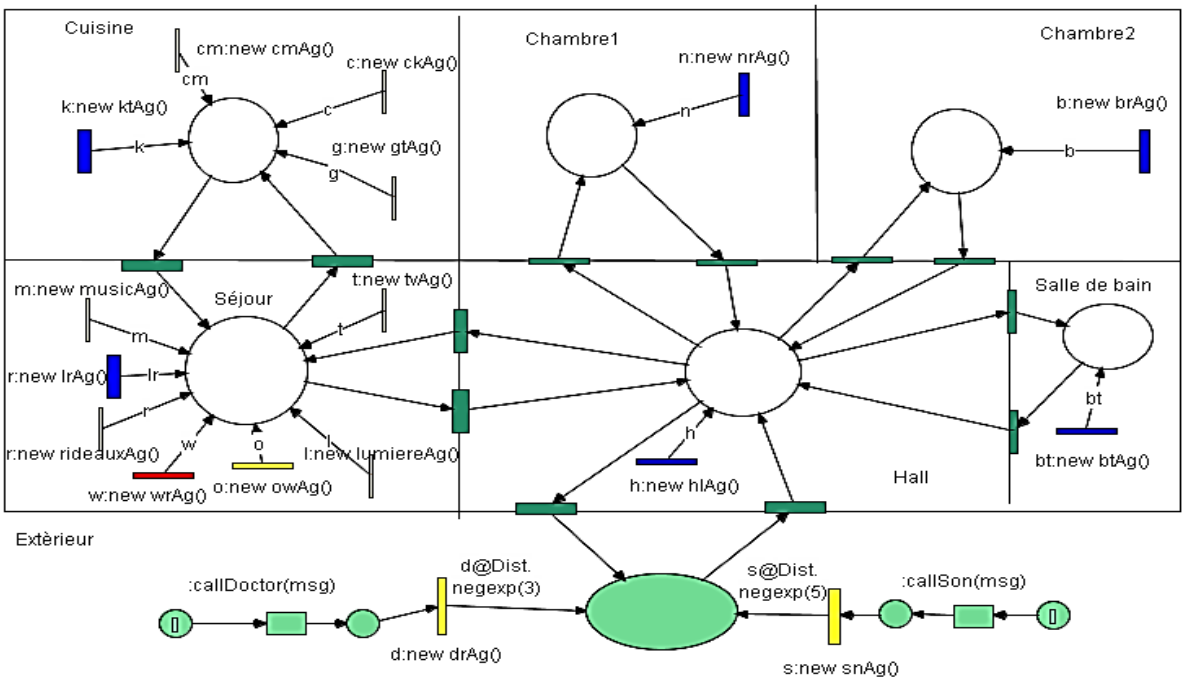


Figure 6-4: Réseau système de la maison intelligente.

6.4.2 Modélisation des agents

Tous nos agents partagent la même structure, représentée dans la figure 6-5, mais avec des bases de connaissances et des protocoles différents (comportement différents).

Nos agents sont encapsulés puisque le seul moyen de communication se fait via l'envoi de message, ils peuvent être réactifs ou proactif (transition réactive et transition proactive) et ils sont intelligents (accès à la place base de connaissance).

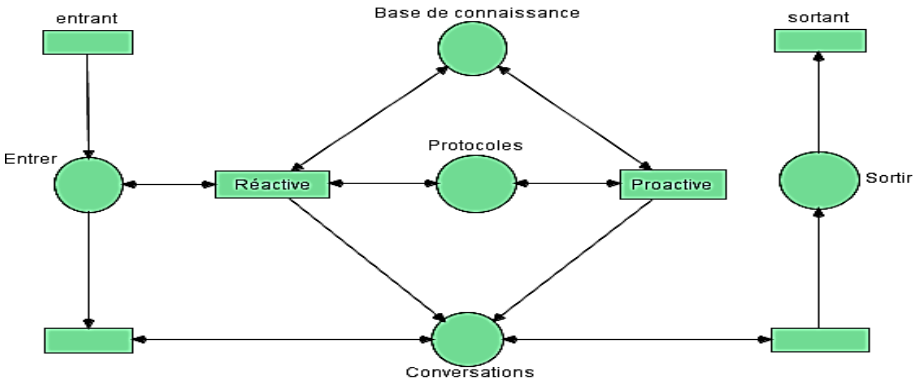


Figure 6-5: Structure d'un agent.

6.4.2.1 Base de connaissance

Dans les situations simples, la base de connaissances pourrait être représentée par un réseau de Petri. Dans d'autres, plus compliquées, on obtient facilement un réseau de taille importante pour lequel il est difficile de maintenir ou d'apporter la moindre modification.

Pour surmonter cette contrainte, nous avons augmenté l'outil de modélisation Renew par une connexion à un système expert, en l'occurrence Java Expert System Shell (JESS)[21] dont l'architecture est donnée figure 6-6.

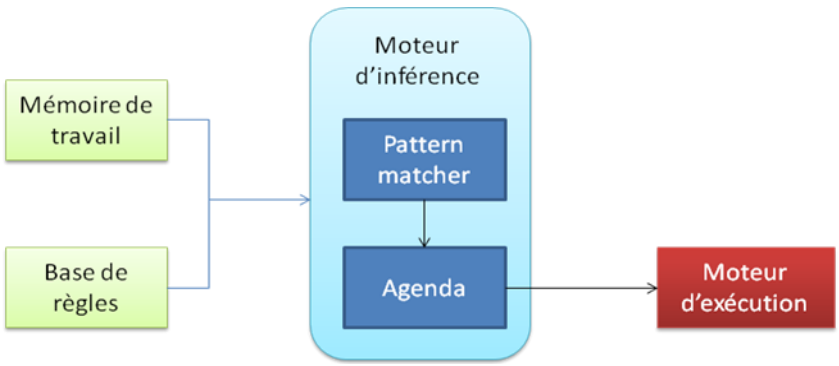


Figure 6-6 : Architecture JESS.

[21] JESS http://herzberg.ca.sandia.gov

Un programme écrit en JESS consiste en un ensemble de règles (base de règles), de faits (mémoire de travail) et un moteur d'inférence qui décide quelles règles doivent être exécutées et quand. Les faits doivent obéir à un modèle (Template). Le modèle a un nom et un ensemble de slots (ou champs).

Il faut noter que chez lui, le comportement du propriétaire est très dépendant des intervalles temps de la journée dans lesquelles il se trouve. Intuitivement ces intervalles sont : « matin », « midi », « après-midi » et « soir ». Ainsi, on peut imaginer différents scénarios : " le mode de réveil ", le mode " sortir", le mode " le retour ", "le mode soir " et "le mode sommeiller ".

Avant de discuter de l'un de ces scénarios, nous commençons par définir la mémoire de travail et les bases de règles de chacun de nos agents. Un intérêt particulier sera accordé à l'agent de la salle de séjour et l'agent de cuisine. En effet, ces agents doivent garder un œil attentif sur la personne âgée résidant dans cette maison. Pour cela, il est nécessaire qu'ils stockent toutes les informations (appelées aussi faits dans la terminologie JESS) sur cette personne, son environnement ainsi que sur les autres agents.

Les modèles suivant définissent le propriétaire de la maison (la personne âgée) ainsi que les personnes qui lui sont proches :

```
(deftemplate proprietaire (slot prenom) (slot sexe) (slot age) (multislot maladies) (multislot preferencesTV) (multislot preferencesBouffe) (multislot preferencesMusic))

(deftemplate proche (slot prenom) (slot sexe) (slot age) (slot type) (slot telephone))
```

Les faits suivants permettent, entres autres, à nos agents de créer l'ambiance souhaitée par le propriétaire une fois entré dans l'une de ces deux pièces (cuisine et salle de séjour) :

```
(deftemplate entrer (slot prenom) (slot chambre) (slot heure))
(deftemplate television (slot etat) (slot chaine))
(deftemplate music (slot etat) (slot genre-music))
 (deftemplate lumiere (slot etat))
(deftemplate rideaux (slot etat))
(deftemplate machine-cafe (slot boisson) (slot sucre))
```

En outre, et étant donné que cette maison est destinée pour une personne âgée, souffrant d'une ou de plusieurs maladies (généralement chroniques), les agents doivent être en mesure

d'analyser les facteurs significatifs en relation avec ces maladies. Pour cela, nos agents utiliseront les faits suivants :

```
(deftemplate pression-arterielle (slot Valmax) (slot Valmin))
(deftemplate glycemie (slot taux) (slot etat))
```

Enfin, il faut noter que dans une maison intelligente, les appareils sont souvent utilisés pour créer un environnement dans lequel de nombreuses caractéristiques sont automatisées. Mais dans certaines situations, l'utilisateur décide d'effectuer lui mêmes des tâches (mode manuel) ou permettre aux appareils de réaliser ses tâches (mode automatique).

Assigner des valeurs à l'ensemble des slots de chaque template permet aux agents d'identifier la ou les personnes ayant accès à cette maison. Un exemple de personnes autorisées est le suivant :

```
(assert (proprietaire (prenom Ben) (sexe male) (age 69) (maladies diabetes hypertension) (preferencesTV news docs sport) (preferencesBouffe the poisson salade) (preferencesMusic orientale  pop classic))
(assert (proche (prenom Tim) (sexe male) (age 45) (type medecin) (telephone 077777777))
(assert (proche (prenom Bob) (sexe male) (age 35) (type fils) (telephone 06666666))
```

Maintenant, nous allons définir quelques règles JESS.

Dans la première, aussitôt que le propriétaire entre dans la salle de séjour, l'agent va vérifier le temps système (le matin dans notre cas) pour, et de manière concurrente, allumer la télévision et la positionner sur une chaine d'information, et ouvrir les rideaux :

```
1 :(defrule rule1
(entrer {prenom == Ben && chambre == sejour && heure >= 8 && heure <= 10})
(proprietaire {preferencesTV == news})
=>          (assert (television (etat ON) (chaine NSN)))
            (assert (rideaux (etat Open)))
```

Dans la seconde, l'agent de cuisine demande à l'agent machine à café de préparer un thé sans sucre :

```
2 :(defrule rule2
(entrer {prenom == Ben && chambre == cuisine && heure > 10 && heure <= 12}) (
proprietaire {preferencesBouffe == the})
=>        (assert (machine-cafe (boisson the) (sucre Non)))
```

Enfin, les règles suivantes permettent à nos agents (salle de séjour et cuisine) de statuer sur l'état de santé du propriétaire en fonction de son taux de glycémie et de prendre les dispositions nécessaires :

```
3 :(defrule rule3
(glycemie {taux < 0.60 })
=> (assert (ACLMessage (communicative-act INFORM) (sender wrAg) (receiver lrAg) (content "hypoglycemie ") (conversation-id " appeler fils "))
```

```
4:(defrule rule4
(glycemie {taux > 1.10 })
=> (assert (ACLMessage (communicative-act INFORM) (sender wrAg) (receiver kiAg) (content " hyperglycemie ") (conversation-id " appeler medecin "))
```

6.4.2.2 Protocoles

Nous arrivons enfin à nos protocoles. En mode "réveil", l'exécution du programme JESS rattaché à la base de connaissance de notre personne âgée (modélisées dans notre cas par un agent mobile), conclura avec un fait valide « prendre le petit déjeuner ». Guidé par ce fait ; l'agent personne âgée sélectionne puis instancie le protocole : « entrée en cuisine »figure 6-7.

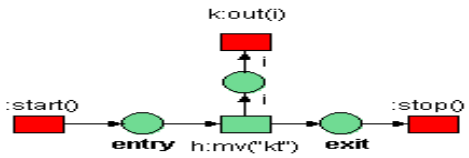


Figure 6-7 : protocole entrer en cuisine.

Sur ce protocole, la transition mvkt () produit une performative i définissant l'identité de la personne accédant à la cuisine qui sera transmise vers l'agent de cuisine via le canal asynchrone k : out(i). Ceci fait, le protocole se termine (en exécutant la transition stop). La transition k : out(i) est satisfaite par synchronisation avec la même transition du réseau de l'agent de cuisine, ce qui induit à l'autorisation de la transition *Rea* de cet agent.

Influencé par l'identité de l'agent entrant en cuisine, et les résultats d'une inférence du programme JESS se trouvant dans sa base de connaissances, l'agent de cuisine va instancier en premier lieu le protocole «bienvenue ». Après l'instanciation de ce protocole (Figure 6-8), et connaissant l'identité et les préférences de la personne entrant en cuisine, un message de bienvenue lui est adressé. Par la suite, l'agent doit vérifier si le mode manuel ou automatique est activé (transition : is manual), si on est en mode manuel, l'agent de cuisine ne fera rien que garder un œil attentif sur la personne âgée.

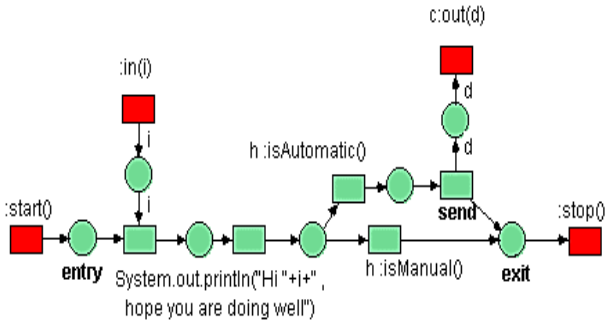


Figure 6-8 : Protocole bienvenue.

Dans le cas contraire, il produit un d performative définissant le type de boisson à préparer et sur le canal c : out (d), ce qui provoque l'instanciation du protocole «demande boisson » (figure 6-9). De la même manière ce protocole se termine en franchissant la transition stop ().

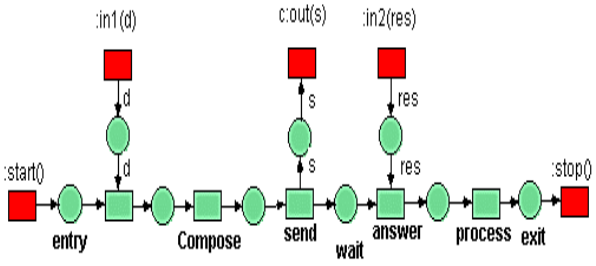


Figure 6-9 : protocole demander boisson.

Après instanciation du protocole « demande boisson » et la réception du type de boisson demandée, la transition « compose » produit un performatif contenant une chaîne qui est dirigée sur le canal c : out (s) à l'agent machine à café ; par la suite le protocole est bloqué en attente d'une réponse. Pendant ce temps le protocole « préparer boisson » s'exécutera (figure 6-10). Deux situations sont possibles : si la réponse reçue est positive, alors l'agent mettra à jour ses croyances ; sinon l'agent de cuisine entreprendra des actions adéquates. Dans les deux cas l'instance de protocole est supprimée (en permettant la transition stop).

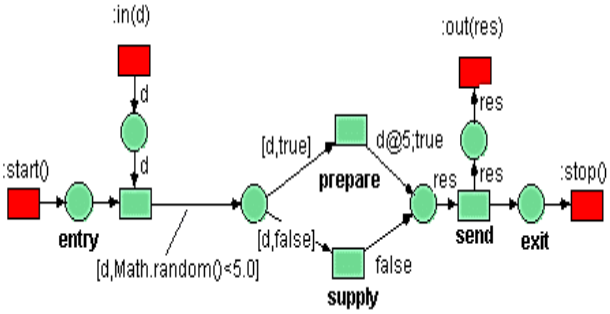


Figure 6-10 : Protocole préparer boisson.

Les protocoles précédents décrivent comment les agents collaborent pour aider les personnes âgées dans le scénario de réveil. Maintenant nous allons voir comment ces agents prennent soin de la santé des personnes âgées. Rappelez-vous de l'agent bracelet qui vérifie le taux de glucose des personnes âgées à un moment aléatoire de la journée. Il est modélisé par un agent mobile (ayant la même structure que les autres agents) qui est transportée par l'agent personne âgée.

Supposons que le mode manuel est activé, l'état de la cuisinière est « on », l'agent bracelet a détecté une hyperglycémie. Dans ce cas précis, l'agent de la chambre où se situe la personne âgée instancie ce protocole (figure 6-11). Ce dernier commence par exécuter les transitions inscrites par les canaux callSon et callDoctor, ce qui provoque la création de ces deux agents.

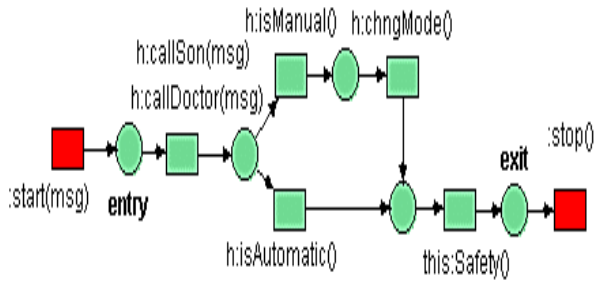


Figure 6-11 : Protocole d'urgence.

Le médecin, par exemple, prend trois heures en moyenne avec une distribution exponentielle négative pour atteindre l'entrée de la maison. Après cela, l'agent vérifie si le mode manuel est activé ou non. Si oui, l'agent passe en mode automatique afin de pouvoir diffuser des messages de sécurité à d'autres agents en exécutant la transition this : safety () (éteindre la cuisinière, TV, l'alarme, etc.).

6.5 Validation du modèle par simulation

Après avoir défini les caractéristiques des composants de la maison intelligente, le système est maintenant prêt pour l'exécution (simulation).

La figure 6-12 montre un état de simulation ordinaire en utilisant l'outil Renew[82]. Seule la salle de séjour et le hall sont représentés. La fenêtre en arrière-plan contient les faits et les règles représentant la base de connaissances de l'agent de la salle de séjour.

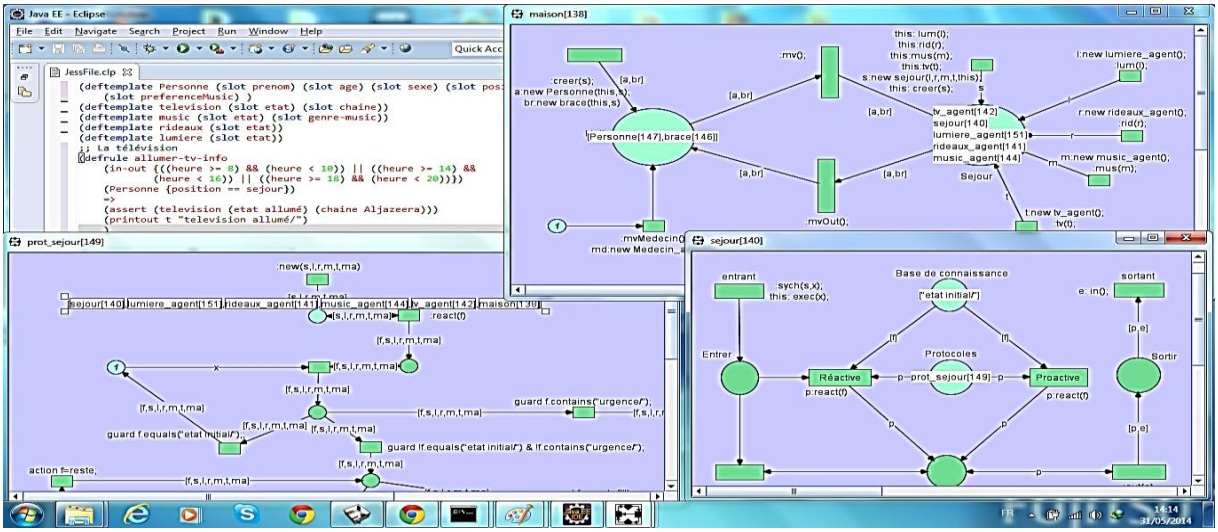


Figure 6-12 : Exemple de Simulation.

La figure suivante (figure 6-13) montre une mise en œuvre de notre approche utilisant la plateforme JADE.

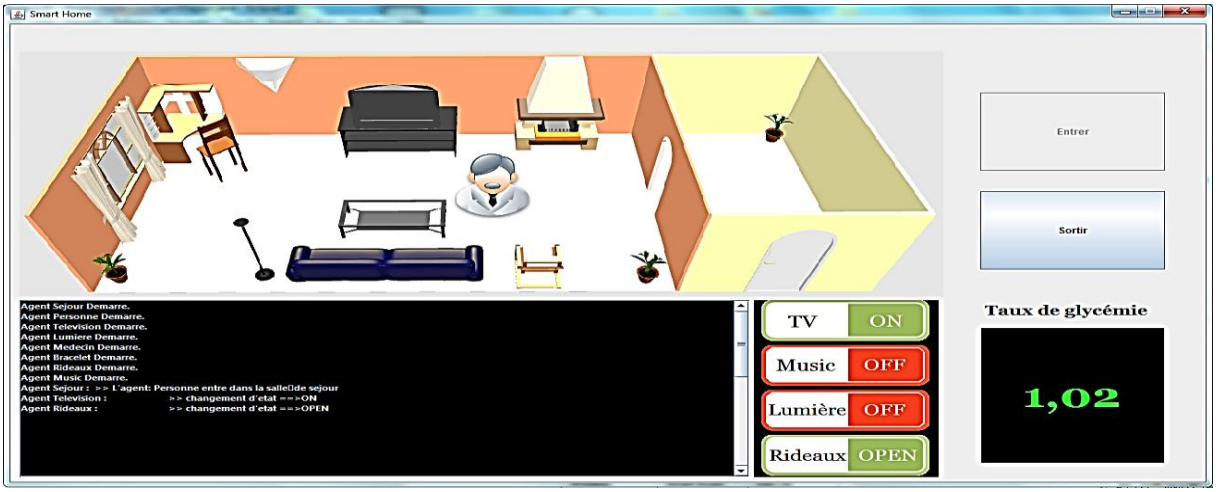


Figure 6-13 : Implémentation de l'approche.

6.6 Conclusion

Pour conclure, nous pouvons dire que contrairement à l'approche que nous proposons, dans laquelle, le processus de modélisation conclut avec un modèle exécutable, un processus de modélisation typique aurait nécessité au moins trois étapes pour égaler un même résultat : (a) modéliser le système, (b) mettre en œuvre le modèle et (c) écrire le programme de visualisation.

Aussi, dans un processus de modélisation typique des erreurs peuvent être introduites lors du passage de la modélisation à l'implémentation, par contre en utilisant les réseaux dans les réseaux le nombre d'erreurs est réduit à cause de l'utilisation d'un seul modèle formel.

Enfin, cette approche rend aisé l'ajout ou la suppression de nouveaux agents, dispositifs ou même faits et règles d'inférence.

Tout ceci aura des retombées bénéfiques telles :

- Le prototypage précoce,
- La vérification/validation,
- Et même supporté la phase de test.

Chapitre 7

L'approche de test

7 L'approche de test

7.1 Introduction

Ayant modélisé notre système ambiant, nous sommes prêt pour le test. Le chapitre suivant détaille notre approche de test basé sur le paradigme des réseaux de référence.

7.2 Approche de test basé sur les réseaux de référence[98]

Un système multi agents peut être aperçu par un testeur selon cinq différents niveaux d'abstraction[99, 100] : le niveau algorithmique, le niveau classe, le niveau agent, le niveau société d'agents et le niveau système. Les techniques de test classiques (fonctionnelles, structurelles ou de non régression) ont été pleinement appliquées au premier niveau d'abstraction (le niveau algorithmique). Par ailleurs, un outil comme JUnit a été approuvé à un niveau pédagogique pour les tests unitaires des classes Java. Son efficacité a poussé beaucoup de chercheurs à proposer des extensions capables de supporter d'autres types d'application : DBUnit (pour les bases de données), NModel (pour les applications écrites en C#). Pour le test basé modèle, une extension de JUnit, appelée ModelJUnit, a été proposée par Mark Utting. La figure 7-1 montre la portée de notre approche.

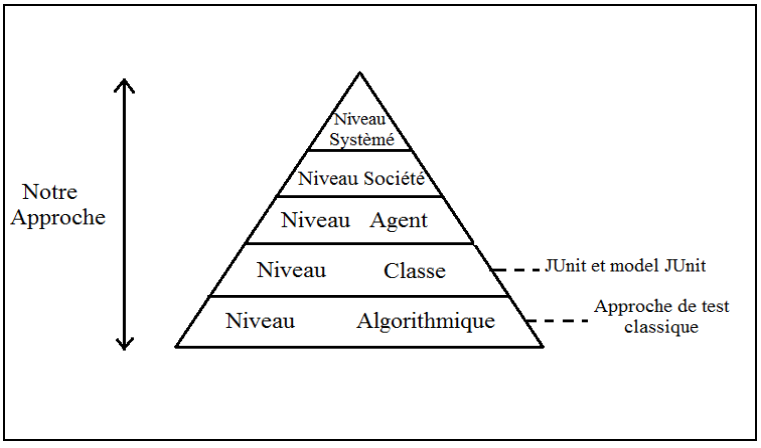


Figure 7-1 : Portée de l'approche de test.

En fait, l'approche que nous proposons s'inspire du ModelJUnit mais elle se différencie par le fait qu'elle est spécifique aux applications à base d'agents dans le sens où les niveaux d'abstraction supérieurs sont aussi couverts (agent, société et système). De plus, alors

que ModelJUnit utilise les machines d'états finis comme modèle, notre approche est basée sur le paradigme des réseaux dans les réseaux. Enfin, dans ModelJUnit, le testeur devra écrire les cas de tests à exécuter et ce, en faisant appel à JUnit, dans notre approche les cas de tests sont générés automatiquement à partir du modèle et leur exécution s'effectue dynamiquement en même temps que l'exécution du système sous test. La figure7-2 récapitule l'architecture générale de l'approche de test proposée.

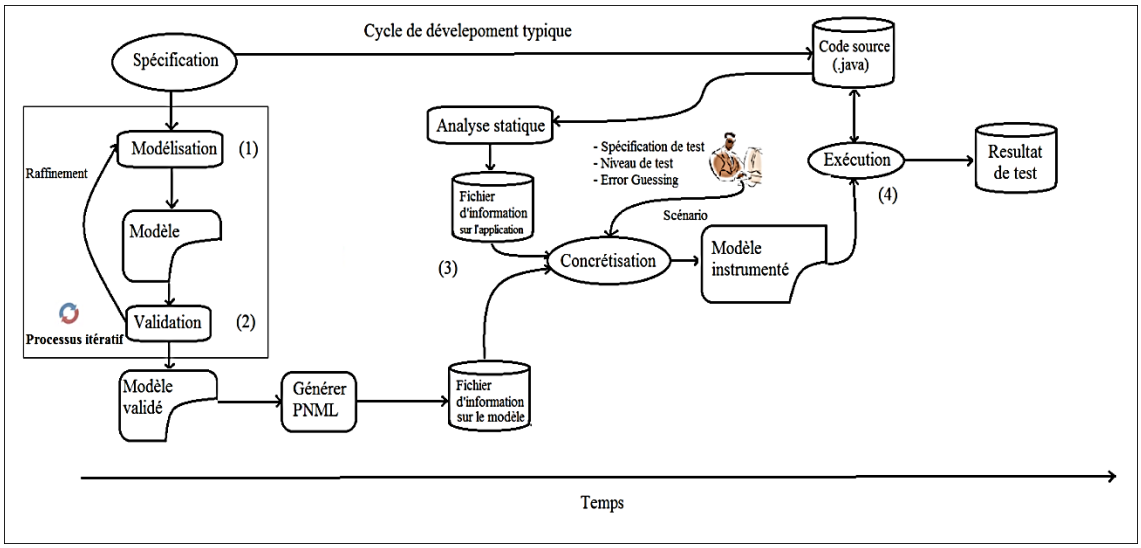


Figure 7-2 : Architecture proposée.

Ainsi, sur la figure 7-2 ci-dessus, l'étape (1) concerne la phase de modélisation. C'est la première étape dans la technique du test basée sur les modèles. Cette phase s'appuie sur la construction d'un modèle abstrait du système sous test (SUT). Ce modèle représente le comportement attendu du SUT. Cependant pour vérifier la validité de ce modèle l'étape (2) de validation est obligatoire. L'ingénieur de test doit simuler le modèle pour chacun des scenarios d'exécution possibles et il peut modifier ou raffiner le modèle en fonction des résultats obtenus jusqu'à ce qu'il soit satisfait et que son modèle réponde bien aux spécifications initiales. Les erreurs non détectées dans la phase de validation seront surement détectées lors de la phase d'exécution du cas de test.

La phase (3) représente la phase de concrétisation. C'est l'étape la plus délicate et constitue notre principale contribution. Elle ne peut être menée sans le code source du système sous test et est dirigée sous la responsabilité du testeur. Elle prend comme entrée un fichier d'information issu du système sous test. Un autre issu du modèle de test (toutes les méthodes présentes dans le code source et toutes les transitions présentes sur le fichier PNML extraites du modèle) et le scénario que le testeur souhaite vérifier. Ce scénario de test se construit en

fonction de la spécification du test, le niveau du test souhaité (agent, société ou système) et en adoptant une technique de conception de cas de test appelée « Error-guessing ».
Une telle technique est basée sur l'expérience du testeur et sur sa faculté d'anticiper sur les endroits (ou les coins) où les fautes pourraient se manifester, et concevoir en fonction de cette connaissance, des tests spécifiques. L'instrumentation génère en sortie un nouveau modèle de test instrumenté. Contrairement aux différentes approches de test, où l'instrumentation porte, la plupart du temps, sur le code source de l'application, notre approche laisse intact l'application sous test permettant ainsi de préserver son comportement initial (l'aspect fonctionnel ou non-fonctionnel ne sera pas modifié).
La dernière étape est la phase d'exécution (4). Lors de cette étape le modèle instrumenté est exécuté déclenchant, par conséquent, les parties du code source correspondantes. Le résultat obtenu est comparé avec celui attendu décrit dans le modèle de test instrumenté. Tout écart provoque l'arrêt du processus de test, sinon d'autres scénarios sont générés et exécutés selon le même principe.

Pour expliquer les différentes étapes de notre approche et pour une meilleure compréhension, nous avons jugé nécessaire de l'introduire via un cas d'étude.

7.3 Cas d'étude

Le modèle producteur- consommateur bien que simple, a été utilisé de manière «abusive » dans la littérature informatique pour résoudre des problèmes de synchronisation, de coordination et de communication. La figure 7-3 décrit notre cas d'étude à un niveau très abstrait.

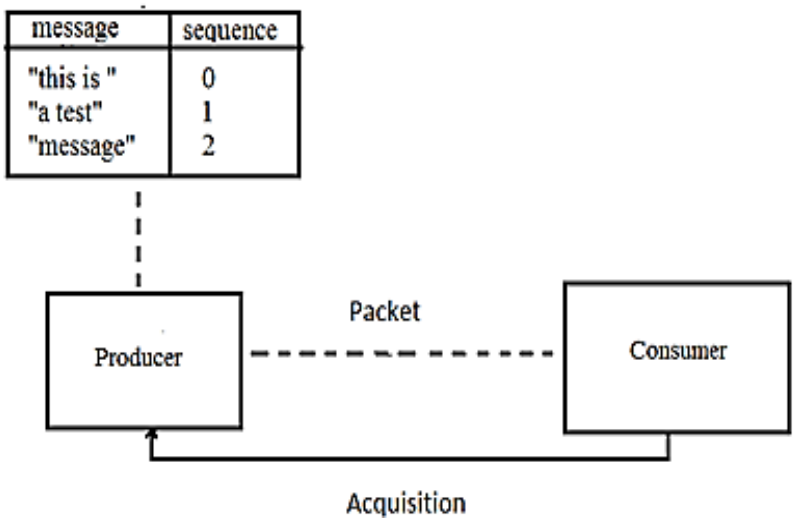


Figure 7-3 : l'exemple producteur-consommateur.

L'exemple est constitué d'un producteur, qui transmet des données à un consommateur. Il est impératif de préciser que cet exemple fonctionne selon les hypothèses suivantes :

- Le consommateur doit rassembler des données originales. Désassemblées par le producteur avant transmission,
- Le principe de transmission se fait selon le protocole « stop-and-wait ». C'est-à-dire que le producteur transmet un paquet de données à la fois et doit attendre un accusé de réception.

Nous allons, dans ce qui suit, détailler précisément les étapes 1 à 4 de l'approche proposée.

7.3.1 Modélisation

Nous pensons que la première étape (construction du modèle) a été pleinement discutée dans le chapitre 6. Néanmoins, nous présentons les modèles abstraits de chaque niveau du système sous test (figure 7-4). Le niveau système est représenté par (a). (b) représente le niveau agent et (c) le niveau protocole.

Au niveau système, le réseau représenté crée deux instances : une pour l'agent producteur et l'autre pour l'agent consommateur. Un message : «*Ceci est un message de test* » est passé comme paramètre à l'agent producteur qui devra le fragmenter et le transmettre au consommateur.

Au niveau agent, après fragmentation du message, l'agent producteur envoie les fragments vers le consommateur avec la méthode «send ()» et attend un accusé de réception du consommateur pour envoyer le deuxième fragment. De l'autre côté l'agent consommateur quand il reçoit le message il émet un accusé de réception vers le producteur avec la méthode «reply ()». L'opération ce répète jusqu'à la fin du message.

Avant tout envoi et réception, la préparation des messages se fait au niveau protocole. Chaque protocole est un réseau dans un autre réseau (la place protocole du réseau agent).

7.3.2 Validation du modèle

Étant donné que ce modèle de test formel abstrait est dérivé manuellement de la spécification des exigences du système, la validation du modèle de test par rapport à la spécification des exigences du système, il doit être effectué en premier afin de détecter les erreurs majeures dans le modèle de test. Avec l'approche MBT, si certaines erreurs restent

dans le modèle, elles sont très susceptibles d'être détectées lorsque les cas de test générés sont exécutés sur le système sous test.

Renew[82] repose entièrement sur la simulation pour explorer les propriétés d'un réseau, où le testeur peut explorer dynamiquement et interactivement l'état de la simulation. En effet, à ce stade, le testeur peut imaginer différents scénarios d'exécution en agissant, par exemple, sur le nombre d'agents ou le nombre d'objets (ressources). En d'autres termes, en simulant le réseau système, le testeur a la possibilité de corriger et/ou raffiner progressivement le modèle abstrait de test. Un tel outil robuste et facile à utiliser réduit considérablement l'effort initial important en termes d'heures-personnes nécessaires principalement dans la construction et la validation du modèle de test.

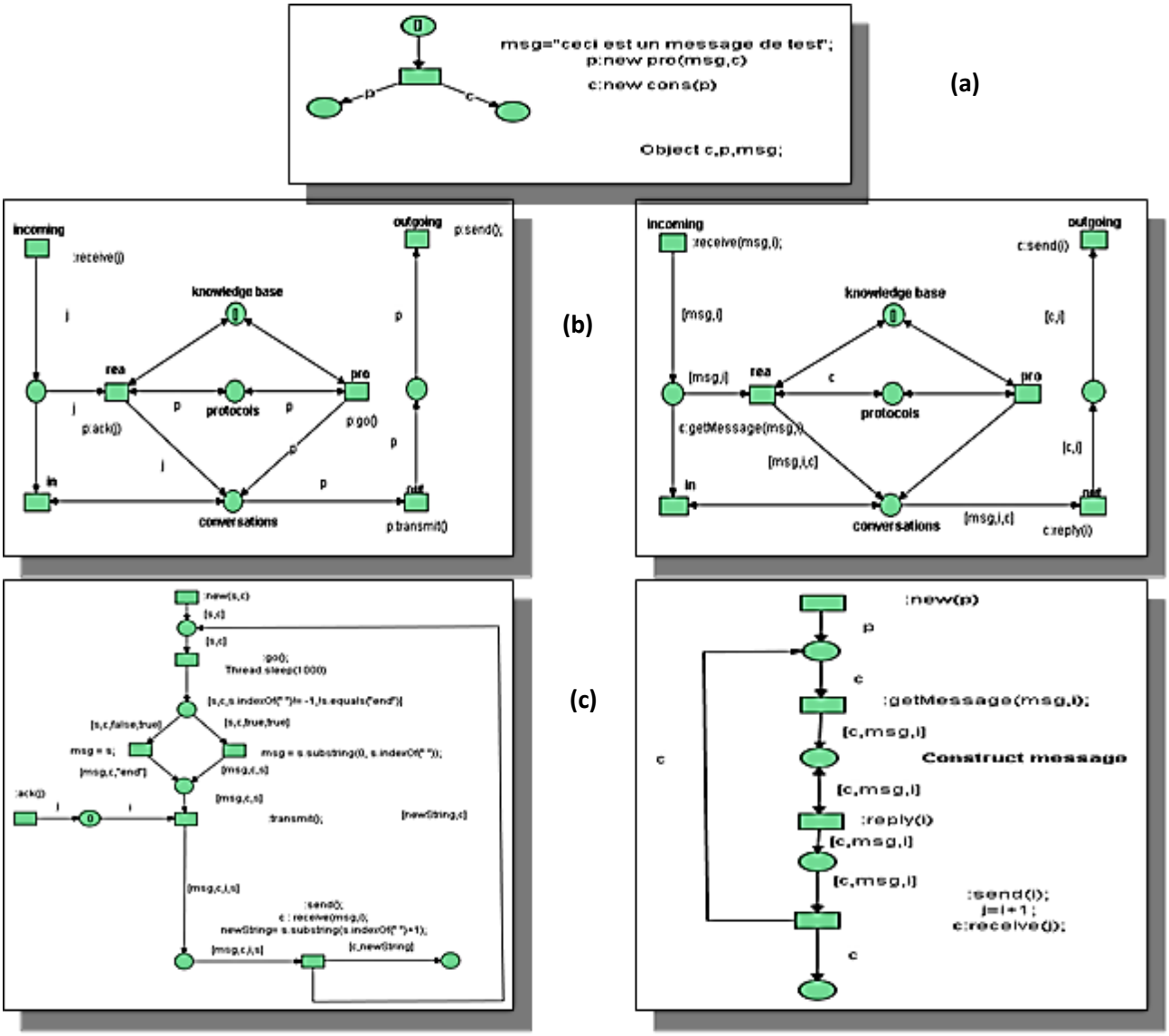


Figure 7-4: Modélisation du cas d'étude.

7.3.3 Concrétisation

Dans la littérature[35], les approches de génération de cas de test ont souffert du problème de l'explosion combinatoire du nombre de cas de tests. Pour pallier cet inconvénient phénoménal, une technique de conception de cas de tests appelée « Error-guessing » est adoptée. Cette dernière est basée sur l'aptitude du testeur de puiser dans ses expériences passées, ses connaissances et son intuition pour mieux détecter la localisation des défauts dissimulés dans un système sous test. L'ingénieur de test peut, ainsi, énumérer une liste des situations à fort risque d'erreurs et préparer un ou plusieurs scénarios d'exécution pour les vérifier.

Par ailleurs, la concrétisation est l'étape la plus délicate dans le processus de test basé sur les modèles. Cette étape agit comme un traducteur qui transforme les cas de tests abstraits en cas de tests concrets. Par la suite, ces cas de tests concrets peuvent être soumis à l'application sous test. Le translateur ayant pour but de réduire le gap entre le modèle abstrait et le système concret en ajoutant des informations manquantes et en transformant les entités abstraites en des structures du langage de la plateforme de test.

La solution MBT assure la couverture du code (système sous test), voir figure 7-5.

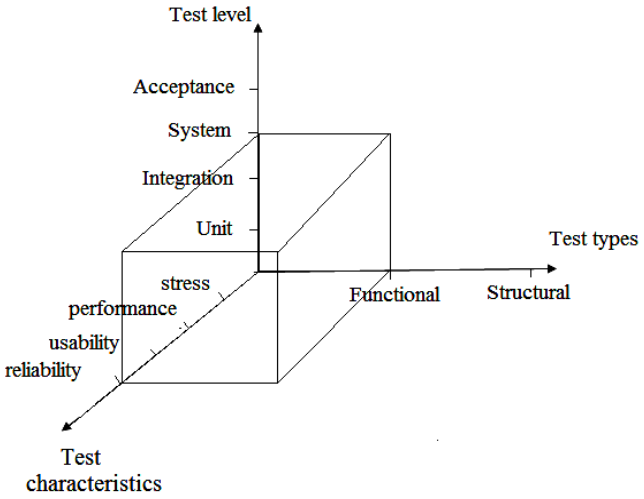


Figure 7-5 : portée de la technique MBT[31].

Cependant, Toutes les approches de test existantes apportent des modifications au niveau du code source de l'application en insérant certain ensemble de routines déterminées (primitives d'instrumentation) dans le programme sous test. Lesquelles en s'exécutant

modifient le comportement initial de ce programme (ce qui concerne les aspects fonctionnel où les aspects non-fonctionnel).

De ce fait, nous proposons de faire le contraire, c'est à dire faire l'instrumentation au niveau du modèle. Puisque les réseaux de référence sont eux-mêmes des objets Java : faire des appels du code Java vers le réseau est facile. Cette phase commence avec le lancement du moniteur de test que nous avons développé (MATT : Multi Agent Testing Tool).

Néanmoins, et avant qu'elle ne soit déclenchée, un travail de fond doit être réalisé en premier, Il s'agit de l'analyse statique.

Concernant le système sous test, l'analyse statique du code source permet de récupérer tous les détails (noms d'agents, noms des classes, méthodes, paramètres, types, etc.) dans un fichier d'informations de l'application sous test. Comme le montre la figure 7-6.

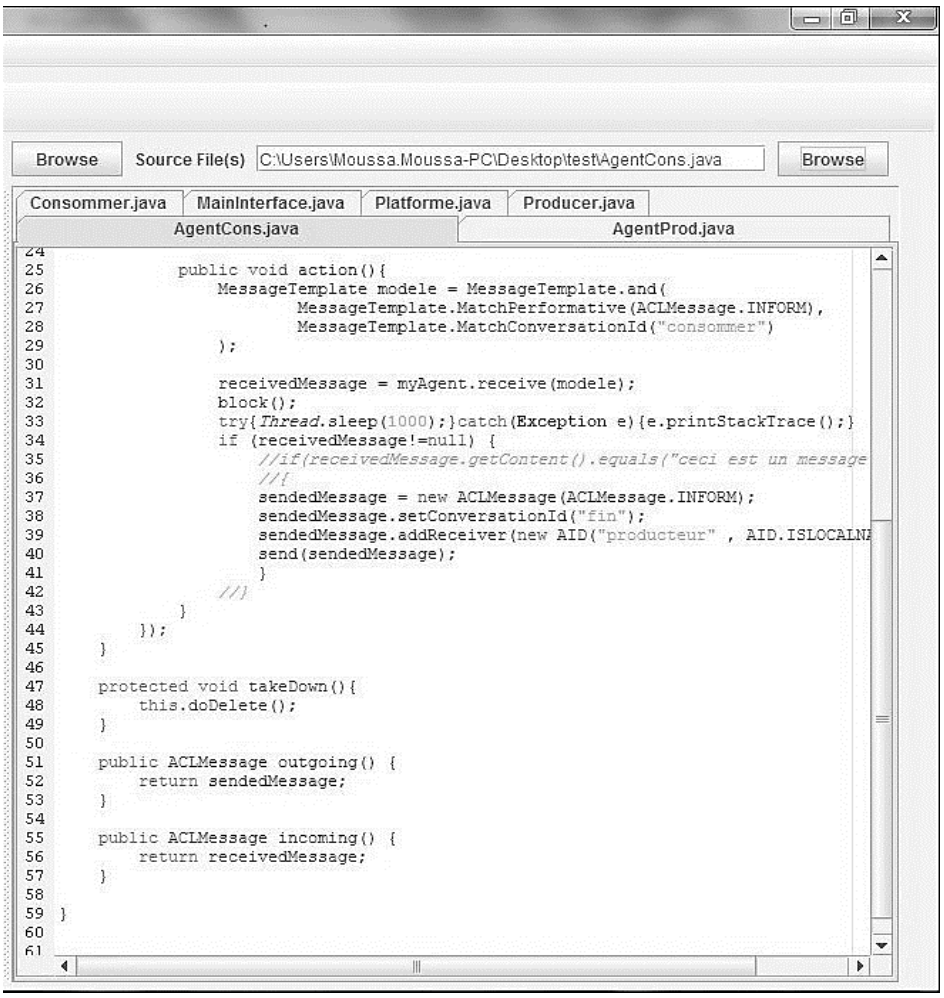


Figure 7-6 : Le résultat de l'analyse statique du code source

Concernant le modèle de test validé, tous les réseaux sont traduits en des fichiers PNML (format spécial de Renew basé sur XML pour représenter les modèles).voir figure 7-7.

Une fois que toutes les entrées (fichier d'informations système, fichier d'information de modèle et scénario de test) sont disponibles, le moniteur de test procède à l'instrumentation du scénario de test via l'adjonction dans le modèle d'un certain ensemble de routines déterminées.

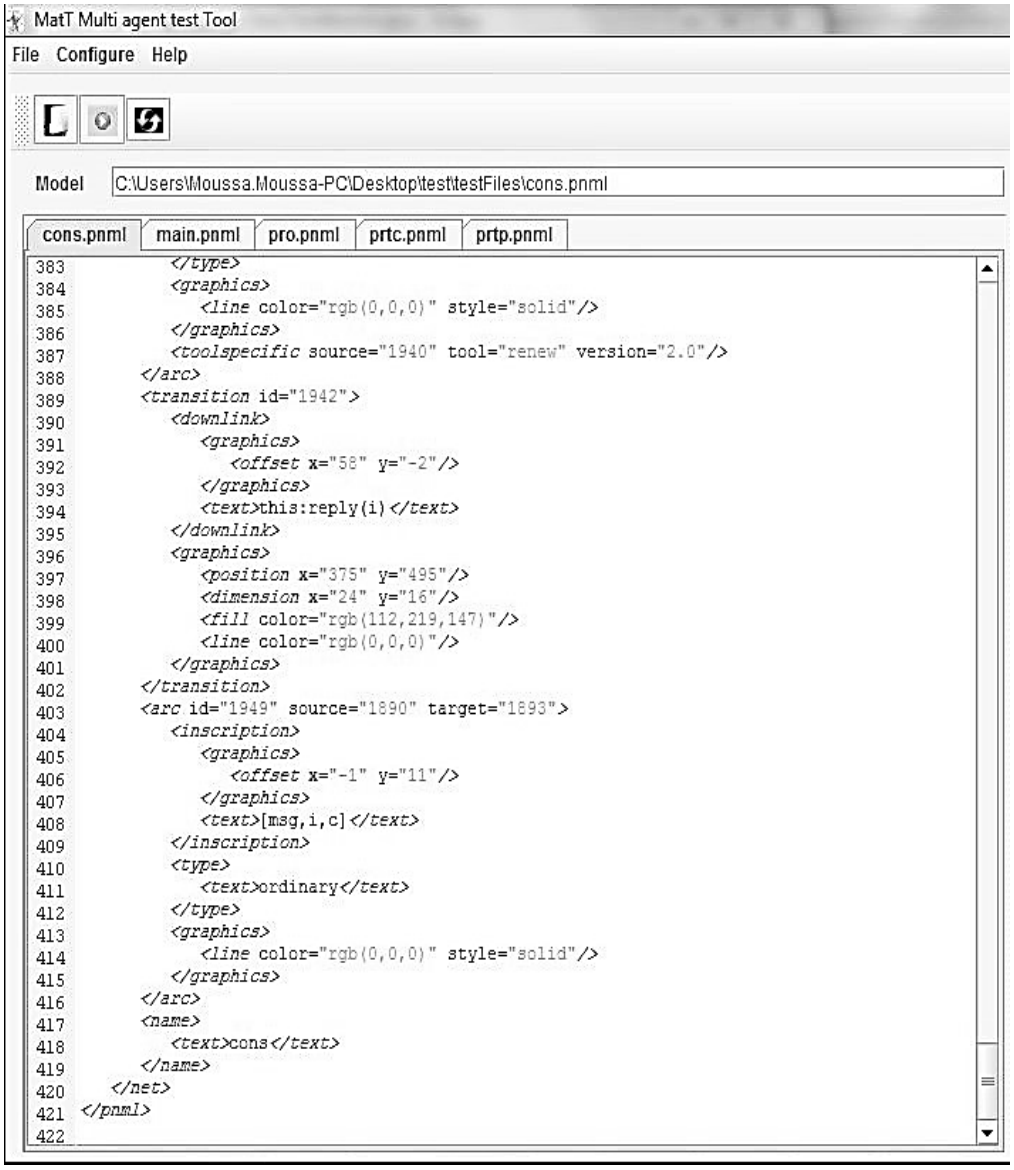


Figure 7-7 : Résultat de l'analyse statique du modèle.

Plus précisément, le testeur sélectionne une de ses *transitions* du modèle avec sa *fonction* correspondante dans le code source. Puis le moniteur de test ajoute les données (automatiquement) dans le modèle pour cette transition. La figure 7-8 montre le module d'instrumentation.

Ceci fait, le testeur peut, à ce stade, accepter, refuser ou modifier les valeurs des variables générées. La figure 7-9 exhibe la version du modèle instrumenté où tous les instruments ajoutés sont encerclés. Le lecteur pourra facilement comparer la figure initiale du modèle (figure 7-4) avec la figure 7-9. Précisons enfin que cette opération est réalisée de manière automatique, ce qui réduit considérablement l'effort de test. Elle laisse également intact le système sous test, ce qui préserve ses spécifications non fonctionnelles (taille du code, temps de réponse, vitesse d'exécution, occupation mémoire, etc.).

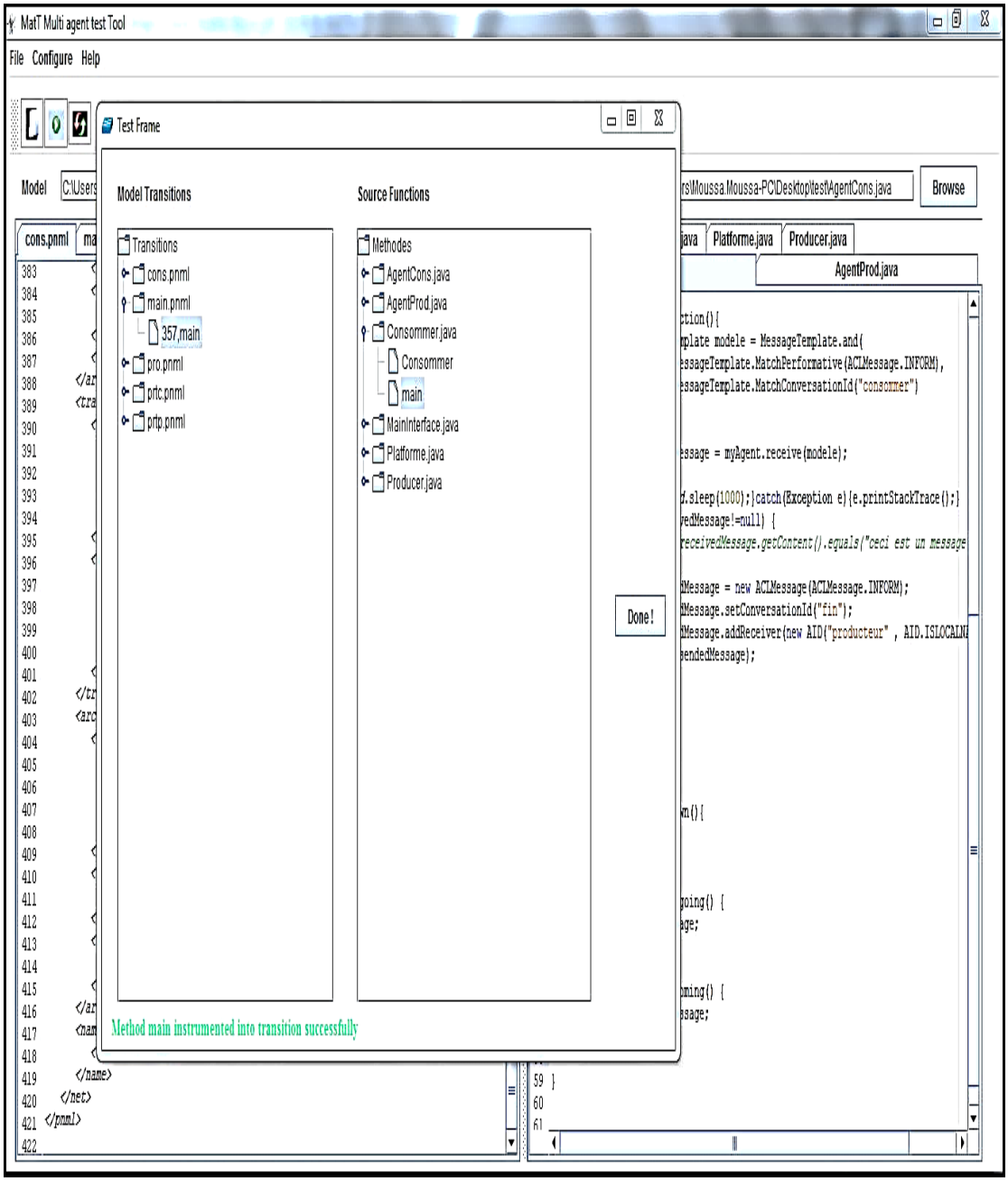


Figure 7-8 : Instrumentation du modèle.

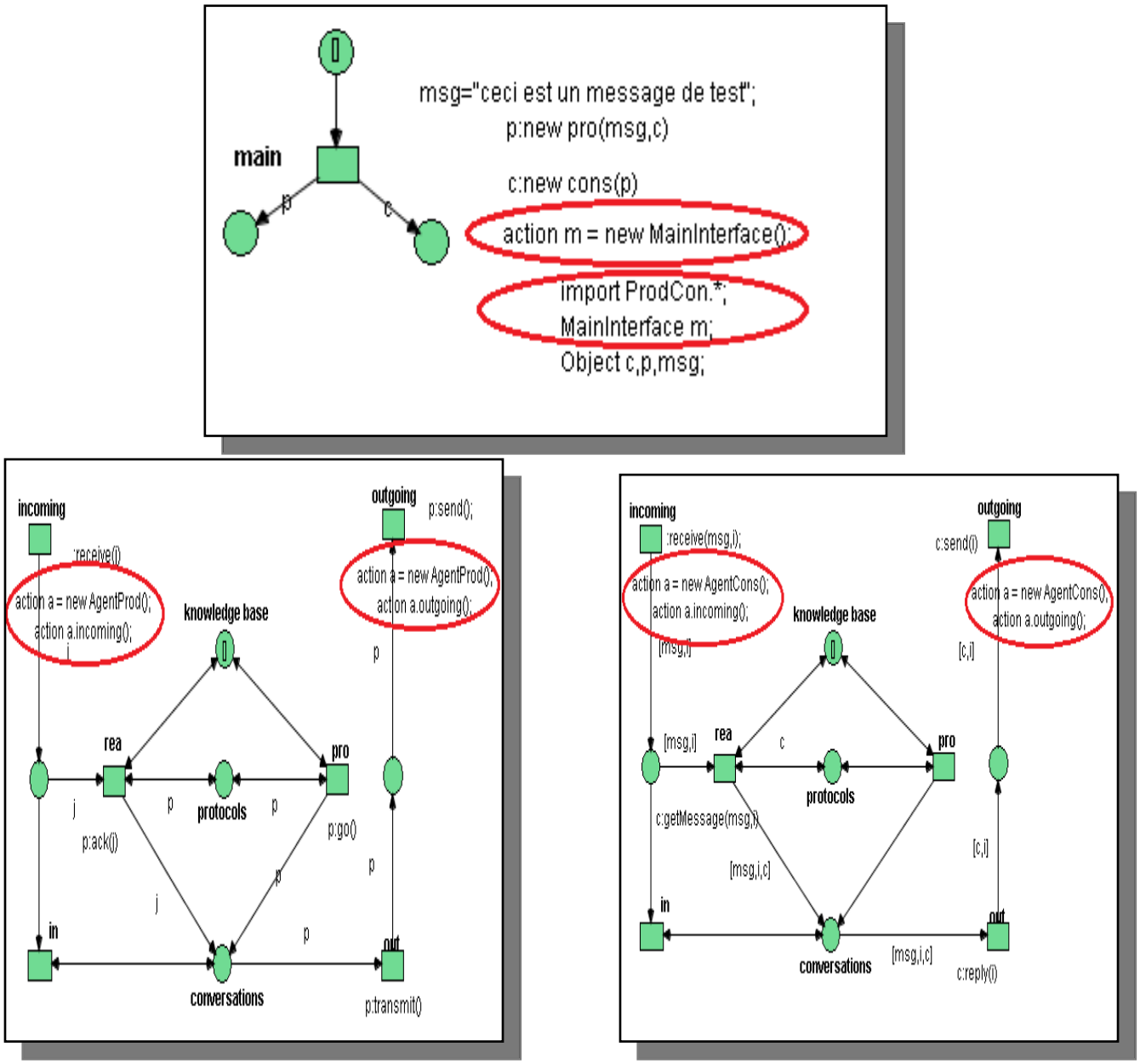


Figure 7-9 : Version modèle instrumenté

7.3.4 Exécution et évaluation des résultats

La dernière phase de l'approche MBT est la phase d'exécution, notre approche consiste à exécuter le modèle instrumenté, tirant profit de l'utilisation des modèles formels, le testeur profite d'une parfaite automatisation de la phase de l'oracle pour pouvoir comparer les résultats obtenus aux résultats attendus pour constituer son verdict. La figure 7-10 présente l'exécution de l'exemple (producteur-consommateur). Une animation (exécution) du modèle instrumenté fait appel aux fonctions concernées par le test dans l'application. Si le test passe

correctement le modèle continue l'animation vers une nouvelle place ou transition comme montré sur (a). Sinon, un message d'erreur s'affiche sur la ligne de commande. Sur la figure (b), par exemple, le test échoue à cause d'une erreur dans le programme développé.

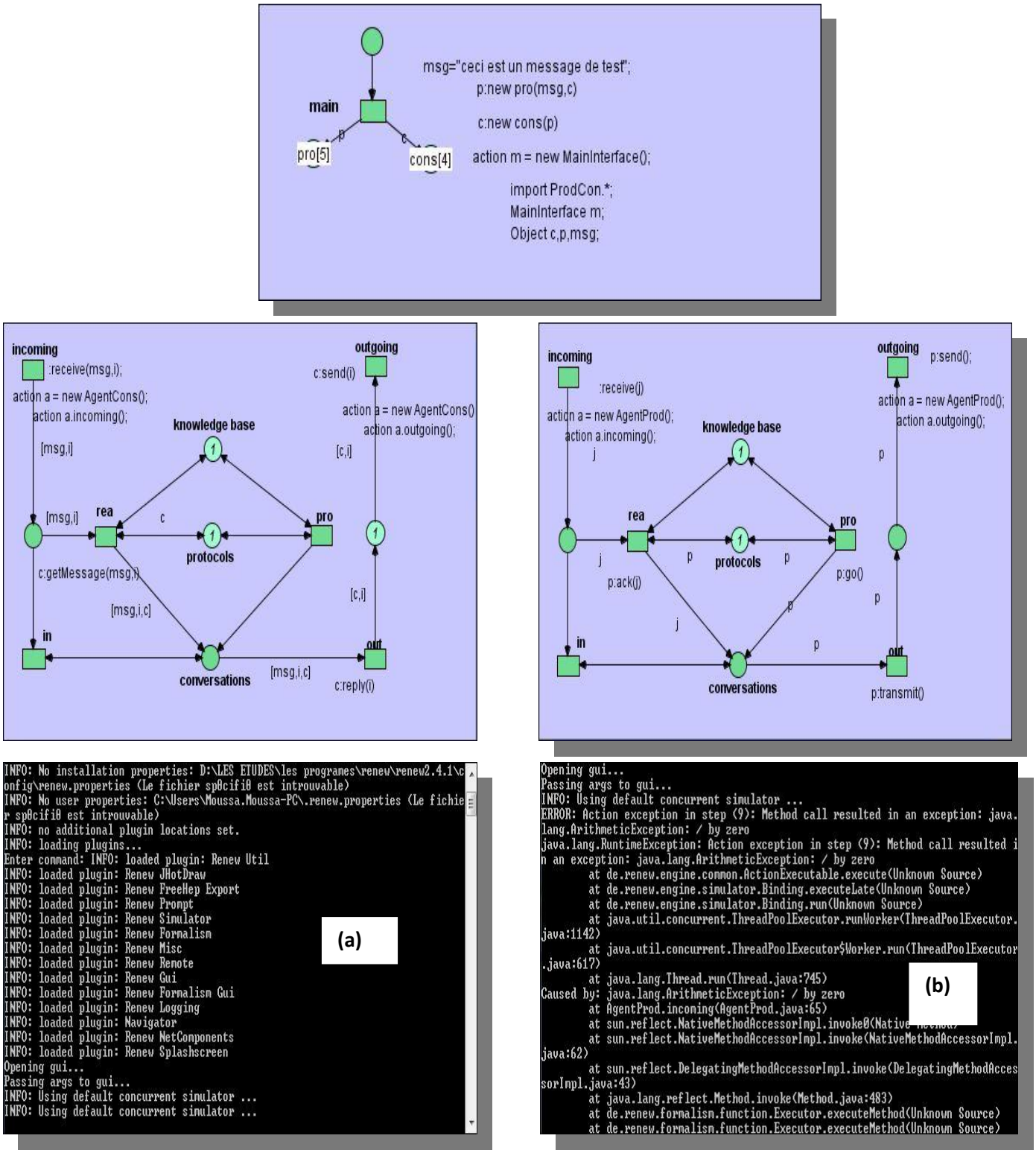


Figure 7-10 : Exécution et évaluation des résultats.

7.4 Discussion

Le lecteur pourrait hâtivement chercher le rapport entre le cas d'étude présenté dans ce chapitre (le modèle producteur-consommateur) et un système ambiant. La réponse à cette question tout à fait logique est que le modèle producteur-consommateur est un modèle abstrait de communication entre deux agents quelconques (l'agent de cuisine et l'agent machine à café présentés dans la chapitre précédent).

Dans une situation simple ou la personne âgée est toute seule dans sa maison, et une fois que les agents sont crées : agent producteur (agent cuisine) et l'agent consommateur (agent machine à café). Un message « Bonjour ! Préparer café svp » est créé par l'agent de cuisine qui doit le transmettre à l'agent machine à café sans besoin de le fragmenter. De l'autre côté l'agent machine à café dès qu'il reçoit le message, il émet un accusé de réception vers l'agent de cuisine. Nous sommes donc dans une situation de communication entre un producteur et consommateur sans fragmentation du message échangé entre les deux agents.

Dans d'autre cas plus généraux, à titre d'exemple où la personne âgée n'est pas seul dans sa maison (présence de plusieurs personnes avec des préférences diverses), et une fois que les agents sont crées : agent producteur (agent cuisine) et l'agent consommateur (agent machine à café). Dans cette situation, plusieurs messages seront créés par l'agent producteur : « Bonjour ! Préparer café pour Monsieur svp », « Préparer du thé pour Madame svp ». Etant donné que le consommateur (l'agent machine à café) ne pourra pas satisfaire toutes les demandes simultanément, la fragmentation des messages est nécessaire. Après fragmentation des messages, l'agent cuisine envoie les fragments vers l'agent machine à café et attend un accusé de réception du consommateur pour envoyer le deuxième fragment. En contrepartie l'agent consommateur quand il reçoit le message il émet un accusé de réception vers le producteur. L'opération ce répète jusqu'à la fin des messages.

La figure 7-11 montre la modélisation de cet exemple en termes de réseaux (nets within nets) aux niveaux : système, agents et protocole.

Le lecteur pourra facilement confirmer le lien entre la figure du modèle présenté ci-dessus (figure 7-11) est celle du modèle producteur-consommateur (figure 7-4).

Ceci dit, tout le processus du test ce répète jusqu'à à la dernière étape qui est celle de l'execution : lors de cette étape on exécute le modèle instrumenté tout en faisant appel aux fonctions concernées par le test dans l'application.

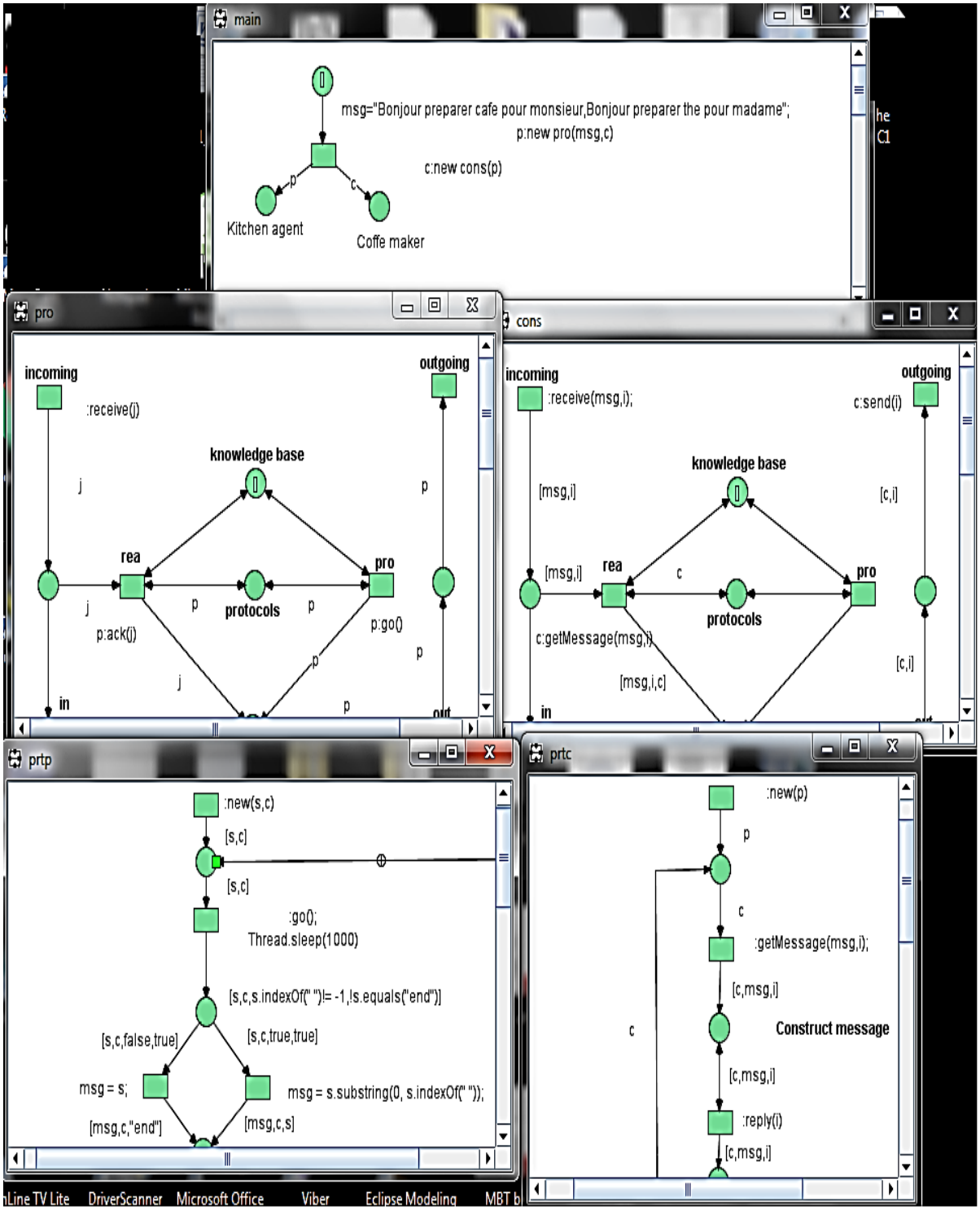
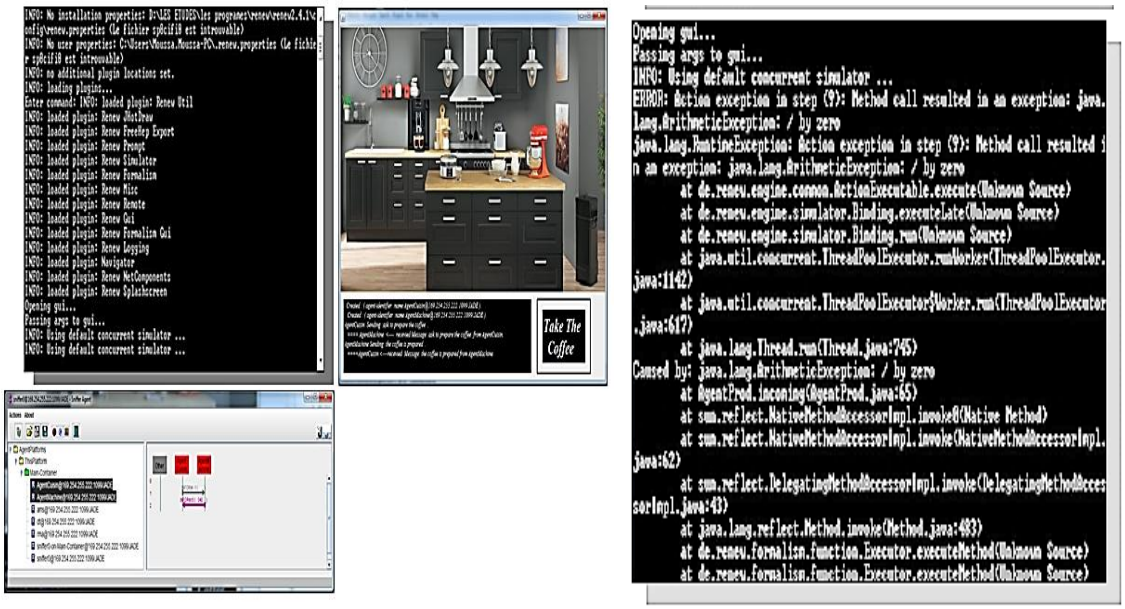


Figure 7-11 : Modélisation d'un scénario du système ambiant.

Si le test passe correctement le modèle continue l'animation vers une nouvelle place ou transition et le programme développé s'ouvre normalement comme montré sur (a) de la figure 7-12. Sinon, un message d'erreur s'affiche sur la ligne de commande de Renew sur (b) de la même figure, par exemple, le test échoue à cause d'une erreur dans le programme développé.



(a) **(b)**

Figure 7-12 : Exécution et analyse des résultats du scénario du système ambiant.

7.5 Conclusion

Dans ce chapitre, nous avons présenté une approche de test basée modèle pour les systèmes multi agents utilisant le paradigme des réseaux de référence. Nous avons choisi comme cas d'étude l'exemple producteur-consommateur.

Un argument important pour l'utilisation du paradigme des réseaux dans les réseaux est que le processus de modélisation conclut non seulement avec un modèle de système en cours d'exécution, mais aussi parce qu'il prend en charge toutes les étapes de la technique de test basée modèle. Cela réduit considérablement les aspects négatifs majeurs des techniques MBT et permet aux testeurs de créer et d'exécuter des tests de manière uniforme et automatique.

L'approche proposée s'appuie fortement sur les résultats de la simulation produits par l'outil Renew. Ainsi, nous avons développé un moniteur de test (MATT : Multi Agent Testing Tool) qui reçoit le modèle test et le système à tester en entrée pour réaliser le processus de test

dans sa totalité. Plus précisément, une analyse statique des structures internes des agents à tester offre à MATT une vue globale sur la manière selon laquelle les agents sont structurés. Ces informations sont ensuite insérées dans le modèle de test. L'exécution du modèle (en même temps que le programme sous test) permet de vérifier si les réponses du système sont conformes aux résultats attendus.

Conclusion générale et perspectives

1. Conclusion générale

La qualité d'un logiciel revêt une importance grandissante dans notre société où l'informatique est de plus en plus présente dans notre quotidien. De nombreuses techniques de vérification ont été développées pour caractériser et détecter des erreurs dans les logiciels. La technique la plus répandue est celle des tests.

Cependant, le processus de test est un processus fastidieux et constitue l'étape la plus exorbitante et la plus difficile à mettre en œuvre dans le cycle de développement des logiciels.

Son importance nous pousse à donner un intérêt particulier et à utiliser et même à développer des outils permettant, lorsque cela est possible, d'automatiser ou d'aider dans la poursuite de ses différentes phases.

C'est dans ce contexte que s'inscrit le travail de cette thèse qui vise à proposer une approche de test basée sur des modèles formels pour les applications agents dans les environnements d'intelligence ambiante.

Pour cela, nous nous sommes intéressés, dans un premier temps, à l'étape de modélisation d'un système ambiant à base d'agent (maison intelligente pour personnes âgées) en utilisant les réseaux dans les réseaux augmenté du système expert JESS. A l'issue de cette étude, nous avons pu conclure que le processus de modélisation conclut avec un modèle exécutable, de plus, le nombre d'erreurs est réduit à cause de l'utilisation d'un seul modèle formel. Enfin cette approche rend aisés l'ajout ou la suppression de nouveaux agents, dispositifs ou même faits et règles d'inférences.

Dans un second temps, nous avons proposé une approche de test basée sur les modèles pour les systèmes à base d'agents. Les intérêts d'une telle approche s'incarnent dans le fait qu'elle couvre tous les niveaux d'un système multi agents à savoir : le niveau algorithmique, classe, agent, intégration et système. De plus, Elle permet de pallier le problème de l'explosion combinatoire du nombre de cas de test et ce en adoptant une technique de conception de cas de tests appelée « Error-guessing » pour la génération de scénarios de test. Ce scénario sera instrumenté au niveau modèle, ce qui constitue un autre avantage par rapport

aux autres techniques dans le sens où le comportement initial de l'application ne sera pas modifié (les aspects fonctionnel et non fonctionnel). Cette phase, également appelée concrétisation de cas de test abstrait, rend aisée l'exécution des cas de test sur l'application et simplifie la phase de l'oracle.

Aussi un outil de test (MATT) a été mis en œuvre permettant ainsi de consolider les aspects de l'approche proposée par une touche pratique. En effet, Les résultats de test générés par l'outil de test faciliteront aux testeurs la construction d'un verdict concernant l'application.

2. Perspectives

Comme suite aux travaux présentés dans cette thèse, plusieurs points restent à développer et à améliorer. Parmi lesquels citons :

- L'application de notre approche de test sur une application qui contient un plus grand nombre d'agents.
- Puisque notre approche de test est basée sur des modèles formels, l'automatisation des modèles de construction et de validation sera une autre perspective.
- Une dernière perspective, consiste à intégrer la couverture structurelle à notre outil de test pour tester d'autres caractéristiques des applications ambiante à base d'agents (robustesse, sécurité et fiabilité).

Références bibliographiques

[1] M. Weiser, R. Gold, and J. S. Brown, “The origins of ubiquitous computing research at PARC in the late 1980s,” *IBM systems journal*, vol. 38, no. 4, pp. 693-696, 1999.

[2] M. Weiser, “Some computer science issues in ubiquitous computing,” *Communications of the ACM*, vol. 36, no. 7, pp. 74-85, 1993.

[3] M. Weiser, and J. S. Brown, "The coming age of calm technology," *Beyond calculation*, pp. 75-85: Springer, 1997.

[4] N. Jeremijenko, “Live Wire (Dangling String).(1995),” 1995.

[5] E. Zelkha, “The future of information appliances and consumer devices,” *Palo Alto Ventures, Palo Alto, California*, 1998.

[6] E. Aarts, and B. De Ruyter, “New research perspectives on Ambient Intelligence,” *Journal of Ambient Intelligence and Smart Environments*, vol. 1, no. 1, pp. 5-14, 2009.

[7] Á. Bermejo Nieto, and C. de los Ángeles, “Noelia. 2004,” *Inteligencia Ambiental*.

[8] P. Reignier, “Intelligence Ambiante Pro-Active: de la Spécification à l'Implémentation,” Doctoral dissertation, Université Joseph-Fourier-Grenoble I, 2010.

[9] G. Chen, and D. Kotz, *A survey of context-aware mobile computing research*, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[10] J. C. Augusto, “Past, present and future of ambient intelligence and smart environments,” in International conference on agents and artificial intelligence, Springer, 2009, pp. 3-15.

[11] A. Yachir, “Composition dynamique de services sensibles au contexte dans les systèmes intelligents ambiants,” Doctoral dissertation, Université Paris-Est, 2014.

[12] F. Sadri, “Ambient intelligence: A survey,” *ACM Computing Surveys*, vol. 43, no. 4, pp. 36, 2011.

[13] M. Wooldridge, *An introduction to multiagent systems*: John Wiley & Sons, 2009.

[14] J. Y. Han, "Multi-touch interaction wall," *in ACM SIGGRAPH 2006 Emerging technologies,2006.* p. 25.

[15] M. Vacher, F. Portet, A. Fleury, and N. Noury, “Development of audio sensing technology for ambient assisted living: Applications and challenges,” *International Journal of E-Health and Medical Communications*, vol. 2, no. 1, pp. 35-54, 2011.

[16] K. Ducatel, *Scenarios for ambient intelligence in 2010*: Office for official publications of the European Communities Luxembourg, 2001.

[17] R. Bergmann, “Ambient Intelligence for Decision Making in Fire Service Organizations,” in European Conference on Ambient Intelligence, Springer, 2007, pp. 73-90.

[18] E. F. Buiël, and J. Lubbers, “Educational agents for the training of tunnel operators,” *ISCRAM - Intelligent Human Computer Systems for Crisis Response Management*, 2007.

[19] P. Martín, M. Sánchez, L. Álvarez, V. Alonso, and J. Bajo, "Multi-agent system for detecting elderly people falls through mobile devices," *Ambient Intelligence-Software and Applications*, pp. 93-99: Springer, 2011.

[20] C. H. Lim, P. Anthony, and L. C. Fan, “Applying multi-agent system in a context aware smart home,” *Learning*, vol. 24, pp. 53-64, 2009.

[21] I. Mocanu, L. Negreanu, and A. M. Florea, "A multi-agent system for service acquiring in smart environments," *Intelligent Distributed Computing VI*, pp. 297-306: Springer, 2013.

[22] Q. Sun, Yu, W., Kochurov, N., Hao, Q., & Hu, F, “A multi-agent-based intelligent sensor and actuator network design for smart house and home automation,” *Journal of Sensor and Actuator Networks*, vol. 2, no 3, pp. 557-588, 2013.

[23] J. A. Fraile, J. Bajo, A. Abraham, and J. M. Corchado, "Hocama: Home care hybrid multiagent architecture," *Pervasive Computing*, pp. 259-285: Springer, 2009.

[24] Y. L. Hsu, Chou, P. H., Chang, H. C., Lin, S. L., Yang, S. C., Su, H. Y., Chang, C. C., Cheng, Y. S., … Kuo, Y. C., “Design and Implementation of a Smart Home System Using Multisensor Data Fusion Technology,” *Sensors (Basel, Switzerland)*, vol. 17(7),1631, 2017.

Références bibliographiques

[25] F. Hong, S. You, M. Wei, Y. Zhang, and Z. J. S. Guo, "MGRA: Motion gesture recognition via accelerometer," vol. 16, no. 4, pp. 530, 2016.

[26] A. Paul, and O. Jeff, "Introduction to software testing," Cambridge University Press, New York, NY, USA, 2008.

[27] A. Bertolino, ``*knowledge area description of software testing-SWEBOK,"*, Tech. Rep., Joint IEEE-ACM Software Engineering Coordinating Committee, 2000.www. swebok. org, 2000.

[28] *AFCIQ Agence Française de Contrôle Industriel de la Qualité*.

[29] S. Xanthakis, P. Régnier, and C. Karapoulios, *Le test des logiciels*: Hermès science publications, 2000.

[30] T. A. Majchrzak, *Improving software testing: technical and organizational developments*: Springer Science & Business Media, 2012.

[31] S. Weißleder, "Test models and coverage criteria for automatic model-based test generation with UML state machines," Doctoral dissertation, Humboldt University of Berlin, 2010.

[32] Y. Kissoum, "Test des systèmes multi-agents," Doctoral dissertation, Université de constantine, 2010.

[33] D. Gelperin, and B. Hetzel, "The growth of software testing," *Communications of the ACM*, vol. 31, no. 6, pp. 687-696, 1988.

[34] L. Luo, "Software testing techniques," *Institute for software research international Carnegie mellon university Pittsburgh, PA*, vol. 15232, no. 1-19, pp. 19, 2001.

[35] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, "The Art of Software Testing," New Jersey: John Wiley & Sons, Inc., 2004.

[36] *National Bureau of Standards, Washington DC, Guidelines for life cycle validation, verification and testing of computer software, 1983.*

[37] R. E. Fairley, "Tutorial: Static analysis and dynamic testing of computer software," *Computer*, vol. 11, no. 4, pp. 14-23, 1978.

[38] K. E. Wiegers, *Peer reviews in software: A practical guide*: Addison-Wesley Boston, 2002.

[39] I. Gomes, P. Morgado, T. Gomes, and R. Moreira, "An overview on the static code analysis approach in software development," *Faculdade de Engenharia da Universidade do Porto, Portugal*, 2009.

[40] S.-D. Gouraud, "Utilisation des structures combinatoires pour le test statistique," Doctoral dissertation, Université Paris Sud-Paris XI, 2004.

[41] M. E. Khan, "Different approaches to black box testing technique for finding errors," *International Journal of Software Engineering Applications*, vol. 2, no. 4, pp. 31, 2011.

[42] J. B. Goodenough, and S. L. Gerhart, "Toward a theory of test data selection," *IEEE Transactions on software Engineering*, no. 2, pp. 156-173, 1975.

[43] J. Favela, Rodríguez, M., Preciado, A., & Gonzalez, V. M. , " Integrating context-aware public displays into a mobile hospital information system," *IEEE transactions on information technology in Biomedicine*, vol. 8(3), pp. 279-286, 2004.

[44] M. D. Rodriguez, Favela, J., Martínez, E. A., & Muñoz, M. A. , "Location-aware access to hospital information and services," *IEEE Transactions on information technology in biomedicine*, vol. 8(4), pp. 448-455, 2004.

[45] M. D. Rodríguez, Favela, J., Preciado, A., & Vizcaíno, A. , "Agent-based ambient intelligence for healthcare," *Ai Communications*, vol. 18(3), pp. 201-216, 2005.

[46] F. Amigoni, Gatti, N., Pinciroli, C., & Roveri, M. , "What planner for ambient intelligence applications?," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 35(1), pp. 7-21, 2005.

[47] J. C. Augusto, H. Nakashima, and H. Aghajan, "Ambient Intelligence and Smart Environments: A State of the Art," *Handbook of Ambient Intelligence and Smart Environments*, H. Nakashima, H. Aghajan and J. C. Augusto, eds., pp. 3-31, Boston, MA: Springer US, 2010.

[48] Z. Houhamdi, "Multi-agent system testing: A survey," *International Journal of Advanced Computer*, 2011.

Références bibliographiques

[49] C. Rouff, " A test agent for testing agents and their communities," *in Aerospace Conference Proceedings,IEEE,2002.* pp. 5-2638.

[50] Z. Zhang, J. Thangarajah, and L. Padgham, "Model based testing for agent systems," *Software and Data Technologies*, pp. 399-413: Springer, 2008.

[51] E. E. Ekinci, A. M. Tiryaki, Ö. Çetin, and O. Dikenelli, "Goal-oriented agent testing revisited," in International Workshop on Agent-Oriented Software Engineering, 2008, pp. 173-186.

[52] A. M. Tiryaki, S. Öztuna, O. Dikenelli, and R. C. Erdur, "SUNIT: A Unit Testing Framework for Test Driven Development of Multi-Agent Systems," *Agent-Oriented Software Engineering VII,2007.* pp. 156-173.

[53] O. Dikenelli, R. C. Erdur, and O. Gumus, "Seagent: a platform for developing semantic web based multi agent systems," *in Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems,2005.* pp. 1271-1272.

[54] R. Coelho, U. Kulesza, A. von Staa, & C. Lucena, "Unit testing in multi-agent systems using mock agents and aspects," *in Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems.* pp. 83-90.

[55] D. N. Lam, and K. S. Barber, "Debugging Agent Behavior in an Implemented Agent System," *Programming Multi-Agent Systems,2005.* pp. 104-125.

[56] M. Núñez, I. Rodríguez, and F. Rubio, "Specification and testing of autonomous agents in e-commerce systems," *Software Testing, Verification Reliability,* vol. 15, no. 4, pp. 211-233, 2005.

[57] C. D. Nguyen, S. Miles, A. Perini, P. Tonella, M. Harman, and M. Luck, "Evolutionary testing of autonomous software agents," *Autonomous Agents and Multi-Agent Systems*, vol. 25, no. 2, pp. 260-283, September 01, 2012.

[58] L. Padgham, M.Winikoff, , & D. Poutakidis, "Adding debugging support to the Prometheus methodology," *Engineering Applications of Artificial Intelligence,* vol. 18(2), pp. 173-190, 2005.

[59] L. Rodrigues, Carvalho, G., Paes, R., & C. Lucena, "Towards an integration test architecture for open MAS," *In 1st workshop on Software Engineering for agent-oriented systems/SBES,* pp. 60-66, 2005.

[60] E. Serrano, and J. A. Botia, "Infrastructure for Forensic Analysis of Multi-Agent Systems," *Programming Multi-Agent Systems,2009.* pp. 168-183.

[61] T. De Wolf, G. Samaey, & T. Holvoet, "Engineering self-organising emergent systems with simulation-based scientific analysis," *in Proceedings of the Fourth International Workshop on Engineering Self-Organising Applications,2005.* pp. 146-160.

[62] J. Sudeikat, and W. Renz, "A Systemic Approach to the Validation of Self–Organizing Dynamics within MAS," *Agent-Oriented Software Engineering IX,2009.* pp. 31-45.

[63] Z. Houhamdi, & B. Athamena, "Structured system test suite generation process for multi-agent system," *International Journal on Computer Science and Engineering,* vol. 3(4), pp. 1681-1688, 2011.

[64] N. Louloudakis, A. Leonidis, and C. Stephanidis, "AmITest: A Testing Framework for Ambient Intelligence Learning Applications," *eLmL,* pp. 87, 2016.

[65] J. M. Conejero, P. J. Clemente, R. Rodríguez-Echeverría, J. Hernández, and F. Sánchez-Figueroa, "A model-driven approach for reusing tests in smart home systems," *Personal ubiquitous computing,* vol. 15, no. 4, pp. 317-327, 2011.

[66] C. Xu, S. C. Cheung, X. Ma, , C. Cao, & J. Lu, "Adam: Identifying defects in context-aware adaptation," *Journal of Systems and Software,* vol. 85(12), pp. 2812-2828, 2012.

[67] T. H. Tse, & S. S. Yau, "Testing context-sensitive middleware-based software applications," *Computer Software and Applications Conference, 2004. COMPSAC 2004.* pp. 458-466.

[68] A. J. Ramirez, A. C. Jensen, B. H. Cheng, & D. B. Knoester, "Automatically exploring how uncertainty impacts behavior of dynamically adaptive systems," *in Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering,2011.* pp. 568-571.

Références bibliographiques

[69] T. Griebe, & V. Gruhn, "A model-based approach to test automation for context-aware mobile applications," *In Proceedings of the 29th Annual ACM Symposium on Applied Computing,2014.* pp. 420-427.

[70] D. Amalfitano, A. R. Fasolino, P. Tramontana, & N. Amatucci, "Considering context events in event-based testing of mobile applications," *In 2013 IEEE sixth international conference on software testing, verification and validation workshops.* pp. 126-133.

[71] M. Jang, Kim, J., & J. C. Sohn, "Simulation framework for testing context-aware ubiquitous applications," *In The 7th International Conference on Advanced Communication Technology, 2005, ICACT 2005.* pp. 1337-1340.

[72] Z. Wang, S. Elbaum, & D. S. Rosenblum, "Automated generation of context-aware tests," *In Proceedings of the 29th international conference on Software Engineering.* pp. 406-415.

[73] H. Lu, W. K. Chan, & T. H. Tse, "Testing context-aware middleware-centric programs: a data flow approach and an RFID-based experimentation.," *in Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering,2006.* pp. 242-252.

[74] H. Lu, Chan, W. K., & T. H. Tse, "Testing pervasive software in the presence of context inconsistency resolution services," *in Proceedings of the 30th international conference on Software engineering,2008.* pp. 61-70.

[75] Y. Qin, C. Xu, P. Yu, and J. Lu, "Sit: Sampling-based interactive testing for self-adaptive apps," *Journal of Systems Software,* vol. 120, pp. 70-88, 2016.

[76] P.-A. Muller, F. Fondement, B. Baudry, and B. Combemale, "Modeling modeling modeling," vol. 11, no. 3, pp. 347-359, July 01, 2012.

[77] B. Combemale, "Approche de métamodélisation pour la simulation et la vérification de modèle--Application à l'ingénierie des procédés," doctoral dissertation, Institut National Polytechnique de Toulouse-INPT, 2008.

[78] J.-M. Jézéquel, S. Gérard, and B. Baudry, "Le génie logiciel et l'IDM: une approche unificatrice par les modèles," Lavoisier, Hermes-science, 2006.

[79] Y. Ridene, "Ingéniérie dirigée par les modèles pour la gestion de la variabilité dans le test d'applications mobiles," Doctoral dissertation, Université de Pau et des Pays de l'Adour, 2011.

[80] M. Utting, and B. Legeard, *Practical model-based testing: a tools approach*: Elsevier, 2010.

[81] S. k. Kangoye, "Elaboration d'une approche de vérification et de validation de logiciel embarqué automobile, basée sur la génération automatique de cas de test," Doctoral dissertation, Université d'Angers, 2016.

[82] O. Kummer, F. Wienberg, M. Duvigneau, J. Schumacher, M. Köhler, D. Moldt, H. Rölke, and R. Valk, "An Extensible Editor and Simulation Engine for Petri Nets: Renew," *Applications and Theory of Petri Nets, 2004.* pp. 484-493.

[83] V. Tran, Moraru, V., "Réseau de Petri," *Institut de la Francophonie pour l'Informatique,* Promotion 10 15 juillet 2005.

[84] R. Valette, "Les réseaux de Petri," *LAASCNRS Toulouse,* Septembre 2000.

[85] V. Augusto, "Modélisation des systèmes complexes," *Ecole Nationale Supérieur des Mines de Saint-Etienne*, 2013.

[86] N. T. Dehimi, "Un Cadre Formel pour La Modélisation et L'analyse Des Agents Mobiles," Doctoral dissertation, Univeristé de Constantine 2, 2013.

[87] K. Jensen, "Coloured Petri nets and the invariant-method," *Theoretical computer science,* vol. 14, no. 3, pp. 317-336, 1981.

[88] P. Merlin, "A study of the recoverability of computer systems," Doctoral dissertation, Université de californie, 1974.

[89] C. Ramchandani, *Analysis of asynchronous concurrent systems by Petri nets*, MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC, 1974.

Références bibliographiques

[90] M. Köhler, D. Moldt, and H. Rölke, "Modelling Mobility and Mobile Agents Using Nets within Nets," *Applications and Theory of Petri Nets, 2003.* pp. 121-139.

[91] R. Valk, "Petri Nets as Token Objects," *Application and Theory of Petri Nets, 1998.* pp. 1-24.

[92] R. Valk, "Concurrency in Communicating Object Petri Nets," *Concurrent Object-Oriented Programming and Petri Nets: Advances in Petri Nets*, G. A. Agha, F. De Cindio and G. Rozenberg, eds., pp. 164-195, Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[93] O. Kummer, "Simulating synchronous channels and net instances," 1998.

[94] M. Köhler, & H. Rölke, "Towards a unified approach for modeling and verification of multi-agent systems," in Workshop on Modelling of Objects, Components, and Agents (MOCA 2001), 2001, August, pp. 85-104.

[95] M. Köhler, & H. Rölke, "Mobile object net systems: Concurrency and mobility," *In Proceedings of the International Workshop on Concurrency, Specification, and Programming (CS&P 2002)*.

[96] M. Duvigneau, D. Moldt, & H. Rölke, " Concurrent architecture for a multi-agent platform," *In International Workshop on Agent-Oriented Software Engineering,2002*. pp. 59-72.

[97] Y. Kissoum, S. Kerraoui, & M. L. Boughaouas, "Smart Home for Elderly: Modeling and Simulation," in ICAASE, 2014, pp. 148-155.

[98] S. Kerraoui, Y. Kissoum, M. Redjimi, & M. Saker, " MATT: Multi Agents Testing Tool Based Nets within Nets," *Journal of Information and Organizational Sciences*, vol. 40(2), pp. 165-184, 2016.

[99] Y. Kissoum, Z. Sahnoun, & K. Barkaoui, "An approach for testing mobile agents using the nets within nets paradigm," in 2009 Third International Conference on Research Challenges in Information Science 2009, April, pp. 207-216.

[100] Y. Kissoum, Z. Sahnoun, & K. Barkaoui, "A Formal Approach for Modeling and Testing Agent Interactions Using Recursive Colored Petri Nets," *Recent Patents on Computer Science*, vol. 3(1), pp. 39-53, 2010.