REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITÉ MOHAMED KHIDER BISKRA

Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique

THÈSE

Pour l'obtention du diplôme de

Docteur en Sciences

Une approche dirigée par les modèles pour une conception flexible des systèmes distribués

Présentée par : **Ahmed HARBOUCHE**

Directeur de thèse : **Pr. NourEddine DJEDI**

Soutenue le : **19 Juin 2018**

Devant le jury composé de

Président :	Dr Abdelmalik BACHIR	Professeur	Université Med Khider, Biskra.
Rapporteur :	Dr NouEddine DJEDI	Professeur	Université Med Khider, Biskra.
Examinateurs:	Dr Hamid AZZOUNE	Professeur	USTHB, Alger.
	Dr Allaoua CHAOUI	Professeur	Université Constantine 2.
	Dr Hammadi BENNAOUI	MCA	Université Med Khider, Biskra.

Biskra, Algérie.

Année Universitaire 2017/2018

الملخص

يعتقد اليوم أن الهندسة المدفوعة بالنماذج (IDM) جلبت تغييرا كبيرا في تصميم التطبيقات من خلال الأخذ في الاعتبار استدامة المعرفة و الخبرة إضافة إلى المكاسب المترتبة عن تحسين الإنتاجية. تعتمد الأنظمة من نوع (IDM) على عمليات لتحويل النماذج بهدف تحقيق حل تقني يعتمد على منصة خاصة انطلاقا من نماذج موجهة للمهن ومستقلة عن أي منصة تذكر. في هذا البحث يرتكز اهتمامنا على المراحل الأولى لتطوير الأنظمة الموزعة، و بصفة خاصة عملية اشتقاق لتصورية (تصميم النظام) تخضع لمواصفات ومتطلبات النظام الإجمالي.

و بالنظر إلى أهمية وفائدة نماذج المتطلبات في تطوير النظام والاشتقاق التلقائي لسلوك النظام، قمنا باقتراح منهج لإشتقاق سلوك مختلف المكونات أو الأدوار المختلفة للنظام القائم، مرتكز على عملية تحويل النماذج. عملية التحويل هذه، التي تحكمها مجموعة من القواعد، سوف تسمح باشتقاق السلوك الخاص بكل عنصر من النظام، وهذا انطلاقا من متطلبات النظام الإجمالي. وستتيح هذه العملية أيضا اشتقاق رسائل التنسيق المناسبة بين السلوكيات المشتقة المختلفة. ويمكن أن تؤدي عملية الإشتقاق التلقائي هذه إلى صياغة أنظمة سليمة وقوية من خلال تجنب الأخطاء التي قد تدخلها وتتسبب فيها العملية اليدوية.

ومن الجوانب الهامة في تطوير النظم الموزعة، نذكر ضمان امتثال تصميم النظام مع مواصفاته الأولية، بتعبير آخر في نهج الاشتقاق لدينا، يجب التأكد من صحة السلوكيات المشتقة والتحقق منها قبل نشرها. في الواقع، توفر الطرق الرسمية أدوات قوية لفحص التطبيقات المتنافسة على مستويات مختلفة من دورات حياتها. وعملية التحقق من النماذج هي واحدة من الطرق الأقوى والأجدى للتحقق من الدقة المنطقية لهذه الأنظمة المتنافسة. في هذا البحث، قمنا باقتراح نهج للتحقق و الفحص للنماذج مبني على أساس تحويل النماذج وذلك للتحقق من السلوكيات المستمدة تلقائيا للأنظمة الموزعة. الهدف هو التحقق ما إذا كان النظام المشتق يتصرف بنفس الطريقة المحددة في مواصفاته الشاملة. يهدف نهج التحقق هذا إلى زيادة الأداء وجودة الخدمة للنظام. هذا النهج يسمح للمطور التركيز على النموذج الشامل للنظام بدلا من التركيز على النظام نفسه.

كما يسهم العمل المقدم في هذه الأطروحة أيضا في تصميم أنظمة المراقبة الطبية المعتمدة على شبكات WBSN والتحقق من صحتها. شبكة الاستشعار اللاسلكية للأجسام (WBSN) تتمثل في شبكة وأجهزة استشعار لاسلكية تعلق على جسم الإنسان وذلك للسماح برصد المعلومات الحيوية لهذا الجسم والبيئة المحيطة به. وقد حظي تصميم وتطوير هذه الشبكات قصد رصد صحة الإنسان، بقدر كبير من الاهتمام مؤخرا في مجالات البحث والصناعة. ويرجع هذا الاهتمام المكثف والمتسارع أساسا إلى الرعاية الصحية المكلفة إضافة إلى التقدم المحرز مؤخرا في تطوير الأجهزة المصغرة للمراقبة الطبية. من جهة أخرى، إن توفير نهج تصميم صريح يمثل نقطة مفيدة جدا لبناء وصيانة هذه الأنظمة. وفي هذه المذكرة، نقدم نظاما يتكفل بتحقيق مراقبة طبية وقائية. ويستند هذا النظام على تصميم مرفق بعقدات غير متجانسة ويحقق في نفس الوقت كل من الرصد اليومي والرصد المستمر إضافة إلى مراقبة محددة. فوق هذا كله، فمنا بتعريف وتحديد نموذج لوصف سلوك النظام WBSN. يستخدم نهج إشتقاق السلوك القائم على النموذج للحصول على سلوك كل عقدة في النظام WBSN انطلاقا من السلوك العام للنظام WBSN. يسمح هذا النهج للمطورين للحصول على تصميم للنظام WBSN انطلاقا من المواصفات العامة لاحتياجاته. تسمح طريقة التدقيق الموجه من قبل النماذج المقترحة أيضا بالتحقق ما إذا كان التعاون السلوكي المشتق يستوفي المواصفات العامة للنظام الأولي.

الكلمات المفتاحية: نماذج، نموذج عالي، تحويل النماذج، اشتقاق السلوك، الهندسة المقيدة بالنماذج، MDA، التدقيق، WBSN، الإشراف الطبي.

Résumé

l'Ingénierie Dirigée par les Modèles (IDM) apporte un changement important dans la conception des applications en prenant en compte la pérennité du savoir faire et les gains en productivité. L'architecture IDM se base sur la transformation de modèles pour aboutir à une solution technique sur une plateforme donnée à partir de modèles métier indépendants de toute plateforme. Dans ce travail, nous nous intéressons aux phases initiales de développement des systèmes distribués, particulièrement au processus de dérivation des modèles conceptuels à partir de la spécification des exigences globales du système.

Considérant l'importance et l'utilité du modèle des exigences dans le développement des systèmes et la dérivation automatique du comportement d'un système, nous proposons une approche de dérivation du comportement des différents composants ou rôles d'un système basée sur la transformation de modèles. Le processus de transformation, régi par un ensemble de règles, permettra la dérivation du comportement local d'un composant, sous la forme d'un automate d'états-transitions, à partir des exigences du système décrites par un diagramme d'activités UML étendu avec les collaborations. Ce processus permettra aussi la dérivation des messages de coordination appropriés entre les différents comportements dérivés. Un tel processus de dérivation automatique peut conduire à des systèmes valides et robustes en évitant les erreurs introduites par les activités de conception manuelle.

Un important aspect dans le développement des systèmes distribués est d'assurer la conformité de la conception du système avec sa spécification, autrement dit dans notre approche de dérivation, les comportements dérivés doivent être validés et vérifiés avant leur déploiement. En fait, les méthodes formelles sont des outils puissants pour la vérification des applications concurrentes aux différents niveaux de leurs cycles de vie. La vérification de modèles est l'une des méthodes formelles les plus puissantes pour vérifier l'exactitude logique des systèmes concurrents. Dans ce travail, nous proposons une approche de vérification de modèles basée sur une transformation de modèles pour valider les comportements automatiquement dérivés d'un système distribué. Cette approche dirigée par les modèles vérifiera si le système dérivé a le même comportement que celui décrit dans sa spécification globale. L'objectif est d'augmenter les performances et la qualité de service QoS du système. Cette approche permet au développeur de raisonner sur un modèle du système global plutôt que sur le système lui-même.

Le travail présenté dans cette thèse contribue également à la conception et la validation des applications de surveillance médicale basées sur les WBSNs. Un réseau de capteurs sans fil corporel (Wireless Body Sensor Network (WBSN)) est un réseau de capteurs sans fil attachés au corps humain afin de permettre la surveillance des paramètres vitaux du corps humain et son environnement. La conception et le développement de tels systèmes pour la surveillance médicale (health monitoring) ont récemment reçu beaucoup d'attention, dans les domaines de la recherche et de l'industrie. Cet engouement est principalement motivé par des soins de santé très coûteux et par les progrès récents accomplis dans le développement de dispositifs miniatures pour la surveillance médicale. L'existence d'une approche de conception explicite devrait être très bénéfique pour la construction et la maintenance de ces systèmes. Dans cette thèse, nous présentons un système de surveillance médicale préventive. Ce système est basé sur une architecture dotée de nœuds hétérogènes et réalise à la fois une surveillance quotidienne et continue ainsi que des contrôles spécifiques. Nous avons défini un modèle pour décrire le comportement global du système WBSN. L'approche de dérivation du comportement dirigée par les modèles est utilisée pour obtenir le comportement de chaque nœud dans le WBSN à partir du comportement global du système. Cette approche permet aux développeurs d'obtenir une conception du système WBSN à partir de la spécification globale de ses besoins. L'approche de vérification dirigée par les modèles proposée permet également de vérifier si la collaboration des comportements dérivés satisfait la spécification globale du système initial.

Mots clés : Modèles, Méta-modèle, Transformation de modèles, Dérivation du comportement, Ingénierie dirigée par les modèles, MDA, Vérification, WBSN, Surveillance médicale.

Abstract

The Model Driven Engineering (MDE) makes a significant change in application design by taking on consideration the durability of the expertise and productivity gains. The MDE architecture is based on the model transformation to achieve a technical solution on a given platform from business models independent of any platform. In this work, we are interested in the early development phases of a distributed software system, especially in the process of obtaining a system design from its global requirements specification.

Considering the great importance and usability of the requirements models in practice during a system development, and the importance of the automatic derivation of the system behavior, this work focuses on the derivation of the different system components' behavior. To this end, we suggest an approach to derive the behavior of the system components by transforming the system global requirements model to the design model. The system global requirements model describes the functional behavior of a given system in an abstract way. The design model represents the local behavior of each component. The appropriate meta-models have been defined at each level of abstraction with the corresponding model transformations. The proposed approach allows designers to describe their system using UML activity diagram extended with collaborations. This requirements model is then automatically transformed to the behavior of the components. A set of rules is needed to govern the model transformations during the derivation process. As these derived system components execute in a distributed environment, the derivation process includes synchronization messages in the derived behaviors to ensure coordination between the system components. The collaboration of these behaviors should satisfy the initial requirements model.

To ensure the conformance of this design to its specification, the derived behaviors should be validated and verified before their deployment. In fact, formal methods are powerful tools for software engineers to verify the logical correctness of concurrent software at different levels of its life cycle. Model checking is one of the most powerful formal methods for verifying the logical correctness of such concurrent systems. In this work, we make use of a model checking approach that is based on a model transformation to validate the automatically derived behavior of a distributed system. This model-driven approach will check whether the derived system behaves correctly according to its global specification, while the objective is to increase the system's performance and QoS. This approach allows the developer to reason about a model of the global system rather than about the system itself.

The work presented in this document also contributes to the design and validation of health monitoring systems based on WBSNs. The Wireless Body Sensor Network (WBSN) is a wireless network that is designed to allow communication among sensor nodes that are attached to a human body to monitor the body's vital parameters and environment. The design and development of such WBSN systems for health monitoring have received a large amount of attention recently, in research studies and in industry. This attention is mainly motivated by costly health care and by recent advances in the development of miniature health monitoring devices. The existence of an explicit approach to address the required software design and verification should be very beneficial for the construction and maintenance of such systems. In this work, we present a preventive health care system that has a flexible design. The proposed system is based on an architecture that has heterogeneous nodes and provides both daily continuous monitoring as well as specific controls. The model-driven engineering (MDE) approach is used to derive each node's behavior in the WBSN from the WBSN global behavior. This approach allows developers to obtain a system design from the global specification of its requirement. Then the proposed model-driven verification approach is used to verify whether the collaboration of the derived behaviors satisfies the initial system specification.

Keywords : Models, Meta-model, Model Transformation, Behavior derivation, Model driven engineering, MDA, Verification, WBSN, health monitoring.

Remerciements

Avant, toute personne, je tiens à remercier notre Dieu Tout Puissant pour m'avoir éclaircit le chemin de ce travail.

Mes vifs remerciements vont tout d'abord à Monsieur *Noureddine DJEDI*, Professeur à l'Université Med Khider, Biskra, mon directeur de thèse pour m'avoir donné la possibilité d'effectuer cette thèse sous sa direction. Je tiens aussi à le remercier pour ses ses précieux conseils durant mes travaux de thèse et la préparation de la soutenance.

Je tiens également à remercier vivement Monsieur *Mohammed ERRADI*, Professeur à l'ENSIAS, Université Mohamed V Souissi, Rabat, Maroc pour l'accueil et l'aide précieuse qu'il m'a prodigué tout au long de cette thèse. Je tiens aussi à exprimer toute ma gratitude à Monsieur *Abdellatif KOBBANE*, Professeur à l'ENSIAS, Université Mohamed V Souissi, Rabat, Maroc pour son soutien et pour son aide.

J'adresse mes plus vifs remerciements à Monsieur *Abdelmalik BACHIR*, Professeur à l'Université Med Khider, Biskra, pour m'avoir fait l'honneur de présider le jury de ma soutenance, ainsi qu'à Monsieur *Samy AIT-AOUDIA*, Professeur à l'ESI, Alger, Monsieur *Hamid AZZOUNE*, Professeur à l'USTHB, Alger, Monsieur *Allaoua CHAOUI*, Professeur à l'Université Constantine 2, et Monsieur *Hammadi BENNAOUI*, Maître de conférences à l'Université Med Khider, Biskra pour avoir accepté d'examiner mon travail et pour avoir bien voulu juger le travail et faisant partie du jury de soutenance.

Un grand merci s'adresse en particulier à Monsieur *KOUIDER EL OUAHED Abdellah*, mon collègue au département d'informatique de l'UHBC de Chlef, qui m'a soutenu pendant toute la durée de ma thèse, et sans oublier toutes les personnes qui m'ont soutenu de prés ou de loin, je m'excuse de ne pouvoir tous les mentionner.

Et enfin, je remercie du fond de mon cœur mon défunt père et ma mère qui ont veillé à mon éducation, toute ma famille et tous mes proches et amis, qu'ils soient chaleureusement remerciés de m'avoir soutenu pour mener à bien cette thèse.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Les systèmes informatiques deviennent de plus en plus complexes. Il devient alors difficile de maintenir correctement les logiciels à moindre coût et ceci dans des délais raisonnables. Cette évolution consiste à redévelopper et porter ces logiciels vers de nouvelles plateformes, alors que globalement la logique métier de l'entreprise n'a pas changé.

L'évolution incessante et inévitable des technologies a été prise en compte au niveau des spécifications des logiciels, avec l'apparition de nouvelles méthodes d'analyse et de conception à base de modèles et à production de code. La plus célèbre de ces méthodes de modélisation de systèmes à objets est UML (Unified Modeling Language) de l'OMG. Avec ce langage graphique, les différentes dimensions d'un logiciel complexe peuvent être spécifiées, assemblées, visualisées et documentées. Ces différentes modélisations sont ensuite utilisées pour générer automatiquement la trame de code du logiciel cible sur une plateforme.

Pour permettre l'interopérabilité entre les différents systèmes, réduire les coûts de développement et augmenter l'évolutivité, l'ingénierie dirigée par les modèles (IDM), ou Model Driven Engineering (MDE) en anglais, est devenue un domaine important de l'ingénierie logicielle. Elle a permis plusieurs améliorations significatives dans le développement de systèmes complexes en se concentrant sur une préoccupation plus abstraite que la programmation classique. C'est une forme d'ingénierie générative dans laquelle tout ou une partie d'une application est engendrée à partir

de modèles. Pour l'IDM, le concept principal est le modèle. Un modèle est une abstraction, une simplification d'un système permettant de le comprendre. Le modèle représente une vue d'un système et il est défini dans le langage de son méta-modèle. Un système peut être décrit par différents modèles liés les uns aux autres. L'utilisation de différents langages de modélisation pour décrire les aspects chronologiques ou technologiques du développement d'un système nécessite la définition de plusieurs méta-modèles. Afin de rendre opérationnels les modèles (pour la génération de code, la documentation et le test, la validation, la vérification, l'exécution, etc.), IDM introduit la notion de transformation de modèles. Autrement dit, un ensemble d'applications exécutables qui indiquent comment dériver un modèle cible à partir d'un modèle source. Les mappages sont décrits en termes de correspondances entre les concepts du méta-modèle source et ceux du méta-modèle cible. La transformation de modèles permet de réaliser une génération (semi-) automatique du code à partir des modèles.

En réponse au défi d'évolutivité (la nécessité de grands (méta) modèles et de transformations entre ces modèles), l'académie et l'industrie ont investi dans le soutien de cet outil. Par conséquent, un certain niveau de maturité a été réalisé pour la (méta) modélisation et le développement de transformation de modèles. L'étape suivante était l'automatisation des tâches de développement de transformation de modèles. Cette automatisation est un réel défi dans le domaine de l'ingénierie des modèles (les transformations manuelles introduisent beaucoup d'erreurs). L'automatisation de la transformation consiste à produire de nouveaux modèles à partir de modèles existants. La transformation n'est alors plus limitée à la production du code source mais peut opérer sur tous les modèles décrivant un système.

L'approche MDA (Model Driven Architecture ou architecture dirigée par les modèles) est un exemple d'application de l'IDM. La MDA est recommandée par l'OMG (Object Management Group). Elle est basée sur d'autres recommandations de ce même organisme. Elle spécifie trois niveaux d'abstractions exprimés en termes de modèles, les modèles des exigences CIM (Computation Independent Model), les modèles indépendants des plateformes PIM (Platform Independent Model), et les modèles spécifiques aux plateformes PSM (Platform Specific Model). Le système est ainsi représenté par différentes vues abstraites. Des transformations sont nécessaires entre les différents niveaux d'abstraction en vue de la génération du code. Plusieurs travaux sur les approches MDA et IDM se sont concentrés sur les niveaux inférieurs et les transformations automatiques

entre les uns et les autres. Ces travaux de recherches ont abordé la transformation des modèles de conception en modèles de déploiement et plus précisément des modèles PIM en modèles PSM et les modèles PSM vers du code dans MDA. Toutefois, moins d'attention a été accordée aux transformations de haut niveau d'abstraction en IDM, à savoir la transformation des modèles des exigences en modèles conceptuels, les transformations des modèles CIM en modèles PIM dans l'approche MDA.

1 Contributions

Considérant l'importance et l'utilité du modèle des exigences dans le développement des systèmes. Celui-ci peut être utilisé pour décrire le comportement global d'un système, à savoir un système distribué. Ceci est dû au fait que le comportement global de ces systèmes n'est pas réalisé par un seul composant mais par la collaboration d'un ensemble de composants. Un tel comportement peut être décomposé en un ensemble de comportements partiels réalisés par les différents composants du système. Par conséquent, une approche de transformation est nécessaire pour dériver automatiquement le comportement de ces composants à partir de la spécification du comportement global du système, modélisé par le modèle des exigences. Le processus de dérivation du comportement des composants ou entités du système est très important dans le développement des systèmes distribués tels que les systèmes d'information, les systèmes multi-agents et les applications distribuées. Un tel processus de dérivation automatique peut conduire à des systèmes valides et robustes en évitant les erreurs introduites par les activités de conception manuelle.

Les travaux présentés dans cette thèse ont pour objectif de contribuer à apporter une solution dans les domaines de la transformation des modèles des exigences et la dérivation du comportement d'un système et plus précisément un système distribué. Pour ce faire, nous proposons une approche de dérivation du comportement des différents composants d'un système basée sur la transformation de modèles. Nous considérons que le comportement global du système est modélisé par le modèle des exigences. Les éléments de base du comportement global du système sont des activités qui sont en fait des collaborations entre les composants ou les rôles du système. Les collaborations sont très appropriées pour modéliser les exigences car elles fournissent un cadre structurel pour ces

exigences qui peut incarner à la fois les comportements des rôles et les interactions entre ces rôles nécessaires pour accomplir un service.

L'approche proposée consiste à dériver le comportement des rôles du système par la transformation du modèle des exigences, qui décrit le comportement fonctionnel d'un système d'une manière abstraite vers des modèles conceptuels où chaque rôle est identifié par son comportement local. Les comportements des rôles du système doivent être déterminés de sorte que de la collaboration de ces comportements résulte un comportement satisfaisant le modèle global initial d'expression des besoins. Le processus de transformation, régi par un ensemble de règles, permet la dérivation du comportement local d'un composant, sous la forme d'un automate d'états-transitions, à partir des exigences du système décrites par un diagramme d'activités UML étendu. Ce processus permettra aussi la dérivation des messages de coordination appropriés entre les comportements dérivés des différents composants du système. Cette approche commence par la définition du méta-modèle source des diagrammes d'activités UML étendus avec des collaborations, et le méta-modèle cible des automates d'états-transitions UML. L'approche proposée considère les étapes suivantes :

- Définition du méta-modèle des exigences pour spécifier le comportement global d'un système donné.
- Choix du méta-modèle cible afin de modéliser le comportement des rôles du système au niveau conceptuel qui reflète le comportement local de chaque rôle.
- Établissement des règles de transformations qui régissent la transformation au cours du processus de dérivation.

Les concepteurs de logiciels savent par expérience que, très probablement, tout système logiciel ne fonctionne pas bien après la première compilation réussie, ni la deuxième ni la troisième. Parfois, il faut un certain temps pour découvrir qu'un logiciel apparemment supposé correct peut échouer de façon subtile. Les petits défauts peuvent se cacher pendant des années et apparaître au moment le plus gênant lorsqu'ils sont les moins attendus. Cependant, il existe des cas dans lesquels ces défauts ne sont pas acceptables, tels que les systèmes de surveillance médicale ou les applications économiques critiques ; les défauts logiciels peuvent entraîner des pertes de vie ou causer des dommages économiques importants.

Une grande partie de notre monde est maintenant contrôlée par des applications logicielles, le défi le plus important consiste à trouver les moyens et les outils nécessaires afin de rendre ces applications plus fiables. La vérification de tels systèmes implique une validation de la conception. Elle vérifie si une conception d'un système donné satisfait les exigences globales de celui ci. (Si ce n'est pas le cas, alors il est souhaitable de le savoir au début du processus de conception !) Ces tâches, la vérification du système et la validation de la conception, peuvent être accomplies en utilisant plusieurs techniques.

La vérification basée sur les modèles est l'une des méthodes formelles les plus puissantes et est particulièrement bien adaptée à la vérification automatisée des systèmes à états finis. La vérification de modèles nécessite des algorithmes sophistiqués basés sur la théorie des automates et la logique pour vérifier les modèles, mais pas les programmes. Ces modèles représentent fidèlement le système, tout en restant suffisamment concis pour permettre de vérifier son exactitude.

L'approche de dérivation du comportement d'un système se concentre sur les premières phases de développement d'un système logiciel distribué, en particulier dans le cas de l'obtention d'une conception d'un système à partir de sa spécification des exigences globales. Les comportements dérivés du système s'exécutent dans un environnement distribué. Ainsi, le défi le plus important consiste à trouver les moyens nécessaires pour vérifier si ces comportements dérivés sont fiables. Cet objectif consiste à vérifier si le système dérivé en question se comporte tel que le système initial a été conçu. Toutes ces raisons ont motivé notre travail pour l'utilisation d'une approche dirigée par les modèles pour effectuer la vérification du comportement des composants dérivés du système.

Dans cette thèse, nous présentons une approche dirigée par les modèles pour vérifier et valider le processus de dérivation. Il s'agit de vérifier si la collaboration des comportements dérivés satisfait la spécification globale du système initial. L'approche décrit un processus de vérification qui permet de vérifier et d'analyser la cohérence logique du système dérivé. Ce processus de vérification est principalement basé sur la transformation automatique des modèles dérivés vers des modèles spécifiques requis par un vérificateur de modèles. Par la suite, la vérification de modèles peut être effectuée pour analyser les comportements dérivés du système et vérifier donc si ce comportement est conforme à la spécification globale du système.

La conception et le développement des systèmes pour la surveillance médicale (health monitoring) utilisant les réseaux de capteurs (Wireless Body Sensor Network (WBSN)) est une tâche très complexe. La majorité des études dans ce domaine se sont axées sur les problèmes de mise en œuvre, et elles abordent rarement une méthodologie pour la construction et la maintenance de tels systèmes. La plupart de ces systèmes sont développés en sélectionnant d'abord la plateforme cible la plus appropriée et, ensuite, son système d'exploitation spécifique et son langage de programmation. L'existence d'une approche de conception explicite devrait être très bénéfique pour la construction et la maintenance de tels systèmes.

Le travail présenté dans cette thèse contribue également à la conception et la validation des applications de surveillance médicale basées sur les WBSN. Nous proposons d'abord une architecture composée de nœuds hétérogènes pour construire un système de surveillance médicale. Le système de surveillance médicale proposé est conçu pour effectuer une surveillance quotidienne et continue ainsi que des contrôles spécifiques pour les patients d'un hôpital. C'est une architecture WBSN basée sur un collecteur de données mobile. Le collecteur mobile est utilisé pour économiser les ressources (consommation d'énergie) des capteurs. Les nœuds capteurs ne peuvent être réveillés que lorsqu'ils sont en présence du collecteur de données. De plus, l'approche de dérivation du comportement dirigée par les modèles est utilisée pour obtenir le comportement de chaque nœud dans le WBSN à partir du comportement global du système WBSN. L'utilisation d'une approche d'ingénierie logicielle pour prendre en charge un tel processus de dérivation peut conduire à une conception hautement indépendante des plateformes d'implémentation. En outre, elle favorise la réutilisation des modules logiciels ; un modèle de comportement dérivé peut être reconsidéré sur différentes plateformes. L'approche de vérification dirigée par les modèles est adoptée pour vérifier les comportements dérivés et valider le processus de dérivation.

2 Organisation du document

La suite de ce document est constitué de sept chapitres, organisés comme suit :

Chapitre 2 : ce chapitre situe le cadre général de la thèse. Nous distinguons au niveau de ce chapitre la notion du paradigme de l'ingénierie dirigée par les modèles (IDM) et les techniques de transfor-

mation dans l'IDM. La première partie de ce chapitre introduit l'ingénierie dirigée par les modèles et ses concepts. Au niveau de la deuxième partie, nous décrirons l'approche MDA et ses principes. A la fin de ce chapitre, les techniques de transformations de modèles sont décrites et notamment les transformations de haut niveau, à savoir les transformations des modèles des exigences (CIM) vers des modèles conceptuels (PIM). Cette description nous a permis de mettre en évidence la transformation des modèles des exigences pour réaliser la dérivation du comportement d'un système distribué.

Chapitre 3 : la première partie de ce chapitre sera consacrée à une description détaillée de la problématique soulevée pour les transformations de modèles. Par la suite, nous décrirons l'approche proposée pour la transformation du modèle des exigences en modèles conceptuels. L'approche proposée consiste à dériver le comportement des rôles du système par transformation du modèle des exigences, qui décrit le comportement fonctionnel d'un système d'une manière abstraite vers des modèles conceptuels où chaque rôle est identifié par son comportement local.

Chapitre 4 : la première partie de ce chapitre sera consacrée à la description des langages de transformation de modèles, en particulier les langages QVT (Query View Transformation) et ATL (Atlas Transformation Language). La deuxième partie sera consacrée à la mise en œuvre de l'approche de dérivation proposée pour la transformation du modèle des exigences en modèles conceptuels.

Chapitre 5 : ce chapitre présente une approche dirigée par les modèles pour vérifier et valider le processus de dérivation décrit dans le chapitre 3. Ce processus de vérification a pour but de vérifier que la collaboration des comportements dérivés satisfait la spécification globale du système initial. La première partie du chapitre est consacrée à la description du processus de vérification dirigé par les modèles qui est principalement basé sur la transformation automatique des modèles dérivés vers des modèles spécifiques requis par un vérificateur de modèles. La deuxième partie présente une vérification du système distribué de télédiagnostic en neuroscience décrit dans le chapitre 3. Cette vérification de modèles est effectuée pour analyser et vérifier le comportement dérivé du système.

Chapitre 6 : ce chapitre présente un système de surveillance médicale préventive. Ce système est composé d'un réseau de capteurs corporels sans fil (Wireless Body Sensor Network (WBSN)) attachés au corps humain afin de permettre la surveillance des paramètres vitaux du corps humain et son environnement. Ce système est basé sur une architecture dotée de nœuds hétérogènes et réalise

à la fois une surveillance quotidienne et continue ainsi que des contrôles spécifiques. Nous avons défini un modèle pour décrire le comportement global du système WBSN. L'approche de dérivation du comportement dirigée par les modèles est adoptée pour obtenir le comportement de chaque nœud dans le WBSN à partir du comportement global du système WBSN. Les comportements dérivés du système WBSN sont vérifiés par l'approche proposée de vérification dirigée par les modèles. Cette vérification consiste à déterminer si la collaboration de ces comportements satisfait la spécification globale du système initial.

Enfin, le chapitre 7 conclut cette thèse en présentant les contributions apportées durant cette thèse, et présente les perspectives ouvertes à l'issue de nos travaux.

Chapitre 2

Ingénierie dirigée par les modèles et transformations

1 Introduction

Suite à l'approche objet des années 80 et de son principe de considérer que "tout est objet", l'ingénierie du logiciel s'oriente aujourd'hui vers l'ingénierie dirigée par les modèles (IDM) et le principe du "tout est modèle". L'ingénierie des modèles peut être vu comme une généralisation de la technologie orientée objet. Les principaux concepts de la technologie objet sont les classes, les instances et les deux relations associées *InstanceOf* et *InheritsFrom*. Pour l'IDM, le concept classe est remplacé par le concept modèle. Un modèle représente une vue d'un système défini dans le langage de son méta-modèle. En d'autres termes, un modèle contient des éléments conformes aux concepts et aux relations exprimés dans son méta-modèle. Les deux relations de base sont *RepresentedBy* et *ConformsTo*. Un modèle représente un système et il est conforme à un méta-modèle. Les méta-modèles sont à leur tour conforme à un méta-méta-modèle qui est défini en termes de lui-même [12]. En ingénierie des modèles, les modèles ne sont pas seulement des moyens de communication entre les développeurs d'applications mais ils sont suffisamment précis pour être utilisés pour générer du code. Les concepts et les éléments peuvent correspondre à des classes et des instances, respectivement. A première vue, cela suggère des ressemblances entre la techno-

logie objet et l'IDM, mais il existe une nette différence entre les deux. Il faut clairement séparer l'approche orientée objet de l'approche orientée modèle (IDM). Cependant, dans bien des cas la confusion est faite entre les concepts que l'on trouve dans l'IDM et leur incarnation dans le monde objet. Par exemple, la vision selon laquelle un modèle serait une "instance d'un" méta-modèle.

L'objectif de l'IDM est de définir une approche pouvant intégrer différents espaces technologiques [58]. Par exemple, l'espace technologique des grammaires, des documents structurés XML, ou des bases de données relationnelles, ne sont pas fondés sur le concept d'objet. Les deux relations de base *InstanceOf* et *InheritsFrom* ne sont pas adaptées aux autres espaces technologiques comme elles le sont pour le monde des objets [14]. L'IDM vise à fournir un grand nombre de modèles pour exprimer séparément chacune des préoccupations des utilisateurs, des concepteurs, des architectes, etc. C'est par ce principe de base, fondamentalement différent, que l'IDM peut être considérée en rupture par rapport aux travaux de l'approche objet. Ainsi, il est nécessaire de définir quels sont les concepts et les relations essentielles à l'IDM.

2 Concepts essentiels de l'IDM

L'approche orientée objet est basée sur deux relations fondamentales : la relation *InstanceOf* qui permet d'introduire la notion de classe et la relation *InheritsFrom* qui permet d'introduire la notion de superclasse. Le concept central de l'IDM est la notion de modèle et les deux relations fondamentales sont *RepresentedBy* et *ConformsTo*. La première relation, *RepresentedBy*, est liée à la notion de modèle, alors que la relation *ConformsTo* permet de définir la notion de modèle par rapport à celle de méta-modèle.

2.1 Modèles

Bien qu'il n'existe pas une définition universelle du modèle, néanmoins de nombreux travaux montrent cependant un relatif consensus sur une certaine compréhension. Les trois définitions ci-dessous sont représentatives de cette tendance.

Définition : *"A **model** is an abstraction of **a physical system**, with a certain purpose."*

Elle provient du standard UML [75]. Un modèle est une abstraction d'un système physique avec un certain objectif. Cette définition est contraignante car elle se limite à la représentation des concepts physiques et exclue les concepts abstraits tels que la notion de session de cours ou d'interaction dans de tels modèles.

Définition : *"A **model** is a set of statements about **some system under study**."* [100]

Un modèle est un ensemble d'énoncés d'un système à l'étude.

Définition : *"A **model** is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual **system**."* [13]

Un modèle est une abstraction d'un système, modélisé sous la forme d'un ensemble de faits construits dans une intention particulière. Un modèle doit pouvoir être utilisé pour répondre à des questions sur le système modélisé. Toutefois, un modèle doit, par définition, être une abstraction pertinente du système qu'il modélise, i.e., qu'il doit être suffisant et nécessaire pour permettre de répondre à certaines questions en lieu et place du système qu'il représente.

Ces définitions provenant de sources différentes suggèrent qu'il existe une relation entre le *modèle* et le *système étudié* qu'il modélise. Cette relation est appelée *RepresentedBy* dans [8, 14, 100]. Notons que la nature exacte des systèmes considérés n'est pas explicitée dans les définitions précédentes. Ces définitions ne sont pas restreintes aux modèles et systèmes informatiques. Par exemple, une carte géographique peut jouer le rôle de modèle, alors que la région étudiée jouera celui du système modélisé. Dans cet exemple, une carte est un modèle (une représentation) de la réalité, avec une intention particulière (carte routière, administrative, des reliefs, etc.).

La notion de modèle dans l'IDM fait explicitement référence à la notion de langage bien défini dans lequel est exprimé le modèle. En effet, pour qu'un modèle soit productif, il faut qu'il puisse être manipulé par une machine. Donc, le langage dans lequel ce modèle est exprimé doit être clairement défini. La définition d'un langage de modélisation a pris la forme d'un modèle, appelé méta-modèle.

2.2 Méta-modèles

La notion de méta-modèle est au cœur même de l'ingénierie dirigée par les modèles (IDM). Un méta-modèle est définit dans la littérature par :*"A meta-model is a model that defines the language for expressing a model".* [80]

Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle, i.e., le langage de modélisation. La notion de méta-modèle conduit à l'identification d'une seconde relation, reliant le modèle et le langage utilisé pour le construire, appelée *ConformTo* [14, 31].

L'originalité de l'IDM n'est pas de mettre l'accent sur la relation *RepresentedBy* liant un modèle au système modélisé. Cette relation est attribuée aux méthodes de modélisation. Par contre, le point clé dans l'IDM est de mettre l'accent sur l'importance de la relation *ConformTo* liant un modèle au méta-modèle auquel il est conforme. Cette relation permet d'assurer qu'un modèle est correctement construit permettant ainsi à des transformations automatisées d'être envisagées.

La relation de conformité peut avoir plusieurs incarnations selon l'espace technologique considéré. Par exemple, dans l'espace technologique XML, on dira qu'un document XML est conforme à un schéma ou à une DTD, dans l'espace technologique des bases de données on dira que le contenu d'une base de données est conforme au schéma de cette base de données, dans l'espace technologique orienté-objet on dira qu'un objet est conforme à sa classe et dans l'espace technologique orienté modèle on dira qu'un modèle est conforme à son méta-modèle. La relation *InstanceOf* ressemble à l'incarnation de la notion de conformité dans le cadre orienté objet mais ce n'est là qu'un cas particulier. Plusieurs variantes de la relation de conformité peuvent être considérées pour l'Ingénierie Dirigée par les Modèles. Les méta-modèles deviennent des entités de première classe. Ils sont définis comme des langages spécifiques de domaine (DSL) pour adresser les spécificités liées à tels secteurs d'activités, à tel métier, à tel point de vue, à tel niveau d'abstraction, ou à telles plateformes technologiques.

Le consensus sur UML fut décisif dans la transition vers des techniques de production basées sur les modèles. L'acceptation du concept méta-modèle comme langage de description de modèle a permis l'émergence de nombreux méta-modèles afin d'apporter chacun leurs spécificités dans un domaine particulier (développement logiciel, entrepôt de données, procédé de développement,

etc.). Cette grande diversité de méta-modèles indépendants et incompatibles induisait un besoin urgent de donner un cadre général pour leur description. La réponse fut donc d'offrir un langage de définition de méta-modèles qui prit lui-même la forme d'un modèle ; ce fut le méta-méta-modèle MOF (Meta-Object Facility) proposé par l'OMG [80]. L'OMG met surtout en avant actuellement une architecture dont le socle est le MOF, sur lequel s'appuie d'une part une collection de méta-modèles dont UML n'est qu'un élément parmi d'autres. Par la suite, l'approche MDA (Model Driven Architecture) [110] est devenue une variante particulière de l'Ingénierie Dirigée par les Modèles.

3 L'approche MDA (Model Driven Architecture)

L'évolution des systèmes informatiques a rendu difficile la maintenance des logiciels à moindre coût et ceci dans des délais raisonnables. Cette évolution a induit le redéveloppement des logiciels sur de nouvelles plateformes, alors que globalement la logique métier de l'entreprise n'a pas changé. En effet, du fait de l'évolution incessante et inévitable ; les technologies se sont succédées. Par exemple, CORBA (Common Object Request Broker Architecture) qui devait permettre une communication entre applications différentes ne s'est pas imposé en tant que standard du Middleware. Cela s'explique par sa complexité et l'apparition d'architectures concurrentes plus simples comme EJB (Entreprise Java Bean). Ainsi, les architectures se suivent et ne se ressemblent pas. Ces bouleversements ont aussi été pris en compte au niveau des spécifications des logiciels, avec l'apparition de nouvelles méthodes d'analyse et de conception à base de modèles et à production de code. La plus célèbre de ces méthodes de modélisation de systèmes à objets est UML (Unified Modeling Language) de l'OMG [17]. Avec ce langage graphique, les différentes dimensions d'un logiciel complexe peuvent être spécifiées, assemblées, visualisées et documentées. Ces différentes modélisations sont ensuite utilisées pour générer automatiquement la trame de code du logiciel cible.

Pour permettre l'interopérabilité entre les différents systèmes, réduire les coûts de développement et augmenter l'évolutivité, l'OMG (Object Management Group) propose la démarche MDA (Model Driven Architecture), où la notion de modèle devient essentielle et centrale. MDA est une

nouvelle approche d'écriture de spécifications et de développement d'applications. Cette approche prône l'utilisation de modèles indépendants de toute plateforme et technologie. Cette approche par abstraction permet de mieux se concentrer sur la partie "intelligente" et la rend pérenne car elle reste utilisable avec les technologies futures.

3.1 Principes du MDA

L'OMG (Object Management Group) propose de revoir la façon de penser une application. Cette révolution s'appuie sur des standards éprouvés, regroupés au sein du MDA (Model Driven Architecture). Cette démarche apporte un changement important dans la conception des applications. Elle introduit une séparation nette entre la logique métier de l'entreprise et la logique d'implémentation. Cette révolution est d'autant plus importante qu'elle change non seulement la façon de voir mais aussi notre façon de faire. En effet, elle propose de mettre à disposition des développeurs des outils d'automatisation et de génération de code à partir de la logique métier.

L'approche MDA [72] fournit des outils pour :

- Spécifier (modéliser) un système indépendamment de la plateforme d'implémentation ;
- Spécifier les plateformes ;
- Choisir une plateforme particulière pour le système ; et
- Transformer la spécification du système pour l'implémenter sur une plateforme particulière.

Les trois objectifs de l'approche MDA sont la portabilité, l'interopérabilité et la réutilisation au moyen de la séparation entre la fonctionnalité d'une application et l'architecture où elle sera implémentée.

A terme, la démarche MDA propose de définir un modèle métier indépendant de toute plateforme technique et de générer automatiquement du code vers la plateforme choisie. Pour cela, l'accent est mis non plus sur les approches orientées objet mais sur les approches orientées modèle. Le but est de favoriser l'élaboration de modèles de plus haut niveau.

Les avantages recherchés par cette approche sont :

- La pérennité du savoir faire. Ainsi il n'est pas nécessaire de recommencer la modélisation des fonctionnalités et des comportement du système chaque fois qu'une nouvelle technologie est adoptée ;
- Les gains de productivité, afin de permettre au développeur de réduire les coûts de mise en œuvre des applications nécessaires à son métier ;
- La prise en compte des plateformes d'exécution, permettant ainsi le déploiement d'une même application sur plusieurs plateformes.

3.2 Architecture MDA

L'OMG définit MDA comme une approche pour la spécification des systèmes informatiques qui sépare les fonctionnalités du système de ses détails d'implémentation sur une plateforme technologique particulière. La MDA est un cadre pour le développement d'applications dirigé par les modèles [72].

Les projets utilisant MDA se concentrent d'abord sur la fonctionnalité et le comportement d'une application ou d'un système, mettant à l'écart les technologies dans lesquelles ils seront mis en œuvre. Cette approche met l'accent sur les modèles qui fournissent un niveau plus élevé d'abstraction au cours du développement. Les applications créées à l'aide de MDA peuvent varier du transport, à l'industrie, à la santé et peuvent être déployées sur une gamme de marchés ouverts et plateformes propriétaires, tels que CORBA, J2EE, .NET et etc. (voir Fig. 2.1). L'architecture MDA se compose de quatre couches qui réfèrent à chaque niveau des standards déjà adoptés par l'industrie ou en cours de normalisation.

Le centre est composé par trois technologies spécifiées par l'OMG pour modéliser la logique métier de l'application : UML (Unified Modeling Language), MOF (Meta Object Facility) et CWM (Common Warehouse Meta-model). Dans la couche suivante, se trouve aussi le standard XMI (XML Meta-data Interchange) qui permet le dialogue entre les middlewares (Java, CORBA,

.NET et web services). La troisième couche contient les services qui permettent de gérer les événements, la sécurité, les répertoires et les transactions. Enfin, la dernière couche propose des frameworks spécifiques au domaine d'application (Finance, Télécommunication, Transport, Espace, Médecine, Commerce électronique, Manufacture,...) [70].

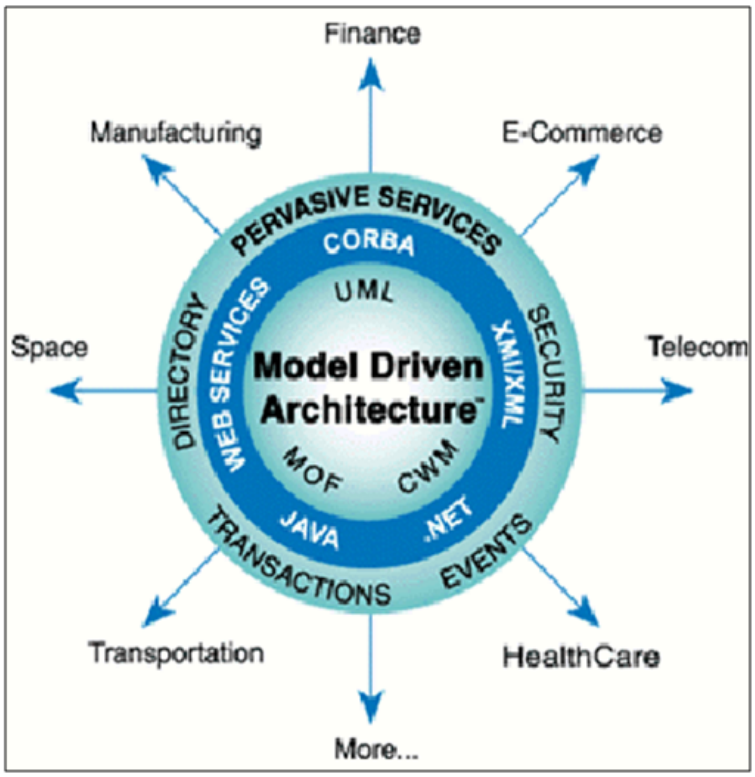

Fig. 2.1 : Architecture du MDA. [70]

3.2.1 Spécifications standards du MDA

Les principales normes qui composent la MDA [70] comprennent l'Unified Modeling Language (UML) [74], Meta-Object Facility (MOF) [80], XML Meta-data Interchange XMI [77] et Metamodel Common Warehouse (MCG).

3.2.1.1 ***Meta-Object Facility (MOF) :*** Le langage MOF fournit le standard de méta-modélisation et d'échange de constructions utilisées par MDA. Il est utilisé pour définir, manipuler et intégrer

les méta-données et les données d'une manière indépendante de la plateforme. Les autres modèles standards de l'OMG, comme UML et CWM, sont définis par des constructions MOF, ce qui permet de les relier entre eux. C'est également le mécanisme par lequel les modèles sont sérialisés en XMI. Le MOF est un exemple de méta-méta-modèle, ou de modèle du méta-modèle. Il définit les éléments essentiels, la syntaxe et la structure des méta-modèles utilisés pour construire des modèles orientés objet. L'intérêt de MOF est de définir des mécanismes génériques sur les méta-modèles grâce à une structuration commune. Il permet de faire inter opérer des méta-modèles différents. Une application MOF peut manipuler un modèle à l'aide d'opérations génériques sans connaissances du domaine.

3.2.1.2 ***UML (Unified Modeling Langage) :*** Le langage UML (Unified Modeling Language) est une spécification, définissant un langage graphique pour visualiser, spécifier, construire et documenter les artefacts de systèmes logiciels. Il a été adopté par l'OMG comme standard de modélisation des systèmes informatiques en novembre 1997. Il permet la modélisation d'architectures, de structure d'objets, d'interactions entre ces objets, de données, de composants, de frameworks, etc. Les concepts définis par UML sont très proches de ceux définis par le MOF. Ainsi, le MOF utilise les représentations graphiques de UML. UML a apporté au domaine de la méta-modélisation sa notation graphique et ses concepts objet. Le langage UML permet la modélisation de systèmes indépendamment de toute démarche ou de plateforme. C'est pourquoi, dans la cadre du MDA, UML peut être utilisé pour décrire des plateformes, des organisations, des situations ou la plupart des systèmes logiciels.

Depuis sa version 1.1, UML intègre le langage OCL (Object Constraint Language) qui est un langage d'expression permettant de décrire des contraintes sur des modèles objet. Une contrainte est une restriction sur une ou plusieurs valeurs d'un modèle non représentable en UML. OCL est un langage sans effet de bord, il est incapable de modifier quoi que ce soit au sein des objets ou même des variables. Il donne des descriptions précises et non ambigües du comportement du logiciel en complétant les diagrammes. Il définit des pré-conditions, des post-conditions ou des invariants pour une opération. Il permet aussi la définition des expressions de navigation ou des expressions booléennes. OCL est un langage de haut niveau d'abstraction parfaitement intégré à UML qui permet de trouver des erreurs beaucoup plus tôt dans le cycle de vie d'une application.

Cette vérification d'erreurs est rendue possible grâce à la simulation du comportement du modèle.

Avec l'apparition de la version 2.0, UML contient entre autres une évolution du langage d'expression de contrainte OCL, des concepts renforcés pour les composants et s'intègre plus facilement dans l'approche MDA grâce à un rapprochement avec le MOF. Le concept de profil UML permet d'adapter le langage UML à un domaine qu'il ne pouvait pas couvrir correctement. Les profils sont utilisés pour la génération de modèles. Les spécificités de chaque plateforme peuvent être modélisées grâce aux mécanismes d'extension d'UML définis par les profils UML. Par exemple, les stéréotypes permettent l'ajout de nouveaux éléments au méta-modèle, les valeurs marquées (tagged values) permettent l'ajout de propriétés à une méta-classe et les contraintes permettent l'ajout ou la modification de règles. Il est possible d'associer les stéréotypes, les valeurs marquées et les contraintes à tout concept UML (classe, attribut, association, cas d'utilisation). Ces éléments permettent d'établir une correspondance entre les concepts UML et les concepts du domaine représentés par le profil. Le profil est aussi composé d'un ensemble de règles de présentation, de conception ou de transformation. Ce sont ces règles qui rendent le profil opérationnel.

3.2.1.3 ***XMI (XML Meta-data Interchange) :*** Le langage XMI est le langage d'échange entre le monde des modèles et le monde XML (eXtensible Markup Language). C'est le format d'échange standard entre les outils compatibles MDA. XMI décrit comment utiliser les balises XML pour représenter un modèle UML en XML. Cette représentation facilite les échanges de données (ou méta-données) entre les différents outils ou plateformes de modélisation. En effet, XMI définit des règles permettant de construire des DTD (Document Type Definition) et des schémas XML à partir de méta-modèles, et inversement. Ainsi, il est possible d'encoder un méta-modèle dans un document XML, mais aussi, à partir de document XML il est possible de reconstruire des méta-modèles. Le langage XMI a l'avantage de regrouper les méta-données et les instances dans un même document ce qui permet à une application de comprendre les instances grâce à leurs méta-données. Ceci facilite les échanges entre les applications.

3.2.1.4 ***Le méta-modèle Common Warehouse :*** Le Common Warehouse Meta-model ou CWM est une spécification de l'OMG. Il est composé d'un ensemble d'interfaces standards qui peuvent être utilisées pour permettre l'échange facile de méta-données d'un entrepôt et des applications

intelligentes avec les outils d'entrepôt, les plateformes et les entrepôts de méta-données dans les environnements distribués et hétérogènes. CWM est le standard pour l'intégration des entrepôts de données. Il standardise la façon de représenter les schémas, les modèles de transformation de schémas et les modèles de datamining.

3.2.2 La méta-modélisation

MDA ne considère plus le code exécutable comme le référentiel unique dans le processus de développement du logiciel. Les différents aspects élaborés par les développeurs vont se concrétiser par des modèles lisibles par les opérateurs humains mais surtout interprétables par les ordinateurs. A un aspect spécifique donné correspondra un modèle particulier. Chacun de ces modèles est exprimé dans un langage spécifié par un méta-modèle. Pour harmoniser ces méta-modèles en un langage unique, l'OMG a défini le MOF (Meta-Object Facilty).

3.2.2.1 ***Méta-modélisation :*** La méta-modélisation est l'activité consistant à définir le méta-modèle d'un langage de modélisation. Elle vise donc à bien modéliser un langage, qui joue alors le rôle de système à modéliser.

3.2.2.2 ***Méta-méta-modèle :*** Un méta-méta-modèle est un modèle qui décrit un langage de méta-modélisation, i.e., les éléments de modélisation nécessaires à la définition des langages de modélisation. Il a de plus la capacité de se décrire lui-même.

MDA doit donc manipuler des modèles de natures très diverses (modèles d'objet métier, modèles de processus, modèles de règles, modèles de service, modèles de plateforme, etc.). Pour structurer et organiser tous ces modèles, l'OMG a défini une "architecture à quatre niveaux" décrite sous une forme de pyramide en se basant sur le principe de la méta-méta-modélisation (Fig. 2.2) :

- Le niveau M0 (ou instance) correspond au monde réel. Ce sont les informations réelles de l'utilisateur, instance du modèle de M1.

- Le niveau M1 (ou modèle) est composé de modèles d'information. Il décrit les informations de M0. Les modèles M1 sont conformes au méta-modèle de M2.

- Le niveau M2 (ou méta-modèle), il définit le langage de modélisation et la grammaire de représentation des modèles M1. Le méta-modèle UML, qui définit la structure interne des modèles UML, fait partie de ce niveau. Les profils UML, qui étendent le méta-modèle UML, appartiennent aussi à ce niveau. Les méta-modèles sont conformes au MOF.

- Le niveau M3 (ou méta-méta-modèle) est composé d'une unique entité qui s'appelle le MOF. Le MOF permet de décrire la structure des méta-modèles, d'étendre ou de modifier les méta-modèles existants. Le MOF est réflexif, il se décrit lui-même.

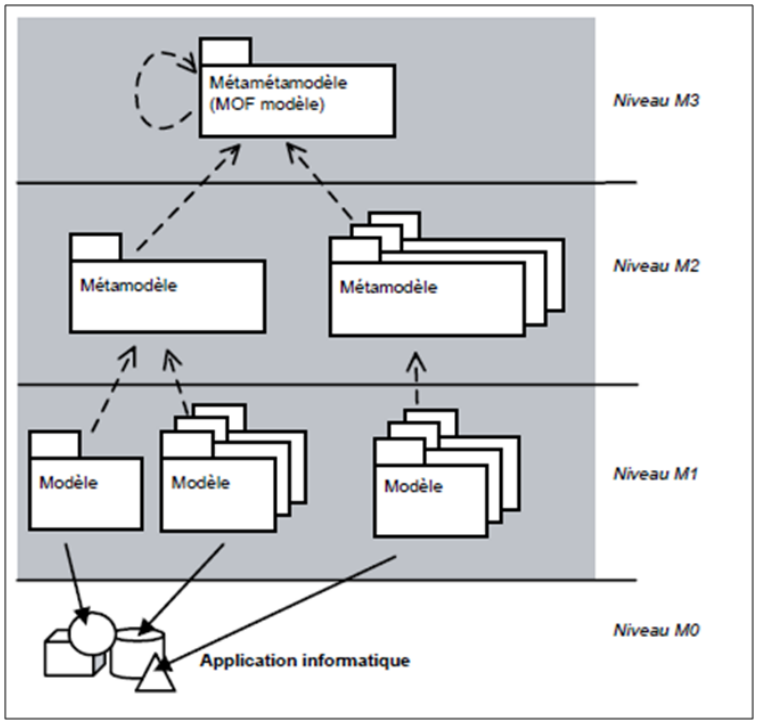

Fig. 2.2 : Architecture MDA à quatre niveaux.

3.3 Processus de développement du MDA

3.3.1 Les différents modèles du MDA

Le principe clé du MDA consiste à décrire séparément des modèles pour les différentes phases du cycle de développement d'une application. L'approche MDA préconise l'élaboration de plusieurs modèles, "descriptions abstraites d'une entité du monde réel en utilisant un formalisme donné" qui vont servir dans un premier temps à modéliser l'application, puis par transformations successives à générer du code. L'approche MDA est fondée sur plusieurs niveaux de modèles, principalement les modèles des exigences (CIM), les modèles indépendants des plateformes (PIM) et les modèles spécifiques aux plateformes (PSM). Dans cette section, nous allons donner une description des différents modèles du MDA.

3.3.1.1 ***Le CIM (Computation Independent Model) :*** Les exigences ou les besoins du système sont modélisés dans le modèle CIM, il est indépendant de tout système informatique [72]. Il doit représenter le système dans son environnement afin de définir quels sont les services offerts par le système et quelles sont les autres entités avec lesquelles il interagit. Le modèle des exigences ne contient pas d'information sur la structure du système, la réalisation de l'application ni sur les traitements. C'est le modèle métier ou le modèle du domaine d'application. Il est utile, non seulement comme aide pour comprendre un problème, mais également comme source de vocabulaire partagé avec d'autres modèles. La création d'un modèle des exigences est d'une importance capitale. Cela permet d'exprimer clairement les liens de traçabilité avec les modèles qui seront construits dans les autres phases du cycle de développement de l'application, comme les modèles d'analyse et de conception. L'indépendance technique de ce modèle lui permet de garder tout son intérêt au cours du temps et il est modifié uniquement si les connaissances ou les besoins métier changent.

3.3.1.2 ***Le PIM (Platform Independent Model) :*** Le modèle indépendant de la plateforme est un modèle avec un haut niveau d'abstraction qui est indépendant de toute technologie de mise en œuvre [56]. Il ne contient pas d'informations sur les technologies qui seront utilisées pour déployer le système. Un langage de modélisation capable de générer tous les artefacts requis, tels que l'Uni-

fied Modeling Language est nécessaire à ce niveau. Le PIM peut être élaboré en utilisant d'autres notations qu'UML. Le modèle PIM couvre les phases d'analyse et de conception du cycle de développement. Le PIM représente la logique métier spécifique au système ou le modèle de conception. Il représente le fonctionnement des entités et des services. Il doit être pérenne, durer au cours du temps et faire le lien entre le modèle des exigences et le code de l'application. Les modèles PIM doivent par ailleurs être productifs puisqu'ils constituent le socle de tout le processus de génération de code défini par MDA. La productivité des PIM signifie qu'ils doivent être suffisamment précis et contenir suffisamment d'informations pour qu'une génération automatique de code soit envisageable. De plus, MDA ne donne aucune indication sur la méthode utilisée pour l'élaboration des PIM.

3.3.1.3 ***Le PSM (Platform Specific Model) :*** MDA considère que le code d'une application peut être obtenu à partir de modèles de code (PSM). La différence principale entre un modèle de code et un modèle d'analyse ou de conception réside dans le fait que le modèle de code est strictement lié à une plateforme d'exécution. Le modèle PSM est dépendant de la plateforme technique spécifiée par l'architecte. Le PSM sert essentiellement de base à la génération de code exécutable vers la ou les plateformes techniques. Il existe plusieurs niveaux de PSM. Le premier, issu de la transformation d'un PIM, se représente par un schéma UML spécifique à une plateforme. Les autres PSM sont obtenus par transformations successives jusqu'à l'obtention du code dans un langage spécifique (Java, C++, C#, etc.) [72]. Le rôle des modèles de code est principalement de faciliter la génération de code. Ils sont donc essentiellement productifs mais ne sont pas forcément pérennes.

3.3.1.4 ***Le PDM (Platform Model) :*** Un PDM contient des informations pour la transformation de modèles vers une plateforme en particulier et il est spécifique à celle-ci. C'est un modèle de transformation qui va permettre le passage du PIM vers le PSM. Le PDM décrit le modèle de la plateforme qui fournit un ensemble de concepts techniques sur la fonctionnalité et l'utilisation de la plateforme. Généralement, ce modèle est sous forme de manuels logiciel et matériel. Le PDM est ainsi basé sur le détail des modèles de la plateforme qui peuvent être exprimés en UML et/ou OCL.

3.3.2 Architecture générale de l'approche MDA

La figure (Fig. 2.3) donne une vue générale de l'approche MDA. La construction d'une nouvelle application débute par l'élaboration d'un ou de plusieurs modèles des exigences (CIM). Elle se poursuit par l'élaboration des modèles d'analyse et de conception abstraite de l'application (PIM). Les modèles PIM doivent être partiellement générés à partir des modèles CIM afin que des liens de traçabilité soient établis. Ils sont des modèles pérennes, qui ne contiennent aucune information sur les plateformes d'exécution.

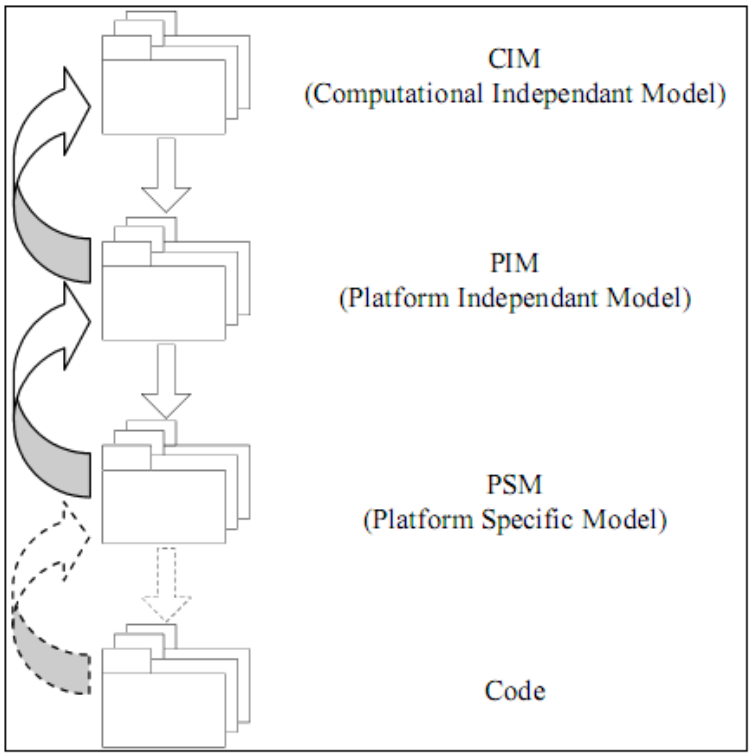

Fig. 2.3 : Architecture générale de l'approche MDA.

La phase suivante consiste à construire des modèles spécifiques aux plateformes d'exécution. Ces modèles sont obtenus par une transformation des PIM en y ajoutant les informations techniques relatives aux plateformes. Les PSM n'ont pas pour vocation d'être pérennes. Leur principale fonction est de faciliter la génération de code. Le passage entre les différents modèles du MDA est réalisé par une suite de transformations. La notion de transformation est un autre concept central

pour MDA. Décrire comment transformer un modèle est une condition indispensable pour le rendre productif. L'OMG propose à terme l'automatisation de ces transformations par des outils logiciels. Il sera alors possible de passer du modèle d'analyse au déploiement en générant automatiquement du code. Les transformations font néanmoins l'objet de recherches actives.

Les transformations de modèles préconisées par MDA sont essentiellement les transformations CIM vers PIM et PIM vers PSM. La génération de code à partir des modèles PSM n'est quant à elle pas considérée comme une transformation de modèles à part entière. MDA envisage aussi les transformations inverses : code vers PSM, PSM vers PIM et PIM vers CIM. Les transformations de modèles portent l'intelligence et le savoir-faire du processus méthodologique de construction d'applications. MDA préconise d'autre part la modélisation des transformations de modèles.

La figure (Fig. 2.4) présente un méta-modèle typique de MDA. Il utilise un ensemble de normes pour une description complète. Ce méta-modèle interagit directement avec les modèles PIM, les modèles PSM et leurs transformations. Le PIM est indépendant de toute infrastructure, mais le PSM dépend d'une structure particulière correspondant à une plateforme d'exécution. Les transformations étudiées dans ce paragraphe sont essentiellement des transformations de PIM vers PSM car ce sont ces transformations qui seront appelées à être utilisées couramment à savoir : Transformation de PIM vers PIM, Transformation de PIM vers PSM, Transformation de PSM vers PSM et Transformation de PSM vers PIM.

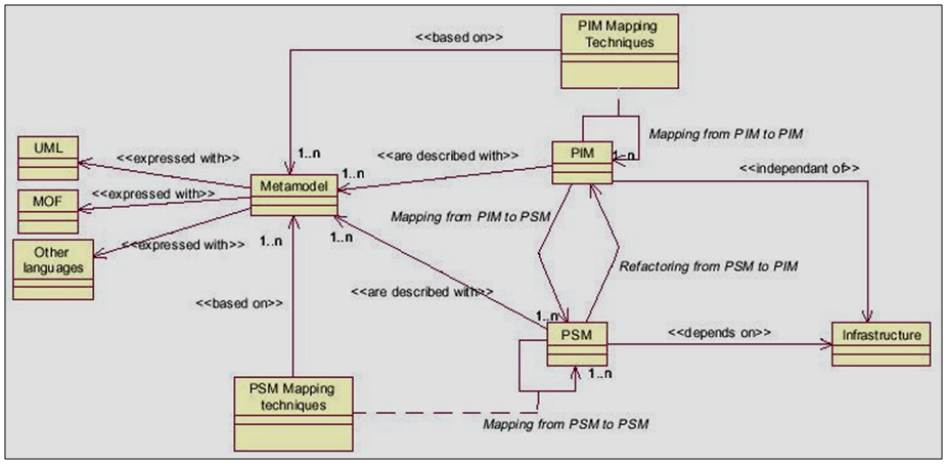

Fig. 2.4 : Le méta-modèle de MDA. [53]

3.3.2.1 ***Transformation de PIM vers PIM :*** Ces transformations visent à enrichir, filtrer ou spécialiser le modèle pendant le cycle de développement sans nécessiter d'informations dépendantes d'une plateforme. Les transformations PIM vers PIM sont généralement utilisées pour le raffinement du modèle.

3.3.2.2 ***Transformation de PIM vers PSM :*** Une fois le PIM est suffisamment raffiné pour pouvoir être spécialisé vers une plateforme donnée, il peut alors être transformé en PSM. Cette opération consiste à ajouter au PIM des informations propres à une plateforme technique. C'est le PDM qui contient les caractéristiques de la transformation. Il est alors possible de passer d'un modèle indépendant à un modèle dépendant d'une plateforme. Les règles de transformation devront être généralisées et capitalisées pour obtenir dans le futur une automatisation importante.

3.3.2.3 ***Transformation de PSM vers PSM :*** Ces transformations s'appliquent sur un modèle spécifique et donnent un autre modèle spécifique à la même plateforme. Elles sont utilisées pour la réalisation et le déploiement de composants. La génération de code, la compilation, la mise en paquetages, l'initialisation et la configuration font parties de ce type de transformation.

3.3.2.4 ***Transformation de PSM vers PIM :*** Ces transformations doivent permettre d'obtenir un modèle indépendant à partir d'une implantation existante sur une plateforme spécifique. Bien que celles-ci ne fassent pas directement partie du processus MDA, elles doivent permettre d'intégrer des applications existantes afin de pouvoir les utiliser dans un processus MDA. Ce sont certainement les transformations les plus difficiles à automatiser. Idéalement, le résultat de cette transformation devrait correspondre à la transformation inverse PIM vers PSM.

3.4 Les approches de transformation de modèles dans MDA

L'essentiel de la démarche MDA réside dans la capacité de passer d'un modèle à un autre. Les techniques de transformation de modèles sont donc au cœur du MDA. Ces transformations peuvent être manuelles, assistées par des outils ou entièrement automatiques.

3.4.1 La transformation par annotation ou marquage

Cette technique de transformation (Fig. 2.5) consiste à annoter (tagger) un PIM à l'aide d'annotations (tags) puis, convertir ce PIM en PSM à l'aide des règles de transformation issues des profils UML ou à l'aide des patrons de conception. Les annotations sont faites manuellement par l'utilisateur. Des traces de cette transformation sont conservées. En les utilisant, il est possible de revenir du PSM vers le PIM car une correspondance entre les éléments source et cible est maintenue. Le terme annotation peut-être remplacé par celui de marquage [72].

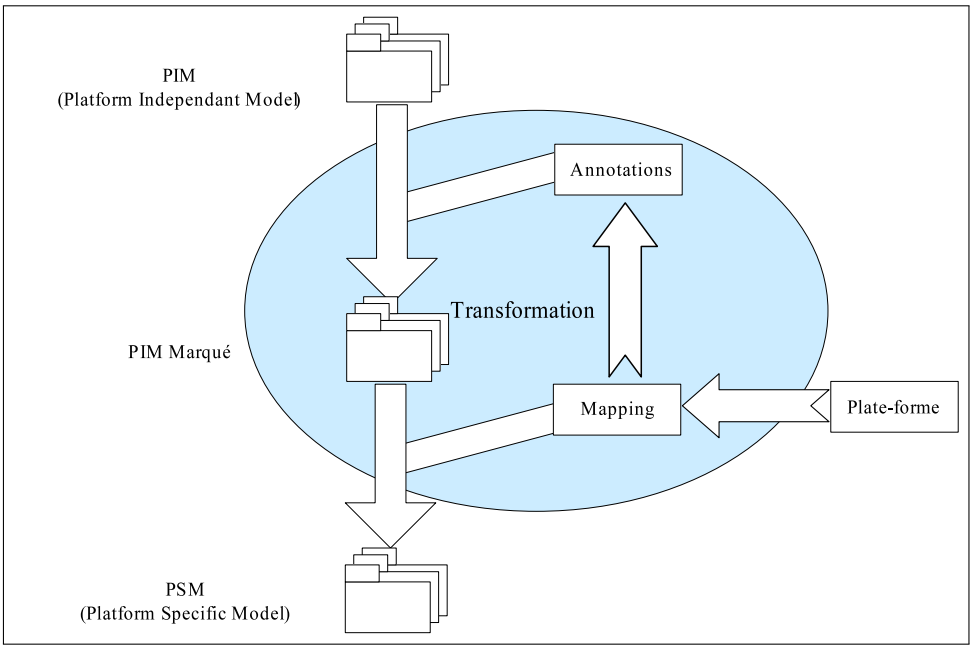

Fig. 2.5 : Transformation par annotation ou marquage.

3.4.2 La transformation par méta-modèles

Cette technique de transformation (Fig. 2.6) s'effectue grâce aux méta-modèles. Les profils UML sont exclus de cette transformation car les méta-modèles peuvent être exprimés dans d'autres langages qu'UML. Elle comporte deux étapes : la première consiste à spécifier les règles de transformation *Mt* décrivant la correspondance entre le langage source *MMa* et le langage cible *MMb*, la deuxième consiste à appliquer ces règles au modèle source *Ma* pour produire le modèle cible *Mb*.

Par exemple, le PIM peut être décrit en UML avec un méta-modèle UML et le PSM est spécifique à une plateforme particulière comme EJB. La transformation se base sur un langage source (ici UML) et sur un langage cible (ici Java).

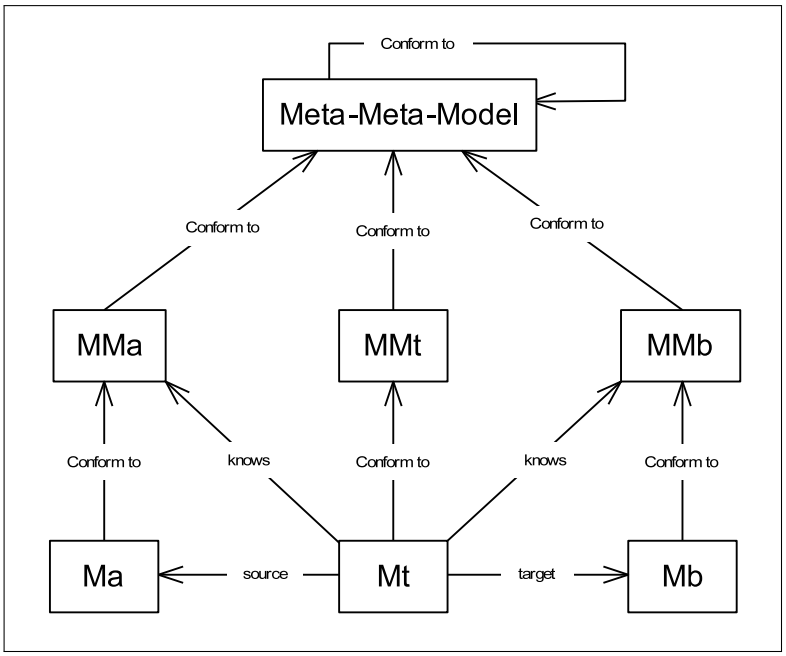

Fig. 2.6 : Transformation par méta-modèles.

3.5 L'approche en double Y

Cette approche synthétise les recherches qui se font sur la transformation des méta-modèles. L'approche en double Y découle de l'approche simple Y qui est devenue insuffisante car elle ne sépare pas les exigences non fonctionnelles des exigences fonctionnelles. Le cycle simple Y a seulement deux branches, l'une contient les spécifications fonctionnelles, l'autre les informations spécifiques à une plateforme.

L'approche double Y permet d'avoir une vue globale de toutes les étapes de la démarche MDA. Elle permet de conserver une indépendance vis à vis des plateformes durant la majorité de la phase de développement. Sur la Figure (Fig. 2.7) en partant d'en haut à gauche, nous avons le CIM ou PIM. Ce tout premier modèle est alors appelé PIM fonctionnel. Il peut être raffiné en de

nouveaux PIM si nécessaire. Sur la branche en face de celle du CIM, nous avons les exigences non fonctionnelles et la qualité de service (performance, sécurité, persistance, ...), qui sont indépendantes d'une plateforme et/ou d'une architecture particulière. Ces deux branches se rejoignent et par une transformation elles donnent un nouveau modèle toujours indépendant d'une plateforme mais avec en plus des exigences non fonctionnelles et/ou des qualités de service. La transformation est effectuée de différentes façons. Par exemple, elle peut être soit manuelle, soit automatique, ou intermédiaire ; et utiliser une transformation par annotation ou une transformation par méta-modèle. Le PIM issu de la transformation peut être raffiné si besoin. La branche en bas à droite contient des informations spécifiques à une plateforme. Grâce à ces informations, le dernier PIM non fonctionnel va être transformé en PSM. Ce PSM est donc un modèle spécifique à une plateforme avec des exigences fonctionnelles et non fonctionnelles. Il va par raffinements successifs donner du code [15].

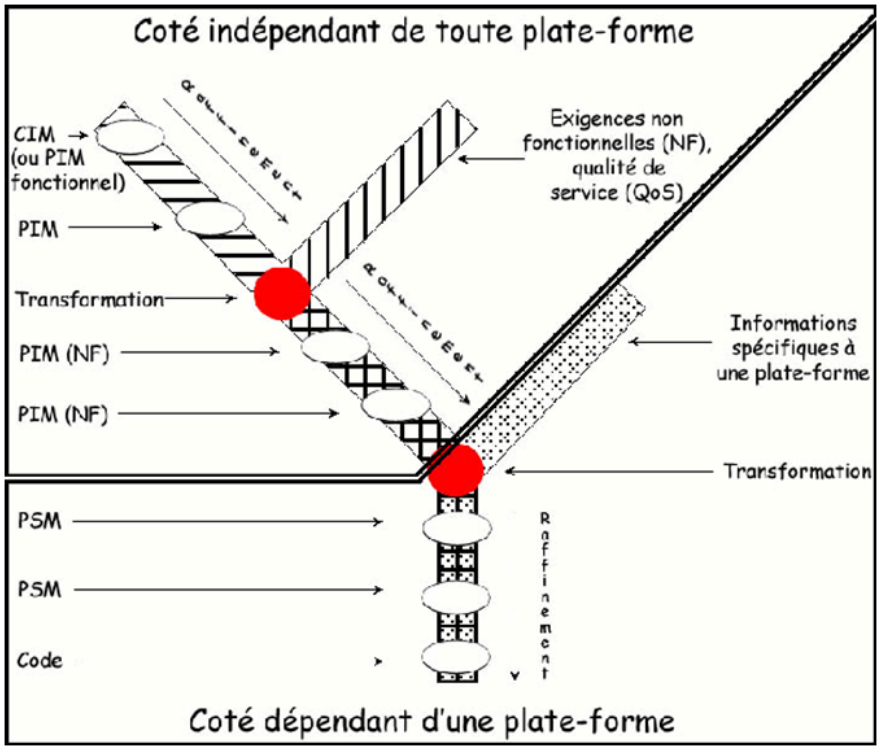

Fig. 2.7 : Cycle de transformation en double Y. [15]

4 Travaux sur la transformation de modèles

Beaucoup de recherches dans le domaine de l'IDM et du MDA se sont concentrées sur les niveaux inférieurs de modèles et les transformations automatiques entre les uns et les autres. Plus particulièrement sur la transformation des modèles PIM en modèles PSM. Notamment les travaux de Jouault [46] sur la transformation de modèles et le langage ATL, les travaux sur les méthodologies de développement de systèmes multi-agents [4, 10, 11, 18, 19, 85, 86, 95, 98] ainsi que les travaux de Silva sur la modélisation des systèmes multi-agents et la transformation des modèles PIM en modèles PSM [102–109]. Toutefois, peu d'attention a été accordée aux transformations de niveau supérieur, à savoir la transformation des modèles CIM en modèles PIM.

La transformation des modèles des exigences d'un système a été adressée dans plusieurs travaux de recherche. Parmi lesquels, nous citerons les travaux présentés dans [3, 28, 55, 65, 92, 94]. Dans [92, 94], les auteurs proposent de transformer le modèle des exigences (CIM) en un modèle indépendant de la plateforme (PIM). Ils commencent à partir d'un processus métier avec la description des exigences de sécurité (BPSec business process with security requirement description), au niveau CIM, afin d'obtenir des cas d'utilisation et des cas d'utilisation sécurisés au niveau PIM. Le BPSec [93] est une extension de BPMN-BPD (Business Process Modeling Notation - Business Process Diagram) [81]. Cette extension a été créée dans le but de permettre la représentation des besoins de sécurité. Le BPSec est utilisé pour spécifier un SBP (Secure Business Process) ; considéré comme modèle CIM. Un ensemble de règles de transformation et de raffinement sont appliquées afin d'obtenir un sous ensemble de cas d'utilisations [75] et de cas d'utilisations de sécurité ; considérés comme modèles PIM. Les règles de transformations et de raffinement sont décrites dans le langage de transformation QVT (Query/ View/ Transformation) [78]. La description du SBP et les cas d'utilisations sont utilisés dans un processus de développement de logiciels tels qu'UP (Unified Process) [43].

Les travaux présentés dans [28] se concentrent sur le développement dirigé par les modèles et le développement des systèmes d'information orientés-services (service-oriented development of information systems). La feuille de route pour la recherche dans le domaine de l'orienté-services décrite par Papazoglou et al. [84] souligne l'importance de définir des méthodologies capables de faciliter l'identification des services utiles et les spécifications des processus métier (BP) existants

dans les scénarios réels, très importants pour le développement des applications orientées-services. Les auteurs proposent une méthode de développement orientée-services (SOD-M) pilotée par les modèles pour le développement des systèmes d'informations. Elle différencie clairement entre les différents niveaux de modélisation, le niveau haut spécifiant le processus métier, la modélisation du système d'informations et la technologie d'implémentation correspondante. Dans cette proposition, le haut niveau du processus métier est représenté par les modèles des exigences (CIM), tandis que le système d'informations est représenté d'abord par l'utilisation des modèles indépendants des plateformes (PIM), puis en les spécifiant dans des modèles spécifiques à une plateforme (PSM). Un ensemble de règles de correspondance entre les deux niveaux d'abstraction CIM et PIM a été mis en œuvre et illustré au moyen d'une étude de cas.

Une méthode analytique de transformation entre le niveau CIM et PIM est décrite dans [55]. Le niveau CIM, dans cette contribution, est représenté par le modèle du processus métier (BP). Le diagramme de flux de données (Data Flow Diagram, DFD) [40] et la description textuelle des processus sont utilisés pour modéliser les processus métier au niveau CIM. Alors que les modèles UML sont utilisés dans le niveau PIM, en particulier les cas d'utilisation, les diagrammes d'activités, les diagrammes de séquence et les diagrammes de domaine. En utilisant ces modèles, un processus de transformation du processus métier représenté par les DFD en diagrammes UML est discuté. Ces transformations sont une base pour une meilleure conception, plus détaillée des systèmes d'informations et des applications logicielles.

Le travail présenté dans [65] décrit STREAM (Strategy for Transition between Requirements models and Architectural Models), une stratégie de transition entre les modèles des exigences et les modèles architecturaux. Les exigences sont définies en termes de modèles i* [126], une approche orienté-buts. La solution est décrite comme un modèle architectural Acme [32]. Le processus proposé est composé de quatre activités, pendant lesquelles, le modèle des exigences est pré-traité pour générer un modèle i* plus modulaire. Pour ce faire, un ensemble de règles de transformation horizontale a été introduit dans [66] et amélioré. Le modèle des exigences résultant est proche d'une conception architecturale, ce qui facilite la transition entre les exigences et la conception architecturale. A partir du modèle modulaire i*, les solutions architecturales Acme sont dérivées à travers un ensemble de mappings entre les concepts de i* et le langage d'Acme.

Dans le cadre de l'utilisation du développement dirigé par les modèles, le travail de recherche

[3] propose une méthode d'interopérabilité entre les modèles. Ce travail introduit un framework de transformation de modèles, SD2PN, permettant une transition entre les diagrammes de séquences et les réseaux de Petri. Cela permet à des modèles conçus avec UML de profiter de l'analyse mathématique rigoureuse offerte par les réseaux de Petri ; et de supporter également l'intégration des différents outils à travers des domaines incompatibles. Les réseaux de Petri sont bien adaptés pour l'analyse structurelle et comportementale d'un modèle grâce à leur puissance et flexibilité.

5 Conclusion

Dans ce chapitre, nous avons présenté l'ingénierie des modèles et ses principaux concepts, principalement la définition des notions de modèle et de méta-modèle et leurs relations. L'approche MDA (Model driven architecture) a été introduite. C'est une approche d'écriture de spécifications et de développement d'applications. Elle prône l'utilisation massive des modèles à différents niveaux d'abstractions. Elle est basée sur les modèles des exigences (CIM), les modèles indépendants des plateformes (PIM) et les modèles spécifiques aux plateformes (PSM).

Nous avons décrit par la suite l'architecture générale du MDA pour la construction des applications qui débute par l'élaboration des modèles des exigences (CIM). Elle se poursuit par l'élaboration des modèles conceptuels de l'application (PIM). La phase suivante consiste à construire des modèles spécifiques des plateformes d'exécution (PSM). Les modèles de chaque niveau sont obtenus par une transformation des modèles du niveau supérieur. Le passage entre les différents modèles du MDA est réalisé par une suite de transformations. Les différents types de transformation sont décrits et plus particulièrement les transformations entre les niveaux de modèles PIM et PSM et les approches de transformations entre ces modèles.

La dernière section de ce chapitre décrit les travaux de recherches réalisés ayant adopter l'approche MDA ou l'IDM dans le développement des applications et notamment ceux qui traitent les transformations entre les modèles des exigences (CIM) et les modèles indépendants des plateformes (PIM). Plusieurs travaux ont été analysés afin de montrer l'importance des modèles des exigences et l'intérêt du passage automatique entre ces modèles et les modèles de niveaux inférieurs. Un tel processus rend les applications et les systèmes conçus plus robustes. Ces travaux ne considèrent pas

le cas des applications distribuées où des propriétés spécifiques, telles que l'échange de messages et la collaborations entre les composants distribués, doivent être prises en compte. D'autant plus qu'ils ne considèrent pas la dérivation du comportement des différents composants du système en question.

Dans ce contexte, en considérant l'importance et l'utilité de la transformation du modèle des exigences en modèles conceptuels et de la dérivation du comportement des composants ou rôles d'un système, nous avons proposé une approche de transformation pour dériver automatiquement le comportement des rôles d'un système à partir de la spécification globale du comportement du système, modélisé par le modèle des exigences. Cette approche est présentée dans le chapitre 3.

Chapitre 3

Dérivation du comportement d'un système par transformation de modèles

1 Introduction

L'ingénierie dirigée par les modèles (IDM) vise à changer l'orientation de développement des logiciels du code aux modèles. IDM est une approche de développement de systèmes, qui met en avant garde la puissance des modèles. Elle sépare la spécification des fonctionnalités du système de sa mise en œuvre sur une plateforme spécifique. Elle fournit les outils nécessaires pour la définition et l'utilisation des modèles des exigences, des modèles de conception, des modèles de déploiement et des modèles de mise en œuvre de code pour les systèmes. Beaucoup de recherches dans IDM se sont concentrées sur les niveaux inférieurs et les transformations automatiques entre les uns et les autres. Plus particulièrement sur la transformation des modèles de conception en modèles de déploiement et plus précisément des modèles PIM en modèles PSM dans l'approche MDA. Notamment les travaux de Jouault [46] sur la transformation de modèles et le langage ATL ainsi que les travaux sur les méthodologies de développement de systèmes multi-agents. Toutefois, moins d'attention a été accordée aux transformations de haut niveau d'abstraction en IDM, à savoir la transformation des modèles des exigences en modèles conceptuels, les transformations des modèles CIM en modèles PIM dans MDA. La transformation du modèle des exigences d'un système

a été adressée dans plusieurs travaux de recherche. Parmi lesquels, nous citerons les travaux présentés dans [3, 28, 55, 65, 92]. Cependant, ces travaux ne considèrent pas le cas des applications distribuées où des propriétés spécifiques, telles que l'échange de messages et la collaboration entre les composants distribués, doivent être prises en compte.

Les systèmes informatiques deviennent de plus en plus complexes. Plusieurs auteurs Castejon [23] et Kruger [57] confirment que le comportement global de ces systèmes n'est pas réalisé par un seul composant mais par la collaboration d'un ensemble de composants. Un tel comportement peut être décomposé en un ensemble de comportements partiels réalisés par les différents composants du système. Par conséquent, une approche de transformation est nécessaire pour dériver automatiquement le comportement de ces composants à partir de la spécification globale du comportement du système, modélisé par le modèle des exigences. Le processus de dérivation du comportement des composants ou entités du système est très important dans le développement des systèmes distribués tels que les systèmes d'information, les systèmes multi-agents et les applications distribuées. Un tel processus de dérivation automatique peut conduire à des systèmes valides et robustes en évitant les erreurs introduites par les activités de conception manuelle.

Pour ce faire, nous proposons dans ce chapitre une approche de dérivation du comportement des différents composants d'un système basée sur la transformation de modèles. Nous considérons que le comportement global du système est modélisé par le modèle des exigences. Les éléments de base du comportement du système sont des activités qui sont en fait des collaborations entre les composants ou les rôles du système. Une collaboration décrit une structure d'éléments (rôles) collaborant, chacun réalisant une fonction spéciale, et qui collectivement accomplissent une certaine fonctionnalité. L'objectif principal d'une collaboration est de montrer le fonctionnement d'un système et par conséquent elle intègre uniquement les aspects de la réalité, jugés pertinents sans se préoccuper des détails. Les collaborations sont très appropriées pour modéliser les exigences car elles fournissent un cadre structurel pour ces exigences. Elles peuvent incarner à la fois les comportements du rôle et les interactions entre les rôles nécessaires pour accomplir le service.

L'approche proposée [36] consiste à dériver le comportement des rôles du système par transformation du modèle des exigences, qui décrit le comportement fonctionnel d'un système d'une manière abstraite vers des modèles conceptuels où chaque rôle est identifié par son comportement local. Les comportements des rôles du système doivent être déterminés de manière que de

la collaboration de ces comportements résulte un comportement satisfaisant le modèle d'expression des besoins global initial. Le processus de transformation, régi par un ensemble de règles, permettra la dérivation du comportement local d'un composant, sous la forme d'un d'automate d'états-transitions, à partir des exigences du système décrites par un diagramme activités UML étendu. Ce processus permettra aussi la dérivation des messages de coordination appropriés entre les automates dérivés.

Dans notre approche, Les modèles des exigences sont modélisés par des diagrammes d'activités étendus (extended UML activity diagrams) où les activités de base sont des collaborations UML. Ils permettent la représentation de concepts qui ne peuvent pas être décrits en utilisant des expressions, comme c'est le cas dans [16]. C'est le cas des conditions intégrées dans les structures de choix et les structures de répétition. En outre, notre approche de transformation considère également tous les nœuds de contrôle spécifiés dans les diagrammes d'activités UML, tels que le nœud de fusion (Merge Node) et le nœud de jonction (Join Node). Dans notre approche, les comportements des rôles du système issus de la transformation sont des machines d'états UML qui pourraient être utilisés pour une génération automatique du code.

Dans cette approche [36], nous nous sommes inspirés de l'algorithme de transformation de Bochmann [16] afin de fournir un processus de dérivation basé sur la transformation par méta-modèles. Cette approche commence par la définition du méta-modèle source (le méta-modèle du diagramme d'activités UML étendus avec des collaborations) et le méta-modèle cible (le méta-modèle de l'automate d'états-transitions UML). L'approche proposée considère les étapes suivantes :

- Définition du méta-modèle des exigences pour spécifier le comportement global d'un système donné.
- Choix du méta-modèle cible afin de modéliser le comportement des rôles du système au niveau conceptuel qui reflète le comportement local de chaque rôle.
- Établissement des règles de transformations qui régissent la transformation au cours du processus de dérivation.

Dans les sections suivantes, nous décrirons le processus de dérivation, l'approche de trans-

formation de modèles et par la suite nous présenterons un cas d'études pour illustrer l'approche de dérivation par transformation de modèles.

2 Processus de dérivation et méta-modèles de base

2.1 L'architecture du processus de dérivation

Le processus de dérivation de notre approche consiste à établir les règles de transformation du modèle des exigences en modèles conceptuels. La définition des règles de transformation, pour régir le processus de dérivation, exige l'identification des diverses relations entre les méta-modèles source et cible en occurrence le méta-modèle des diagrammes d'activités étendus par les collaborations et celui des automates à états-transitions. L'architecture (Fig. 3.1) [37] montre les éléments essentiels de ce processus de dérivation. L'architecture se compose de trois blocs :

1. Le premier bloc est constitué :
   - du méta-modèle des exigences (diagramme d'activités UML étendu avec les collaborations comme indiqué sur la figure (Fig. 3.2)) pour spécifier les besoins globaux d'un système donné ;
   - du méta-modèle cible (UML State Machine) pour décrire le comportement d'un composant (rôle) ; et
   - du modèle source qui représente les exigences globales d'un système.
2. Le deuxième bloc est le composant principal de cette architecture. Il est responsable de la dérivation du comportement des différents rôles du système. Ce bloc consiste en : un module de conformité "*Model to Meta-Model Conformity*", nommé *M2MMC* et un module "*Derivation process rules*". Le module *M2MMC* réalise la conformité du modèle source des exigences avec son méta-modèle et celle du comportement dérivé des rôles avec le méta-modèle cible. Le module "*Derivation process rules*" réalise la dérivation. Il est formé des règles de transformation de modèles qui accomplissent la dérivation du comportement des différents rôles

à partir du modèle des exigences globales du système. Cette dérivation est faite de manière à générer les différents concepts d'une machine d'états pour chaque rôle du système.

3. Le troisième bloc est constitué des automates d'états-transitions obtenus par dérivation et qui reflètent le comportement local de chaque rôle.

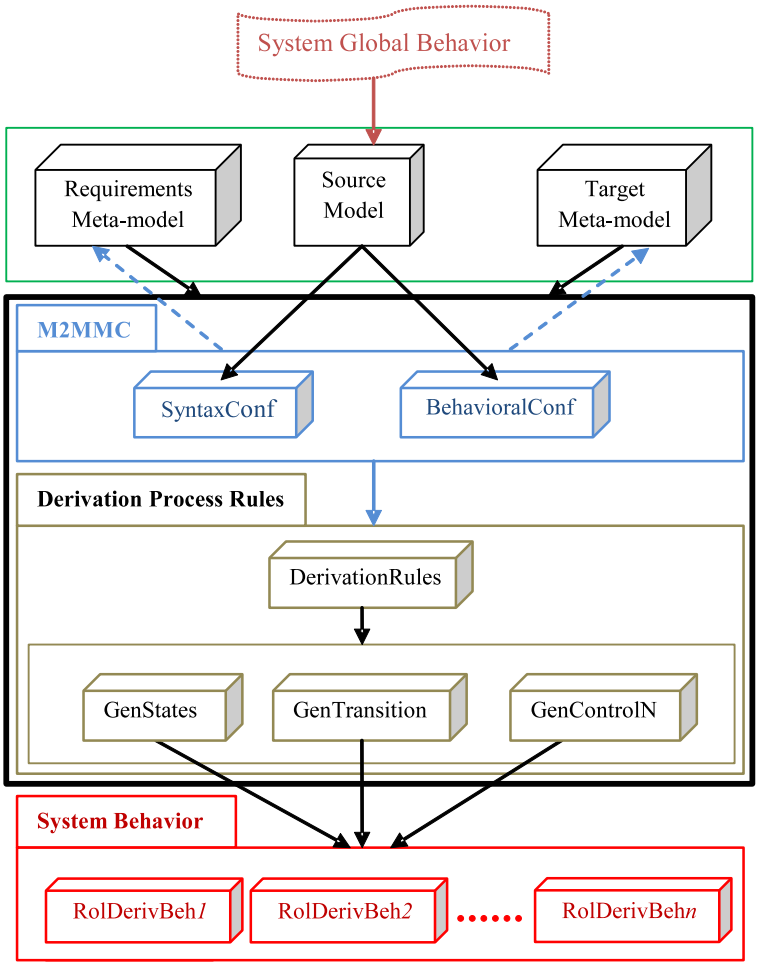

Fig. 3.1 : L'architecture du processus de dérivation.

Plus de détails sur les différentes composantes de l'architecture seront donnés dans les sections suivantes.

2.2 Les méta-modèles de base

Les méta-modèles de base de notre approche sont le méta-modèle des exigences et le méta-modèle cible des automates à états-transitions UML.

2.2.1 Le méta-modèle des exigences

Les diagrammes d'interactions et les diagrammes de communications ou de séquences UML [76] sont généralement utilisés pour décrire des comportements collaboratifs et ils ont prouvés leurs validités. Cependant, ils ont certaines limites. Ils sont exprimés en termes d'échanges de messages entre les différents composants collaborant à des étapes préliminaires du développement où il n'est pas nécessaire d'avoir trop de détails [22, 24]. Plusieurs notations ont été proposées pour définir les modèles des exigences ou des besoins pour les systèmes, en particulier les diagrammes d'activités, les cas d'utilisation, les diagrammes de collaborations en UML et le langage d'exécution de processus (BPEL). Les différentes notations ont des concepts communs mais aussi des différences importantes. Dans ces notations, le comportement global d'un système peut être décomposé en plusieurs sous activités et ainsi jusqu'à l'obtention d'activités primitives ou simples. Pour la plupart des applications distribuées, systèmes distribués ou systèmes multi-agents, les blocs de base dans un comportement sont les activités et les collaborations entre les différents composants du système. Notons que les diagrammes d'activités (AD) sont la seule notation standard de l'OMG pour la modélisation des processus métier (BP) et de workflow [97]. Un diagramme d'activités est destiné à décrire une séquence d'actions pouvant appartenir à plusieurs objets.

Pour représenter le modèle des exigences du système, nous proposons l'utilisation des diagrammes d'activités UML (AD) étendus avec des collaborations. Les collaborations représentent les principaux blocs d'activités pour la construction des modèles des besoins [23]. Les diagrammes d'activités UML sont très appropriés pour représenter une chorégraphie de collaborations et de sous-collaborations (les activités de base dans un diagramme d'activités). Le comportement global d'un système est ainsi décrit par une collaboration composite. Les diagrammes d'activités peuvent exprimer des comportements séquentiels, alternatifs et concurrents incluant les possibilités d'utilisation de structure de choix, de répétition aussi bien que les interruptions.

Le méta-modèle des exigences proposé (Fig. 3.2) est considéré comme le méta-modèle source des transformations. Il définit le méta-modèle du diagramme d'activités avec ses principales classes et associations avec l'introduction des collaborations comme activités de base. Un diagramme d'activités est composé de plusieurs *ActivityNode* et *ActivityEdge*. La classe *ActivityEdge* permet de spécifier les contrôles de flux (connections) entre les classes *ActivityNode*. La classe *ActivityNode* spécifie les différents opérateurs de séquencement des activités définis par la classe *ControlNode*. Cette classe définit aussi les activités et les régions interruptibles. Les activités au niveau de notre méta-modèle sont définies par des collaborations impliquant plusieurs rôles. Une collaboration peut être composée d'une ou de deux collaborations ou sous-collaborations. Au niveau modèles, le comportement global d'un système sera défini par une composition de diagramme d'activités et de collaborations ; instance du méta-modèle des exigences.

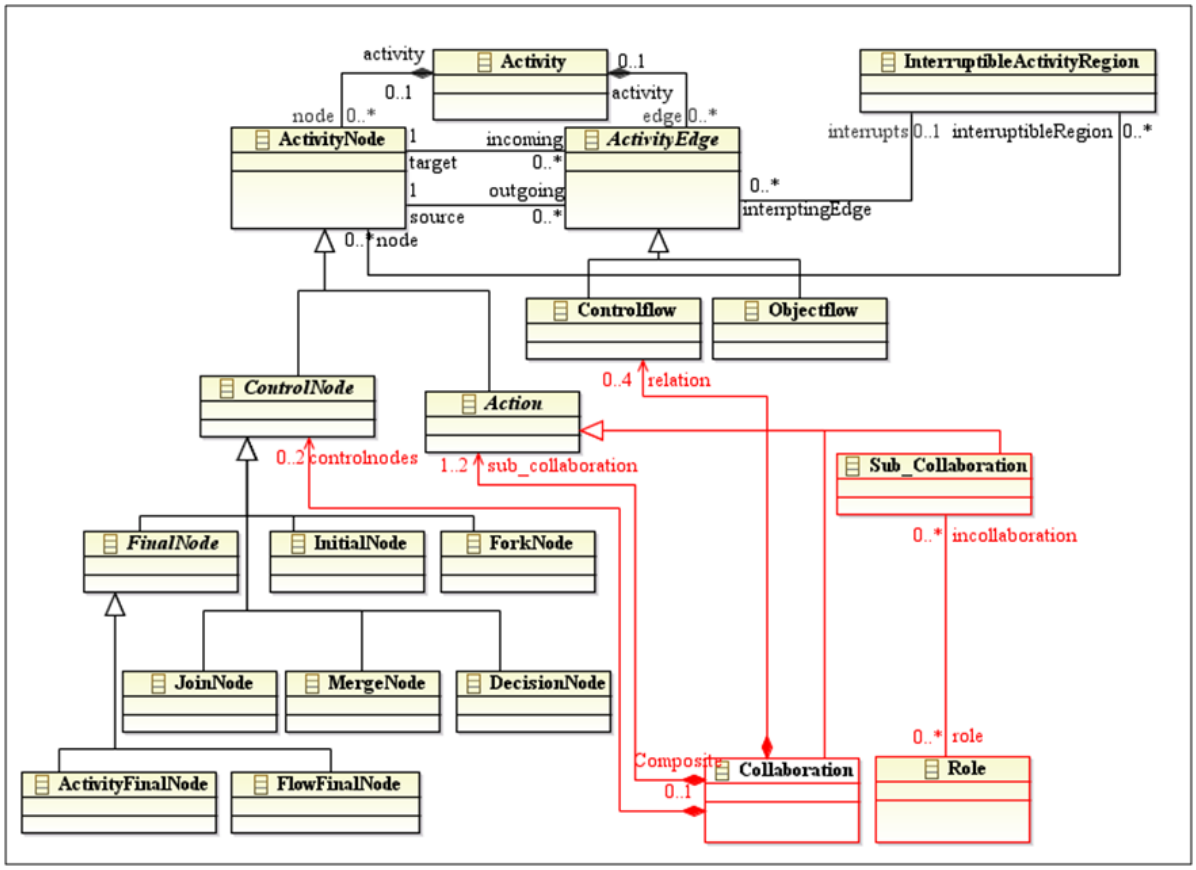

Fig. 3.2 : Le méta-modèle des exigences.

2.2.2 Le méta-modèle cible des automates à états-transitions

Au niveau conceptuel, le comportement des différents composants du système est modélisé en utilisant les diagrammes d'activités ou les automates d'états-transitions UML par exemple. Les transformations de ces modèles vers du code peuvent être effectuées et automatisées.

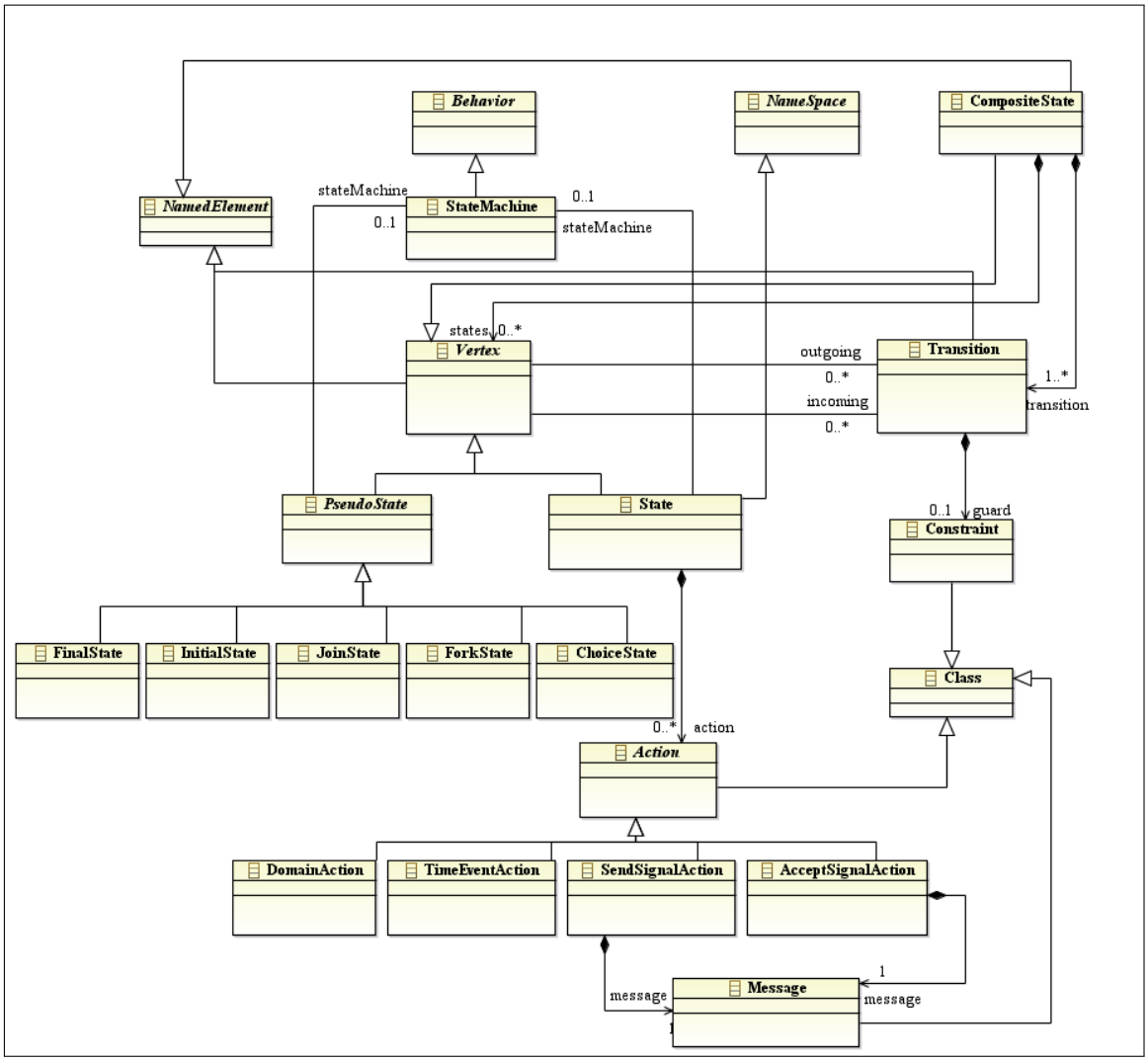

Fig. 3.3 : Le méta-modèle cible de l'automate d'états-transitions.

La figure (Fig. 3.3) illustre le méta-modèle du diagramme d'automate à états-transitions (State Machine) [76]. Il est utilisé comme méta-modèle cible de la transformation. Le diagramme

d'états-transitions permet de décrire le comportement d'un composant du système. Les principales classes de ce méta-modèle sont les classes *State* et *Transition*. La classe *State* modélise une situation ou une phase significative durant le cycle de vie du composant. La classe *Transition* modélise une relation entre un état source et un état cible, représentant comment le composant peut répondre à l'occurrence d'un événement donné quand il se trouve dans l'état source. Au niveau modèle, le comportement d'un rôle du système sera décrit par un automate d'états-transitions issu du processus de dérivation ; instance du méta-modèle cible.

3 L'approche de transformation

La figure (Fig. 3.1) montre que notre approche de transformation est intégrée dans le bloc de transformation. La composante la plus importante de ce bloc est "*Derivation Process Rules*", composée des règles de transformation. Une règle de transformation consiste à transformer un concept défini dans le méta-modèle source en un autre concept correspondant dans le méta-modèle cible. Pour ce faire, nous définissons la fonction nommée *Transform (Requirement_Concept, SM_Concept, Messages)*. Cette fonction effectue deux types de transformation des concepts modélisés dans le modèle des exigences : le premier type consiste en un mapping direct d'un concept du méta-modèle des exigences en un concept du méta-modèle automate d'états-transitions (comme dans le cas des transitions et des nœuds de contrôle). Le deuxième type consiste à transformer le concept collaboration en un état composite. Si le type de la collaboration est une sous-collaboration, cet état composite contiendra les actions du rôle concerné après transformation sinon la fonction de transformation déclenchera la transformation de la collaboration composite. Le paramètre *Messages* dans cette fonction représente les messages de coordination inclus dans l'état composite généré. Les règles de transformations correspondantes seront décrites dans la section 3.2.

La définition des règles de transformation nécessite l'identification des diverses relations entre les deux méta-modèles source et cible ; dans notre cas entre le méta-modèle des exigences et celui des automates d'états-transitions. Le tableau (Tab. 3.1) montre les correspondances entre les concepts du méta-modèle des exigences et ceux du méta-modèle cible.

Concept Source	Concept Cible
Collaboration	Composite State
Rôle	Automate d'états-transitions
SendMessage	State with action SendSignalAction
ReceiveMessage	State with action AcceptSignalAction
Action	State with action
ActivityEdge	Transition
Guard	Condition
DecisionNode	ChoiceState
MergeNode	JunctionState
JoinNode	JoinState
Fork Node	ForkState

Tab. 3.1 : Les correspondances entre les concepts des méta-modèles source et cible.

3.1 L'algorithme du processus de dérivation

L'algorithme du processus de dérivation réalise en premier lieu, pour un rôle donné, la conformité du modèle source (Ms) avec le méta-modèle des exigences (MMs). Le modèle source représente la spécification des exigences globales d'un système donné. Après vérification de la conformité, l'algorithme détermine alors la règle à appliquer pour chaque concept *Collaboration* représenté dans le modèle (Ms) et il génère un état composite dans le modèle cible. Cet état composite possède le même nom que la collaboration transformée et englobe un automate d'états-transitions spécifiant les actions de ce rôle. L'algorithme génère également les transitions nécessaires à l'interconnexion des états générés dans l'état composite. En plus des actions du rôle, l'état composite contient également les messages de coordination appropriés. Pour chaque concept *ControlFlow*, l'algorithme appliquera la règle appropriée afin de générer les transitions nécessaires pour connecter l'état composite généré avec ses états prédécesseurs et successeurs. De la même façon, le concept *ControlNode* est transformé en un *ControlState* et il est relié par les transitions appropriées.

Algorithm 1 L'algorithme du processus de dérivation.

Init r = Role_name ;
Check syntax conformity of the (M_s) with (MM_s) ;
for all *Concept*= collaboration C in M_s **do**
  Generate a composite State S in the RolderivBeh r conform to the Target Meta-model MM_t ;
  $S.Name = C.Name$; Generate (Initial State) in S ;
  if type_collaboration C = sub-collaboration **then**
    for each action = a in C **do**
      Transform (a, a')
    end for
  else
    for each C' in C **do**
      Transform(C', S', Messages) ;
    end for
  end if
  if *Messages* $\neq$ Nil **then**
    Integrate the coordination messages in S
  end if
  Generate the transitions to connect the states in S ;
  Generate (Final State) in S ;
end for
for all *Concept*= Controlflow or ControlNode C in M_s **do**
  Transform(controlflow, Transition) ;
  Transform(controlNode, ControlState) ;
end for

3.2 Les règles de transformation

Pour réaliser la dérivation, nous identifions plusieurs cas de chorégraphie de collaborations exprimés dans le méta-modèle des exigences : les comportements séquentiels, alternatifs (structure de choix), concurrents, de répétition aussi bien que les interruptions. Parmi les rôles participants d'une collaboration, on distingue les rôles *initiateurs* et les rôles *terminateurs* [16]. Un rôle *initiateur* est un rôle qui accomplit une action initiale dans une collaboration ou dans une de ses sous-collaborations. Un rôle *terminateur* est un rôle qui accomplit une action finale dans une collaboration ou dans une de ses sous-collaborations. Ils sont notés comme il est indiqué sur la figure (Fig. 3.4).

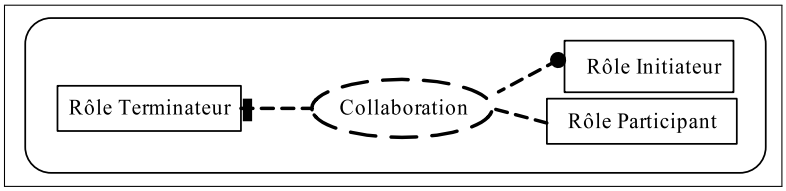

Fig. 3.4 : Structure d'une collaboration.

Nous considérons que la décision de choix est faite localement à un rôle. Un tel rôle est le rôle initiateur dans la collaboration de choix (il devrait être le rôle initiateur dans les deux collaborations de la structure de choix). Dans le cas où la décision n'est pas locale, un traitement supplémentaire et des messages de coordination sont ajoutés pour déterminer le rôle responsable du choix. Ce traitement est défini par une négociation entre les nœuds initiateurs de la collaboration structure de choix ou répétition. Afin de déterminer les ensembles des rôles participants, des rôles initiateurs et des rôles terminateurs, nous utilisons les mêmes techniques de calcul que celles utilisées dans [16]. Ces ensembles sont calculés pour une collaboration en fonction des opérateurs de séquencement utilisés dans les diagrammes d'activités. Deux types de séquencement sont distingués [22, 23] : Strong et Weak. Le séquencement Weak (Fig. 3.6) spécifie qu'une activité A2 sera exécutée après une autre activité A1. Par contre, le séquencement Strong (Fig. 3.5) implique que toutes les sous-activités de A1 soient finies avant qu'une activité de A2 ne puisse commencer. Le séquencement Weak permet d'obtenir seulement un ordre local des activités pour chaque rôle du système et n'implique pas un ordre global. Le tableau (Tab. 3.2) énumère les règles de calcul de ces ensembles.

Cas de Chorégraphie	**Rôles Initiateurs (SR)**	**Rôles Terminateurs (TR)**	**Rôles Participants (PR)**
Sub-collaboration <C>	SR(C)	TR(C)	PR(C)=R
Weak Sequencing <C_1, C_2>	SR(C_1)U(SR(C_2)-PR(C_1))	TR(C_2) U(TR(C_1)-PR(C_2))	PR(C_1) U PR(C_2)
Strong Sequencing <C_1, C_2>	SR(C_1)	TR(C_2)	PR(C_1) U PR(C_2)
Choice composition<C_1, C_2>	SR(C_1) U SR(C_2) = {r}	TR(C_1) U TR(C_2)	PR(C_1) U PR(C_2)
Strong While loop <C_1, C_2>	SR(C_1) U SR(C_2) = {r}	TR(C_2) ; SR(C_1) if C_2=ε	PR(C_1) U PR(C_2)
Weak While loop <C_1, C_2>	SR(C_1) U SR(C_2) = {r}	TR(C_2) U (TR(C_1)- PR(C_2)	PR(C_1) U PR(C_2)
Parallel behavior <C_1, C_2>	SR(C_1) U SR(C_2)	TR(C_1) U TR(C_2)	PR(C_1) U PR(C_2)

Tab. 3.2 : Règles de calcul des ensembles de rôles initiateurs, terminateurs et participants. [16]

Deux messages de coordination sont introduits au niveau du processus de dérivation afin de réaliser la coordination des différents rôles. Nous utilisons le même de type de messages décrits dans [16, 24]. Chaque message contient les paramètres suivants :

a) le rôle Source (Sr) ;

b) le rôle Destinataire (Dr) ;

c) le nom de l'état concerné (St).

Les messages de coordination sont :

i) Message de flux (Flow message) pour réaliser la coordination entre les rôles dans un séquencement, nommé *Flowm(Sr, Dr, St)*.

ii) Message d'indication de choix (Choice indication message) pour la propagation du choix aux rôles non participants dans l'alternative de choix sélectionnée dans une structure de choix ou de répétition, nommé *Choicem(Sr, Dr, St)*.

La transformation du concept collaboration déclenche la transformation de ses sous-collaborations dans le cas où elle est composée. Cette propriété nécessite la définition de règles de transformation récursives. Nous avons défini les différentes règles de transformations nécessaires pour les différents cas de chorégraphie exprimés dans le méta-modèle des exigences. Nous exprimons ci-dessous les règles qui vont permettre la dérivation du comportement des différents rôles impliqués dans le comportement global du système. Chaque règle réalise la transformation appropriée pour un rôle *r* participant dans une collaboration C_i.

3.2.1 Séquence Strong entre deux collaborations

Le message *Flowm(Sr, Dr, St)* est utilisé pour réaliser la synchronisation entre les comportements dérivés des rôles.

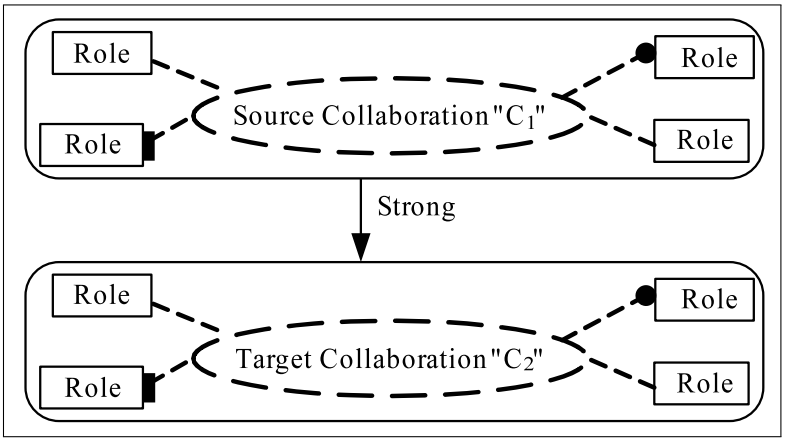

Fig. 3.5 : Séquence Strong entre deux collaborations.

Règle 1 : Si le rôle *r* est un rôle terminateur de la collaboration source Alors le concept *collaboration* est transformé en un concept *état composite*. Cet état composite est constitué des actions du rôle *r* après transformation et des actions d'envoi de messages de coordination *Flowm* à l'ensemble des rôles initiateurs de la collaboration cible (à l'exception de soi même).

If $r \in TR(C_1)$ then $Transform(C_1, S_1, Send(Flowm(r, r', S_2)))$ $\forall r' \in (SR(C_2) - r)$;

La fonction transforme la collaboration C_1 en un état composite S_1. Cet état composite est

constitué des actions du rôle r après transformation et des actions d'envoi de messages de coordination *Flowm* à l'ensemble des rôles initiateurs de la collaboration C_2 (à l'exception de soi même s'il appartient à l'ensemble des rôles initiateurs de la collaboration C_2).

Règle 2 : Si le rôle r est un rôle initiateur de la collaboration cible Alors le concept *collaboration* est transformé en un concept *état composite*. Cet état composite est constitué des actions de réception de messages de coordination *Flowm* de l'ensemble des rôles terminateurs de la collaboration source (à l'exception de soi même) et des actions du rôle r après transformation.

$$\text{If } r \in SR(C_2) \text{ then } Transform(C_2, S_2, Receive(Flowm(r', r, S_2))) \qquad \forall r' \in (TR(C_1) - r);$$

La collaboration C_2 est transformée en un état composite S_2. Cet état composite est constitué des actions de réception de messages de coordination *Flowm* de l'ensemble des rôles terminateurs de la collaboration C_1 (à l'exception de soi même s'il appartient à l'ensemble des rôles terminateurs de la collaboration C_1) et des actions du rôle r après transformation.

Règle 3 : Si le rôle r participe dans la collaboration source (non terminateur) ou la collaboration cible (non initiateur) Alors le concept *collaboration* est transformé en un concept *état composite*. Cet état composite est constitué des actions du rôle en question après transformation.

$$\text{If } r \in (PR(C_1) - TR(C_1)) \text{ or } r \in (PR(C_2) - SR(C_2)) \text{ then } Transform(C_i, S_i) \qquad \forall i = 1, 2;$$

Règle 4 : Si le r rôle participe dans les collaborations source et cible Alors le concept *Controlflow* est transformé en un concept *Transition*. La transition résultante relie les états composites source et cible issus de la transformation des collaborations source et cible respectivement.

$$\text{If } r \in PR(C_1) \text{ and } r \in PR(C_2) \text{ then } Transform(Controlflow, Transition);$$

Cette transition connecte les états S_1 et S_2 issus de la transformation de C_1 et C_2.

3.2.2 Séquence Weak entre deux collaborations

Pour la transformation du concept *Controlflow*, la Règle 4 est appliquée.

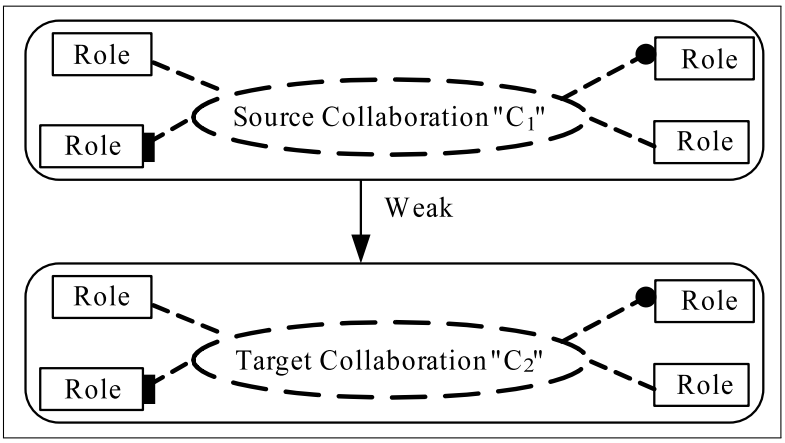

Fig. 3.6 : Séquence Weak entre deux collaborations.

Règle 5 : Si le rôle *r* participe dans la collaboration source ou cible Alors le concept *collaboration* est transformé en un concept *état composite*. Cet état composite est constitué des actions du rôle *r* après transformation.

$$\text{If } r \in PR(C_i) \text{ then } Transform(C_i, S_i) \qquad \forall i = 1,2;$$

3.2.3 Choix entre deux collaborations (Choix local)

Nous considérons que la décision de choix est faite localement au niveau d'un rôle donné. C'est le rôle initiateur dans la structure de choix (Fig. 3.7). Le message d'indication de choix *Choicem(Sr, Dr, St)* est utilisé pour la propagation du choix à un rôle non participant dans la collaboration sélectionnée mais il participe dans la deuxième alternative de la structure de choix.

Règle 6 : Si le rôle *r* participe dans une collaboration de choix et il est responsable du choix (rôle Initiateur dans la collaboration) Alors le concept *collaboration* est transformé en un concept *état composite*. Cet état composite est constitué des actions du rôle en question après transformation et des actions d'envoi de messages de coordination *Choicem* à l'ensemble des rôles ne participant pas à la collaboration choisie et membres de l'autre collaboration.

$$\text{If } r \in SR(C_i) \text{ then } Transform(C_i, S_i, Send(Choicem(r, r', S_i)))$$

$$\forall r' \in (PR(C_{i'}) - PR(C_i)) \qquad \forall i, i' = 1,2 \text{ et } i \neq i';$$

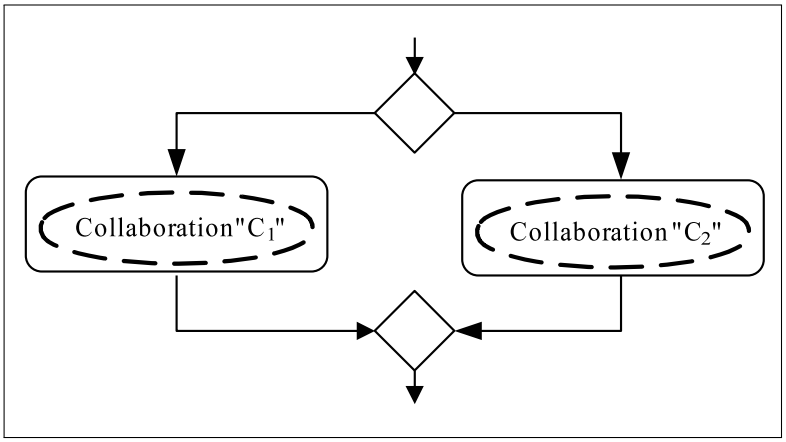

Fig. 3.7 : Structure de choix.

Règle 7 : Si le rôle *r* participe dans une collaboration de choix et il n'est pas responsable du choix Alors le concept *collaboration* est transformé en un concept *état composite*. Cet état composite est constitué des actions du rôle *r* après transformation.

$$\text{If } r \in PR(C_i) \text{ and } r \notin SR(C_i) \text{ then } Transform(C_i, S_i) \qquad \forall i = 1,2;$$

Règle 8 : Si le rôle *r* participe dans une des deux collaborations du choix Alors le concept *collaboration* est transformé en un concept *état composite* au niveau de la collaboration à laquelle il n'est pas membre. Cet état composite est constitué d'une action de réception de message de coordination *Choicem* de la part du rôle initiateur de la collaboration de choix.

$$\text{If } r \in (PR(C_i) - PR(C_{i'})) \qquad \forall i, i' = 1,2 \text{ and } i \neq i' \text{ then}$$

$$Transform(C_{i'}, S_{i'}, Receive(Choicem(r', r, S_{i'}))) \qquad \forall r' \in SR(C_{i'});$$

Règle 9 : Si le rôle *r* participe dans une collaboration de choix Alors le concept *Controlflow* est transformé en un concept *Transition*. La transition résultante relie les états S_1 et S_2 issus de la transformation des collaborations C_1 et C_2 composant la structure de choix aux nœuds de contrôle correspondants.

$$\text{If } r \in PR(C_i) \text{ then } Transform(Controlflow, Transition) \qquad \forall i = 1,2;$$

Règle 10 : Si le rôle *r* participe dans une collaboration de choix Alors le concept *DecisionNode* est transformé en un concept *ChoiceState*. L'état résultant est relié aux états composites S_1 et S_2 issus de la transformation des collaborations C_1 et C_2 composant la structure de choix par les transitions correspondantes obtenues par application de la règle 9.

$$\text{If } r \in PR(C_i) \text{ then } Transform(DecisionNode, ChoiceState) \qquad \forall i = 1,2;$$

Règle 11 : Si le rôle *r* participe dans une collaboration de choix Alors le concept *MergeNode* est transformé en un concept *JunctionState*. L'état résultant est relié aux états composites S_1 et S_2 issus de la transformation des collaborations C_1 et C_2 composant la structure de choix par les transitions correspondantes obtenues par application de la règle 9.

$$\text{If } r \in PR(C_i) \text{ then } Transform(MergeNode, JunctionState) \qquad \forall i = 1,2;$$

3.2.3.1 Exemple : Nous présentons ci-dessous un exemple afin d'illustrer le processus de dérivation. Le modèle des exigences globales du système (Fig. 3.8) est décrit par une collaboration. Cette collaboration est une séquence entre la sous-collaboration "C_1" et une collaboration composite nommé "C_2". La collaboration "C_2" définit un choix entre les deux sous-collaborations "C_3" et "C_4". Nous supposons que chaque sous-collaboration est constituée d'une seule action. Le tableau (Tab. 3.3) montre les différents ensembles calculés selon le tableau (Tab. 3.2).

Cas de Chorégraphie	**Rôles Initiateurs (SR)**	**Rôles Terminateurs (TR)**	**Rôles Participants (PR)**
Sub-collaboration "C_1"	{r1}	{r1}	{r1, r2, r3}
Sub-collaboration "C_3"	{r1}	{r1, r4}	{r1, r2, r3, r4}
Sub-collaboration "C_4"	{r1}	{r3}	{r1, r3}
Collaboration "C_2"	{r1}	{r1, r3, r4}	{r1, r2, r3, r4}

Tab. 3.3 : Les rôles initiateurs, terminateurs et participants au niveau des collaborations.

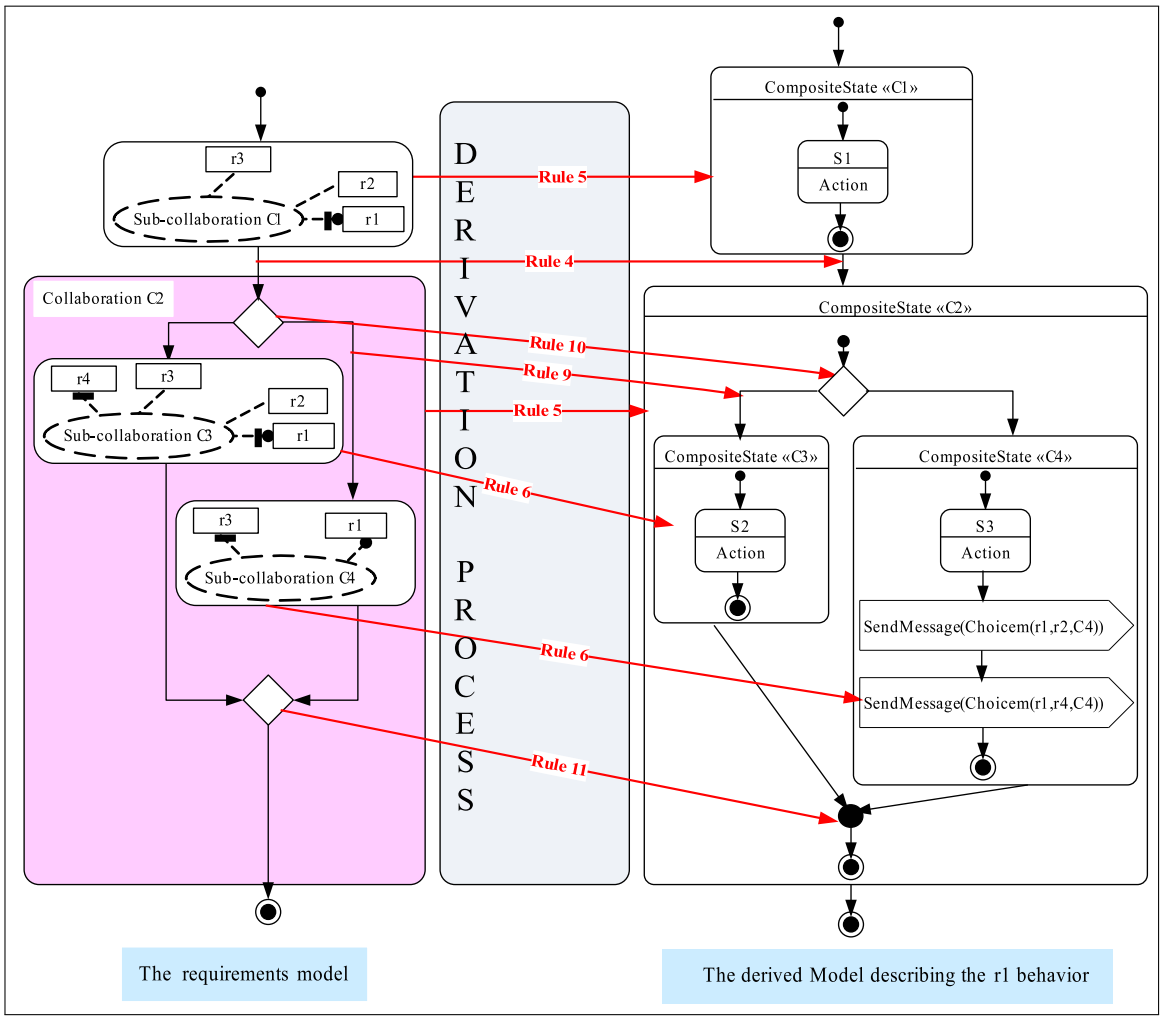

Fig. 3.8 : Exemple du processus de dérivation.

Nous considérons dans cet exemple, les règles de transformation qui permettent de dériver le comportement du rôle *r1*. Le modèle des exigences globales est constitué d'une séquence Weak entre les collaborations "C_1" et "C_2" et le rôle *r1* participe dans les deux collaborations. Par conséquent, le processus de dérivation déclenche les règles 4 et 5. La règle 5 transforme la sous-collaboration "C_1" en un état composite contenant les actions du rôle concerné et déclenche la transformation de la collaboration "C_2". La règle 4 effectue la transformation du *Controlflow* en une transition qui relie les états composites résultant de la transformation de "C_1" et "C_2". La collaboration "C_2" est une structure de choix entre les sous-collaborations "C_3" et "C_4" et le rôle *r1* est responsable du choix (il est le rôle initiateur dans les deux sous-collaborations). Par conséquent,

le processus de dérivation déclenche les règles 6, 9, 10 et 11. La règle 6 réalise la transformation de chaque sous-collaboration intervenant dans la structure de choix en un état composite en occurrence les sous-collaborations "C_3" et "C_4". Chaque état composite contiendra les actions effectuées par le rôle *r1* au niveau chaque sous-collaboration (C_3 ou C_4). Par ailleurs, il doit informer les rôles *r2* et *r4* ne participant pas à la sous-collaboration "C_4" en envoyant un message de coordination Choicem pour indiquer la sélection de "C_4". Ces actions sont incluses dans l'état composite "C_4". Les règles 9, 10 et 11 réalisent de la transformation du *Controlflow* et des nœuds de contrôle. Le processus de dérivation génère un automate à états décrivant le comportement du rôle *r1* (Fig. 3.8).

3.2.4 Boucle Tantque avec Séquencement Strong (Choix local)

Il y a un séquencement strong entre la fin de la collaboration C_1 et le choix de l'exécution de C_1 une autre fois ou de terminer la boucle avec l'exécution de la collaboration C_2 (Fig. 3.9).

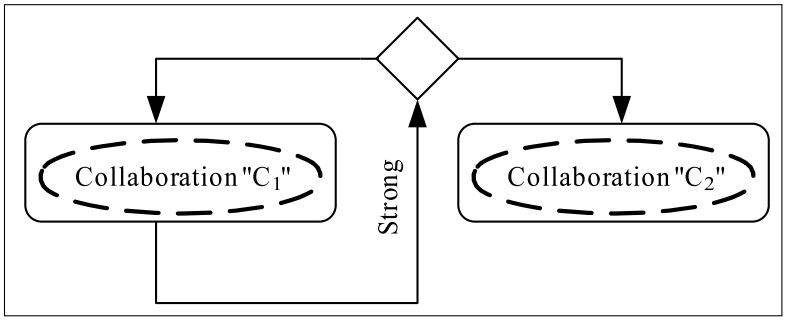

Fig. 3.9 : Boucle Tantque avec séquencement Strong.

Règle 12 : Si le rôle *r* est un rôle terminateur de la collaboration 1 Alors le concept *collaboration* est transformé en un concept *état composite*. Cet état composite est constitué des actions du rôle en question après transformation. Comme le rôle est un rôle de terminateur dans la collaboration 1, il doit envoyer en parallèle un message de coordination *Flowm* à l'ensemble des rôles initiateurs de cette collaboration (à l'exception de soi même s'il appartient à cet ensemble).

$$\text{If } r \in TR(C_1) \text{ then } Transform(C_1, S_1, Send(Flowm(r, r', S_1))) \qquad \forall r' \in (SR(C_1) - r)\,;$$

Règle 13 : Si le rôle *r* est un rôle initiateur de la collaboration 1 Alors le concept *collaboration* est

transformé en un concept *état composite*. Cet état composite est constitué des actions de réception des messages de coordination *Flowm* de l'ensemble des rôles terminateurs de cette collaboration (à l'exception de soi même s'il appartient à cet ensemble) et des actions du rôle en question après transformation.

$$\text{If } r \in SR(C_1) \text{ then } Transform(C_1, S_1, Receive(Flowm(r', r, S_1))) \qquad \forall r' \in (TR(C_1) - r)\,;$$

Règle 14 : Si le rôle *r* participe dans la collaboration 1 (non terminateur et non initiateur) Alors le concept *collaboration* est transformé en un concept *état composite*. Cet état composite est constitué des actions du rôle en question après transformation.

$$\text{If } r \in PR(C_1) \text{ and } (r \notin SR(C_1) \text{ and } r \notin TR(C_1)) \text{ then } Transform(C_1, S_1)\,;$$

Règle 15 : Si le rôle *r* participe dans la collaboration 2 et il est responsable du choix (rôle initiateur dans la structure de répétition) Alors le concept *collaboration* est transformé en un concept *état composite*. Cet état composite est constitué des actions du rôle en question après transformation. Comme le rôle est responsable du choix, il doit envoyer en parallèle un message de coordination *Choicem* à l'ensemble des rôles ne participant pas à la collaboration 2 et participant dans la collaboration 1.

$$\text{If } r \in SR(C_2) \text{ then } Transform(C_2, S_2, Send(Choicem(r, r', S_2))) \quad \forall r' \in (PR(C_1) - PR(C_2))\,;$$

Règle 16 : Si le rôle *r* participe dans la collaboration 2 et il est non initiateur Alors le concept *collaboration* est transformé en un concept *état composite*. Cet état composite est constitué des actions du rôle en question après transformation.

$$\text{If } r \in PR(C_2) \text{ and } r \notin SR(C_2) \text{ then } Transform(C_2, S_2)\,;$$

Règle 17 : Si le rôle *r* participe dans la collaboration 1 et ne participe pas dans la collaboration 2 Alors le concept *collaboration* est transformé en un concept *état composite* au niveau de la collaboration 2. Cet état composite est constitué d'une action de réception de message de coordination *Choicem* du rôle initiateur de la collaboration 2 (rôle initiateur de la collaboration structure de choix).

$$\text{If } r \in (PR(C_1) - PR(C_2)) \text{ then } Transform(C_2, S_2, Receive(Choicem(r', r, S_2))) \quad \forall r' \in SR(C_2)\,;$$

Règle 18 : Si le rôle *r* participe dans la collaboration 1 Alors le concept *Controlflow* est transformé en un concept *Transition*. La transition résultante de la transformation relie les états S_1 et S_2 issus de la transformation des collaborations C_1 et C_2 à l'état de contrôle correspondant.

If $r \in PR(C_1)$ then $Transform(Controlflow, Transition)$;

Règle 19 : Si le rôle *r* participe dans la collaboration 1 Alors le concept *DecisionNode* est transformé en un concept *ChoiceState*. L'état résultant de la transformation est relié aux états composites S_1 et S_2 issus de la transformation des collaborations C_1 et C_2 par les transitions correspondantes obtenues par application de la règle 18.

If $r \in PR(C_1)$ then $Transform(DecisionNode, ChoiceState)$;

3.2.5 Boucle Tantque avec Séquencement Weak (Choix local)

Il y a un séquencement weak entre la fin de la collaboration C_1 et le choix de l'exécution de C_1 une autre fois ou de terminer la boucle avec l'exécution de la collaboration C_2 (Fig. 3.10). La transformation du concept collaboration pour un rôle *r* participant dans la collaboration 2 et initiateur, un rôle participant dans la collaboration 2 non initiateur et pour un rôle participant dans la collaboration 1 et ne participant pas dans la collaboration 2 nécessite le déclenchement respectif des règles 15, 16 et 17. La transformation des concepts *Controlflow* et *DecisionNode* pour un rôle participant dans la collaborations 1 déclenche les règles 18 et 19 respectivement.

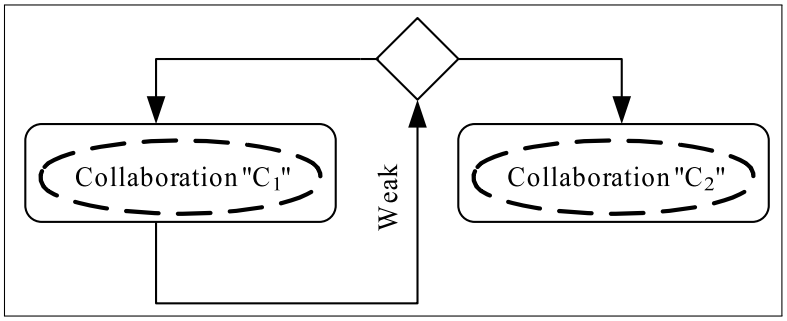

Fig. 3.10 : Boucle Tantque avec séquencement Weak.

Règle 20 : Si le rôle *r* participe dans la collaboration 1 Alors le concept *collaboration* est transformé

en un concept *état composite*. Cet état composite est constitué des actions du rôle en question après transformation.

If $r \in PR(C_1)$ then $Transform(C_1, S_1)$;

3.2.6 Parallélisme entre deux collaborations

Dans le cas où un rôle *r* participe seulement à une des deux collaborations (Fig. 3.11), les nœuds de contrôle et les *Controlflow* ne sont pas transformés mais seulement la collaboration concernée est transformée.

Règle 21 : Si le rôle *r* participe dans une collaboration Alors le concept *collaboration* est transformé en un concept *état composite* ; constitué des actions du rôle en question après transformation.

If $r \in PR(C_i)$ then $Transform(C_i, S_i)$ $\forall i = 1,2$;

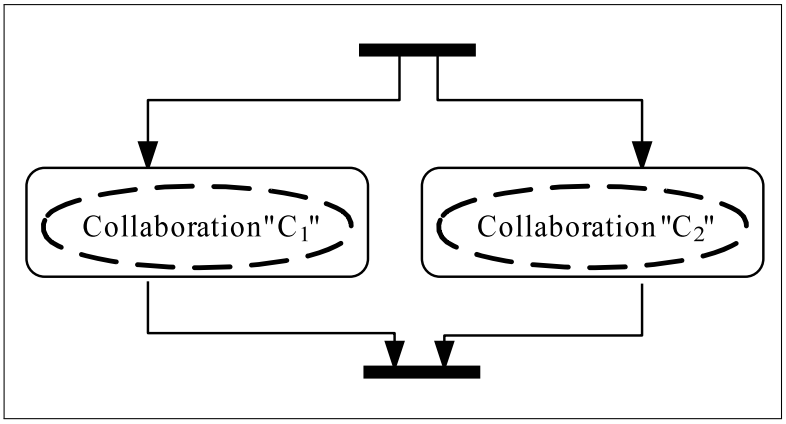

Fig. 3.11 : Parallélisme entre deux collaborations.

Règle 22 : Si le rôle *r* participe dans les deux collaborations Alors le concept *Controlflow* est transformé en un concept *Transition*. La transition résultante de la transformation relie les états composites S_1 et S_2 issus de la transformation des collaborations C_1 et C_2 aux états de contrôle correspondants.

If $r \in PR(C_1)$ and $r \in PR(C_2)$ then $Transform(Controlflow, Transition)$ $\forall i = 1,2$;

Règle 23 : Si le rôle r participe dans les deux collaborations Alors le concept *ForkNode* est transformé en un concept *ForkState*. L'état résultant de la transformation est relié aux états composites S_1 et S_2 issus de la transformation des collaborations C_1 et C_2 par les transitions correspondantes obtenues par application de la règle 22.

$$\text{If } r \in PR(C_1) \text{ and } r \in PR(C_2) \text{ then } Transform(ForkNode, ForkState) \qquad \forall i = 1,2\,;$$

Règle 24 : Si le rôle r participant dans les deux collaborations Alors le concept *JoinNode* est transformé en un concept *JoinState*. L'état résultant de la transformation est relié aux états composites S_1 et S_2 issus de la transformation des collaborations C_1 et C_2 par les transitions correspondantes obtenues par application de la règle 22.

$$\text{If } r \in PR(C_1) \text{ and } r \in PR(C_2) \text{ then } Transform(JoinNode, JoinState) \qquad \forall i = 1,2\,;$$

4 Étude de cas

On présente dans cette section une étude de cas. Cette étude est réalisée sur l'application de télémédecine décrite dans le projet TENEMO [29]. Classiquement, lorsqu'un cas d'AVC (accident vasculaire cérébrale) se présente dans un hôpital d'accueil, il est pris en charge par un urgentiste (HA). Ce dernier contacte le SAMU (service d'aide médicale d'urgence) via un médecin régulateur pour l'informer de l'admission aux urgences d'un malade qui présente des symptômes faisant suspecter un AVC (troubles du langage, de la vision, des déficits sensitivomoteur, des troubles de la coordination, etc.). Au niveau du système d'information du SAMU (EMS), le médecin régulateur crée un dossier médical (DM) en se basant sur un bilan initial composé du GCS (Glasgow Coma Score), de l'état hémodynamique, de l'heure d'apparition des signes, de l'heure de l'admission aux urgences, des antécédents, des données anamnestiques et de l'examen clinique, etc. Tout en restant en contact avec l'urgentiste, le régulateur du SAMU appelle le neurologue de garde du CHU (centre hospitalier universitaire UHC) et initie une conférence téléphonique à trois pour asseoir le diagnostic de l'AVC et pousser les investigations. Le cas échéant, le patient est transféré d'urgence vers le CHU dans une ambulance équipée SMUR (assistance avancée au patient ALS) ou une ambulance non équipée VLS (ACA), selon la situation du patient. La dérivation du comportement de chacun

des rôles du système est réalisée par un ensemble de règles de transformations en suivant les étapes mentionnées ci-dessous :

Étape 01 : Spécification du modèle des besoins globaux décrivant le comportement global du système. Ce modèle est une instance du méta-modèle des exigences de la figure (Fig. 3.2). Le comportement global du système est modélisé par un diagramme d'activités dont les activités de base sont des collaborations et des sous-collaborations (Fig. 3.12).

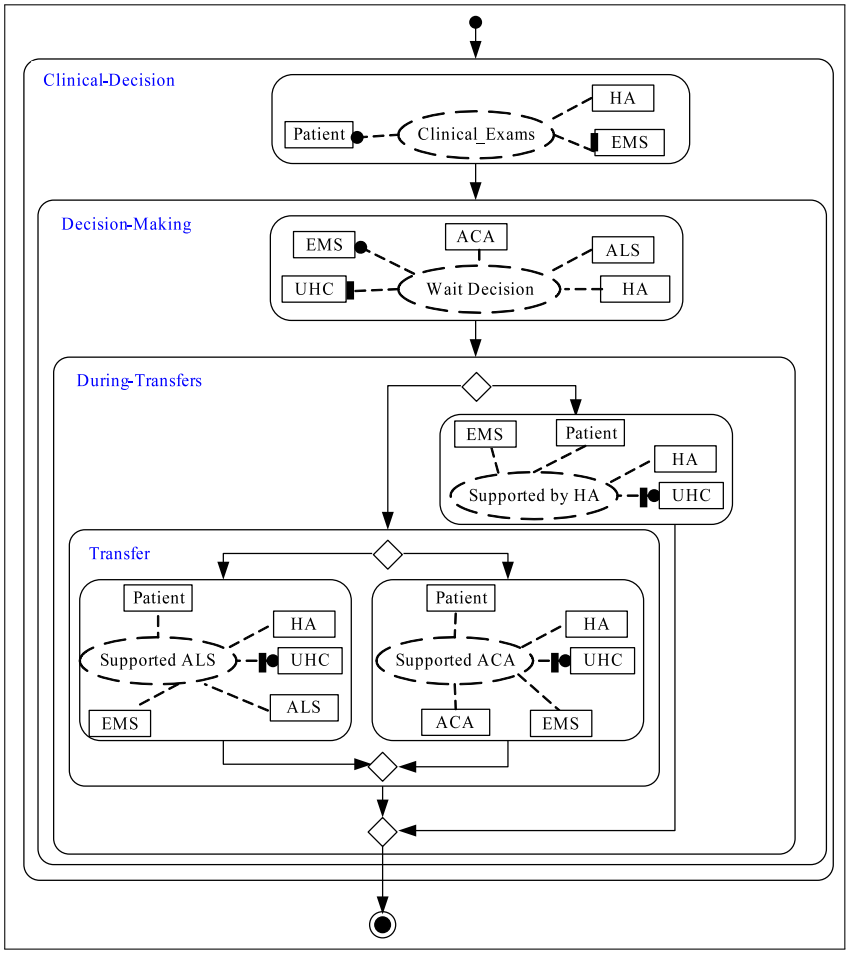

Fig. 3.12 : Le comportement global du système (modèle des exigences).

Le modèle du comportement global du système [35] est décrit par une collaboration composée de deux collaborations ou sous-collaborations selon le méta-modèle des exigences. Cette collaboration est une séquence entre la sous-collaboration *Clinical_Exams* et une collaboration com-

posite nommé *Decision-Making*. La collaboration *Decision-Making* est également une séquence composée de la sous-collaboration *Wait_Decision* et la collaboration *During-Transfers*. Cette dernière est définie par une structure de choix entre les collaborations : *Supported by HA* et *Transfer* qui est aussi une structure de choix. La collaboration *Transfer* définit un choix entre les deux sous-collaborations *Sending ACA* et *Sending ALS*.

Étape 02 : Cette étape consiste à appliquer les règles de transformations de modèles suivant l'algorithme du processus de dérivation au comportement global du système décrit par le modèle (Fig. 3.12). Ces règles sont appliquées afin de générer un automate d'états-transitions décrivant le comportement de chaque rôle du système. Nous considérons dans cet exemple les règles de transformation qui permettent de dériver les comportements du neurologue (CHU), du médecin urgentiste (HA) et du SMUR (ALS). Le processus de dérivation opérera sur le modèle global qui consiste en une séquence Weak entre la sous-collaboration *Clinical_Exams* et la collaboration *Decision-Making*.

Le neurologue (CHU) participe à la collaboration cible nommée *Decision-Making* et non pas à la sous-collaboration *Clinical_Exams*. Le processus de dérivation déclenche seulement la règle 5. La règle 5 effectue la transformation de la collaboration *Decision- Making*. Cette collaboration *Decision-Making* est également composée d'une séquence *Weak* entre la sous-collaboration *Wait_Decision* et la collaboration *During-Transfers*. Comme le neurologue participe à la fois aux collaborations source et cible, le processus de dérivation déclenche les règles 4 et 5. La règle 5 réalise la transformation des actions qu'il effectue au niveau de la sous-collaboration *Wait_Decision* et déclenche la transformation de la collaboration *During-Transfers*. La règle 4 effectue la transformation du controlflow en une transition reliant les états composites résultant de la transformation des collaborations source et cible à savoir la sous-collaboration *Wait_Decision* et la collaboration *During-Transfers*. La collaboration *During-Transfers* est une structure de choix entre la sous-collaboration *Supported by HA* et la collaboration *Transfer*. Le neurologue est responsable du choix dans cette structure, car il joue le rôle initiateur dans les deux collaborations. Le processus de dérivation déclenche alors les règles 6, 9, 10 et 11. La règle 6 réalise la transformation des actions qu'il effectue au niveau de la sous-collaboration *Supported by HA*, déclenche la transformation de la collaboration *Transfer* et le neurologue doit informer les rôles ne participant pas à la collaboration courante par l'envoi d'un message de coordination *Choicem* pour indiquer son choix. Par

exemple, au niveau de la sous-collaboration *Supported by HA*, le neurologue envoie un message de coordination *Choicem* aux rôles SMUR (ALS) et VLS (ACA), afin de leurs indiquer la sélection de *Supported by HA*. Les règles 9, 10 et 11 transforment les Controlflow et les nœuds de contrôle. La collaboration *Transfer* est également une structure de choix entre les sous-collaborations *Sending ACA* et *Sending ALS*. Comme le neurologue est un rôle initiateur dans les deux sous-collaborations, le processus de dérivation est similaire à celui de la collaboration *During-Transfers*. Les règles 6, 9, 10 et 11 sont déclenchées. La règle 6 effectue la transformation des actions effectuées par le neurologue au niveau de chaque sous-collaboration. Le neurologue informe les rôles ne participant pas à la collaboration courante en envoyant un message de coordination pour indiquer son choix. Le modèle décrit par un automate d'états-transitions représentant le comportement du neurologue (CHU) est présenté dans la figure (Fig. 3.13).

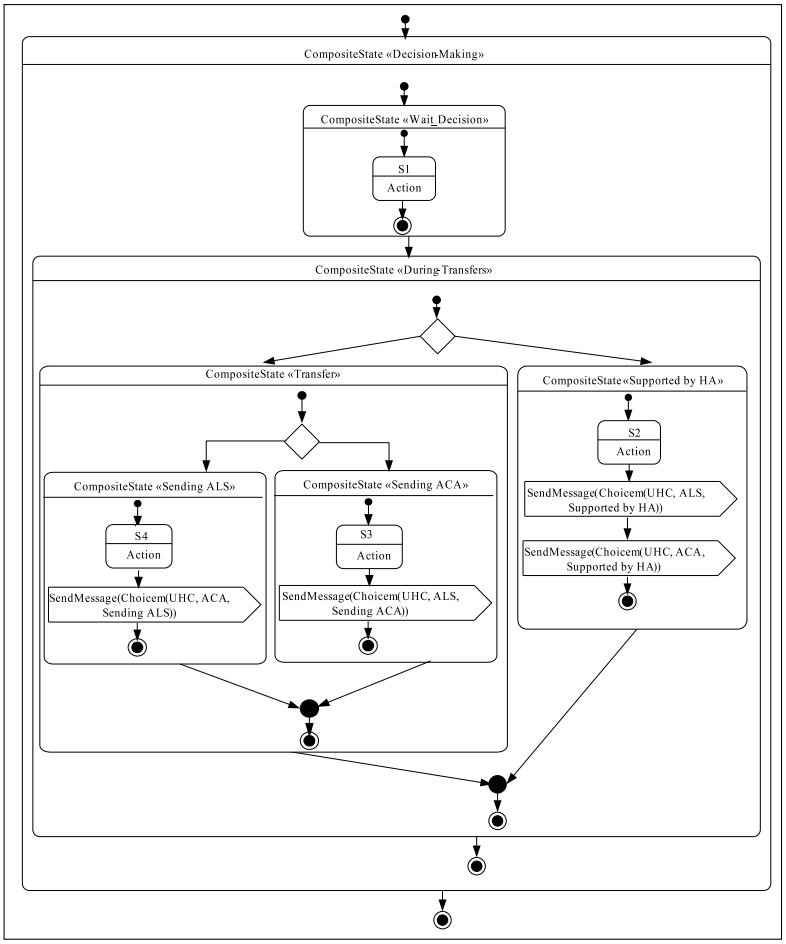

Fig. 3.13 : Comportement Dérivé du neurologue (CHU).

Pour le médecin urgentiste (HA), il participe dans les deux collaborations source et cible à savoir la sous-collaboration *Clinical_Exams* et la collaboration *Decision-Making*. Le processus de dérivation déclenche les règles 4 et 5. La règle des 5 réalise la transformation des actions accomplies par l'urgentiste au niveau de la sous-collaboration *Clinical_Exams* et déclenche la transformation de la collaboration *Decision-Making*. La règle 4 transforme le controlflow en une transition qui relie les états composites issus de la transformation des deux collaborations.

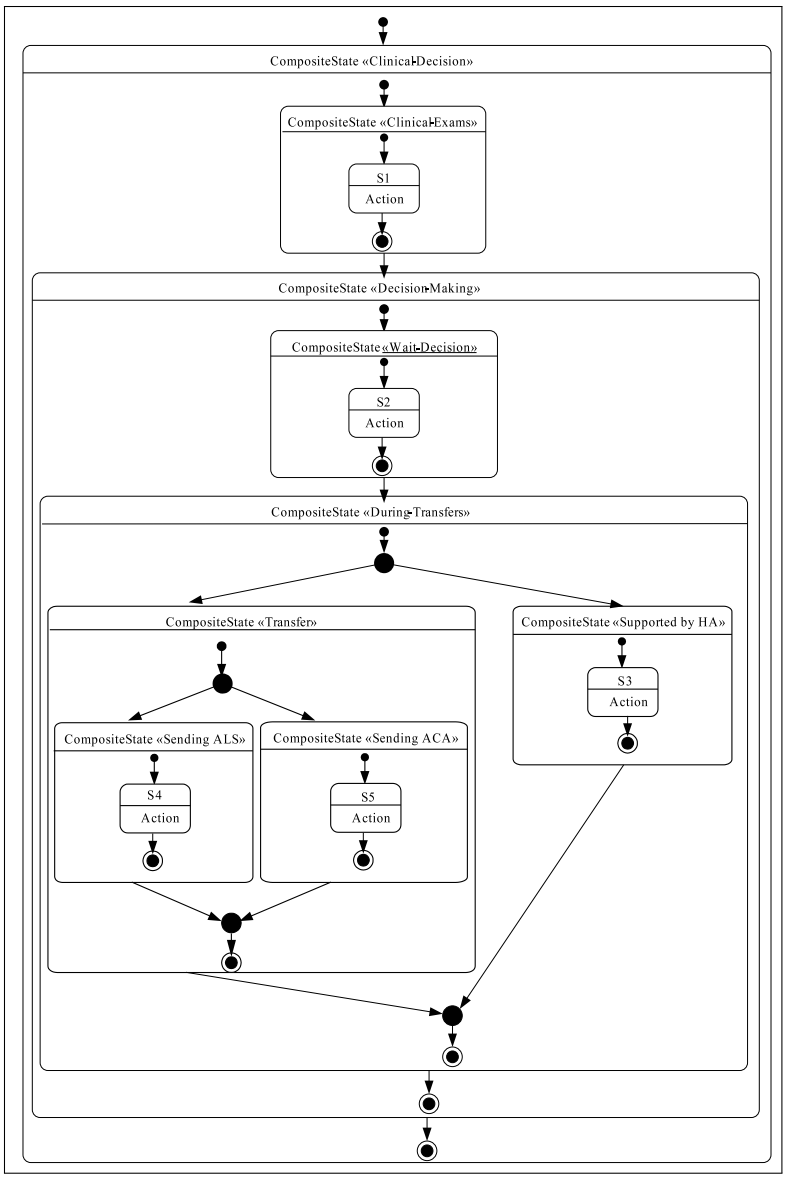

Fig. 3.14 : Comportement Dérivé du médecin urgentiste (HA).

La collaboration *Decision-Making* est transformée de la même manière que celle décrite dans la dérivation du comportement du neurologue à l'exception de la transformation de la collaboration *During-Transfers* qui est différente. Dans la collaboration *During-Transfers*, le médecin urgentiste participe dans *Supported by HA* et *Transfer*. Alors, le processus de dérivation déclenche les règles 7, 9, 10 et 11. La règle 7 réalise la transformation des activités du rôle au niveau de la sous-collaboration *Supported by HA* et réalise la transformation de la collaboration *Transfer*. Les règles 9, 10 et 11 transforment les Controlflow et les nœuds de contrôle. Le médecin urgentiste participe dans les deux sous-collaborations composant la collaboration *Transfer* ; le processus de dérivation est pratiquement similaire au précédent. Il déclenche les règles 7, 9, 10 et 11 qui transforment les actions accomplies par le rôle au niveau de chaque sous-collaboration, les Controlflow et les nœuds de contrôle. Le processus de dérivation génère un automate d'états-transitions représentant le comportement du médecin d'urgentiste (HA) (Fig. 3.14).

La dérivation du comportement du rôle SMUR (ALS) est obtenue de la même manière que celle décrite dans la dérivation du neurologue. Elle se résume dans la transformation de collaboration *Decision-Making* parce qu'il ne participe pas à la sous-collaboration *Clinical_Exams*. La collaboration *Decision Making* est composée d'une séquence *Weak* entre la sous-collaboration *Wait_Decision* et de la collaboration *During-Transfers*. Comme le SMUR participe à la fois aux collaborations source et cible, le processus de dérivation déclenche les règles 4 et 5. La règle 5 réalise la transformation des actions qu'il effectue au niveau de la sous-collaboration *Wait_Decision* et déclenche la transformation de la collaboration *During-Transfers*. La règle 4 effectue la transformation du controlflow en une transition reliant les états composites résultant de la transformation des collaborations source et cible. Au niveau *During-Transfers*, le SMUR participe dans la collaboration *Transfer*, mais il ne participe pas dans la sous-collaboration *Supported by HA*. Par conséquent, le processus de dérivation déclenche les règles 7, 8, 9, 10 et 11. La règle 7 réalise la transformation de la collaboration *Transfer*. La règle 8 permet au SMUR de recevoir un message coordination *Choicem* émis par le neurologue dans le cas où la sous-collaboration *Supported by HA* est choisie. La réception d'un tel message déclenche la poursuite du traitement au niveau de cette sous-collaboration. Les règles 9, 10 et 11 transforment les Controlflow et les nœuds de contrôle. La transformation de la collaboration *Transfer* qui est une structure de choix est réalisée d'une manière similaire à la précédente. Comme le SMUR participe à la sous-collaboration *Sending ALS* et ne participe pas dans *Sending ACA*, le processus de dérivation déclenche les règles 7, 8, 9, 10 et 11

qui permettent de réaliser la transformation des actions effectuées par le rôle en tant que participant à la sous-collaboration *Sending ALS*, permet la réception du message de coordination envoyé par le neurologue en cas de choix de la sous-collaboration *Sending ACA* et transforment les Controlflow et les nœuds de contrôle respectivement. Le processus de dérivation génère un automate d'états décrivant le comportement du SMUR (Fig. 3.15.).

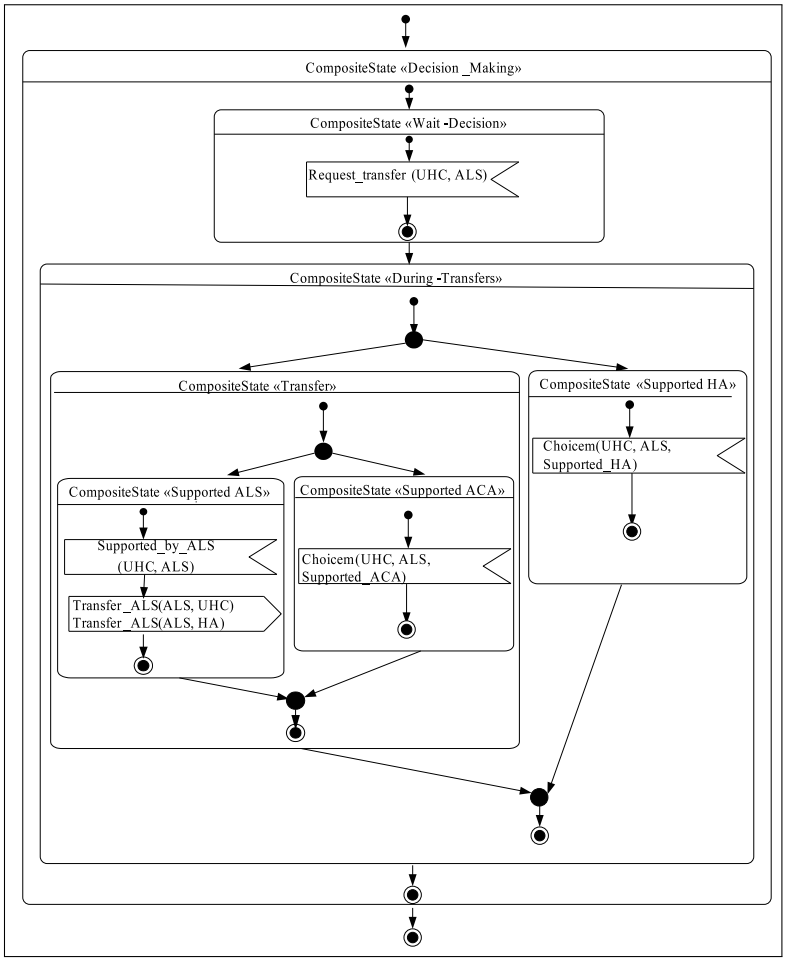

Fig. 3.15 : Comportement Dérivé du SMUR (ALS).

5 Conclusion

Dans ce chapitre, nous avons présenté une approche basée sur la transformation des modèles pour dériver le comportement des différents rôles d'un système distribué. Le comportement global du système est modélisé par le modèle des exigences. Les diagrammes d'activités étendus par les collaborations (extended UML activity diagrams) sont utilisés pour exprimer les modèles des exigences où les activités de base sont les collaborations entre les rôles du système. Ils permettent la représentation de concepts qui ne peuvent pas être décrits en utilisant des expressions de collaborations, comme c'est le cas dans [16]. C'est le cas des conditions intégrées dans les structures de choix et les structures de répétition. En outre, l'approche proposée considère également tous les nœuds de contrôle spécifiés dans les diagrammes d'activités UML, tels que le nœud de fusion et le nœud de jonction.

L'approche proposée consiste à dériver le comportement des rôles du système par transformation des modèles des exigences, qui décrivent le comportement fonctionnel d'un système d'une manière abstraite vers des modèles conceptuels où chaque rôle est identifié par son comportement local. Le processus de transformation, régi par un ensemble de règles, permet la dérivation du comportement local d'un rôle, sous la forme d'un automate d'états-transitions UML. Ce processus permet aussi la dérivation des messages de coordination appropriés entre les automates dérivés.

Cette approche commence par la présentation de l'architecture du processus de dérivation. L'architecture consiste en la définition du méta-modèle des exigences pour spécifier le comportement global d'un système donné, le méta-modèle cible des automates d'états-transitions UML afin de modéliser le comportement des rôles du système et l'établissement des règles de transformations qui régissent la transformation au cours du processus de dérivation. Par la suite, nous avons présenté l'algorithme du processus de transformation basé sur la transformation par méta-modèles ainsi que les règles de transformation pour les différents cas de chorégraphie recensés dans le méta-modèle des exigences.

Une étude de cas a été réalisée sur l'application de télémédecine décrite dans le projet TENEMO. Cette étude a permis d'appliquer les phases du processus de dérivation du comportement sur le modèle des exigences du système de télémédecine afin d'obtenir le comportement de chacun

des acteurs de l'application sous la forme d'un automate d'états-transitions. Cette étude, nous a permis de mettre en évidence les caractéristiques de l'approche proposée et son intérêt dans le développement des systèmes distribués. Un tel processus de dérivation automatique permet de rendre la conception de systèmes plus robustes et fiables en évitant les erreurs introduites par les activités de conception manuelle. Le chapitre 4 sera consacré à la mise en œuvre de l'approche de dérivation du comportement dirigée par les modèles.

Chapitre 4

Mise en œuvre de l'approche de dérivation du comportement d'un système

1 Introduction

L'approche Ingénierie Dirigée par les Modèles (IDM) fournit une infrastructure qui simplifie le développement des systèmes en augmentant le niveau d'abstraction. La représentation des connaissances impliquées dans le développement des applications apparaît sous formes de modèles et de méta-modèles. En plus de la définition des modèles, le processus de transformation de ces modèles est crucial pour l'approche IDM.

L'approche IDM ou MDA (Model Driven Architecture) est adoptée pour soutenir le développement des systèmes par la transformation de modèles en composants et applications exécutables. La motivation principale de l'IDM est de transférer le centre de travail de la programmation à une solution de modélisation en traitant des modèles. La transformation de modèles et le mapping entre les modèles sont les aspects principaux de l'approche IDM [6, 7, 50, 51]. Les mappings entre les modèles sont responsables de la transformation des éléments d'un modèle conforme à un méta-modèle source en éléments d'un autre modèle qui est conforme à un méta-modèle cible.

En réponse à ce besoin d'avoir une approche standard pour définir ces mappings entre les modèles, l'OMG a publié un appel à proposition pour le MOF 2.0 Query/ View/ Transformation (QVT) [71, 73]. Cet appel concernait la proposition d'un langage capable d'exprimer les requêtes, vues et transformations sur des modèles dans le contexte de l'architecture de modélisation MOF 2.0 [79]. Les différentes réponses données ont convergé vers la recommandation QVT adoptée en 2005 [78]. Le but principal du langage QVT est de développer un standard qui fournit des règles pour exprimer les transformations [51]. Une transformation en QVT décrit le rapport entre un méta-modèle source et un méta-modèle cible.

Dans ce contexte, plusieurs groupes de recherches travaillent sur leurs propres approches et langages de transformations de modèles. Nous citerons parmi ces langages de transformations de modèles proposés ; le langage ATL (ATLAS Transformation Language), le langage Tefkat, le langage MT (Model Transformation Language) et le langage AGG (Attributed Graph Grammar).

Le langage ATL [2, 46, 48] est un langage de transformation hybride. Il contient un mélange de constructions déclaratives et impératives. Les transformations ATL sont unidirectionnelles. Les transformations ATL opèrent sur des modèles sources en lecture seule et produisent des modèles cibles en écriture seule.

Le langage Tefkat [61] est un langage de transformation déclaratif. Les modèles sur lesquels opèrent les transformations Tefkat sont appelés *extents*. Ces derniers sont des ensembles d'éléments de modèles correspondant aux modèles d'entrée et de sortie. Les règles de transformation sont déclaratives et unidirectionnelles. Elles sont faiblement couplées : il n'existe pas de référence directe entre les règles. Le moteur de Tefkat peut être utilisé comme extension de la plateforme Eclipse. L'algorithme d'exécution s'inspire de l'unification de Prolog.

Le langage AGG [112, 113] est un environnement de développement pour les systèmes de transformation de graphes supportant une approche algébrique de la transformation de graphes. Cette approche est basée sur des fondements formels [30]. Les méta-modèles sources et cibles sont décrits par des graphes typés. Les transformations peuvent être exécutées dans deux modes différents : le mode interprétation qui correspond à l'application automatique des règles, le deuxième mode correspond au lancement explicite de l'application de chaque règle par l'utilisateur.

Le langage MT [114, 116, 117] est un dérivé de l'approche QVT-Partners [78]. Le langage MT est un langage de transformation de modèles unidirectionnel. Il est mis en œuvre comme un langage spécifique au domaine (DSL) avec le langage de programmation *Converge* [115]. Il permet à des langages de modélisation du même style qu'UML et leurs instances d'être exprimés.

Les langages de transformations de modèles ATL (ATLAS Transformation Language) et le langage QVT (Query View Transformation) de l'OMG exposent une architecture multicouches et partagent certaines caractéristiques communes [45, 47]. Nous présentons les langages de transformation QVT et ATL dans les sections suivantes.

2 Le langage de Transformation QVT

Le langage QVT (Queries, Views and Transformations) de l'OMG [78, 82, 111] s'adresse à une famille de problèmes relatifs au développement dirigé par les modèles. La spécification du standard QVT décrit trois langages pour les transformations : le langage *Relations*, le langage *Opérationnel Mappings*, et le langage *Core*. Ils sont organisés en une architecture présentée à la figure (Fig. 4.1). Le langage OCL 2.0 [76] est utilisé pour naviguer à travers les modèles. Les spécifications de QVT ont une nature hybride déclarative/impérative. Les langages *Relations* et *Core* sont tous deux déclaratifs et *Operational Mappings* est un langage impératif qui étend *Relations* et *Core*.

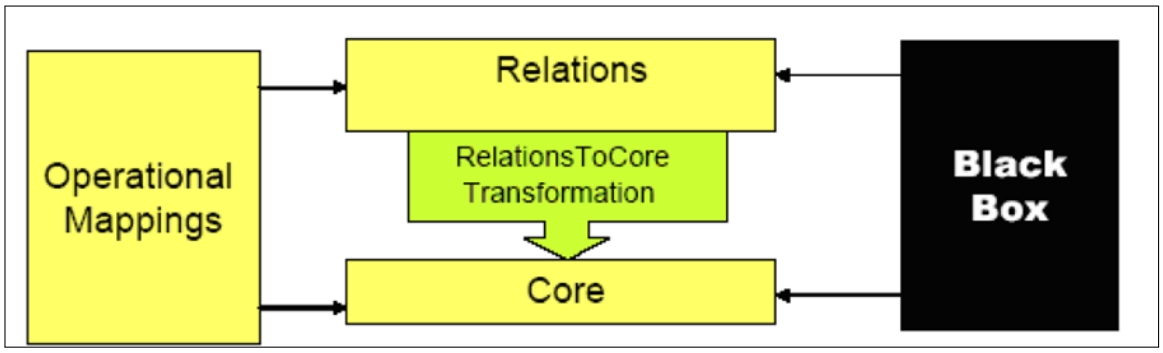

Fig. 4.1 : Architecture du langage QVT.

La figure (Fig. 4.2) décrit le contexte opérationnel de QVT. Le langage QVT est un méta-modèle conforme au MOF 2.0. C'est un langage de transformation par méta-modèles.

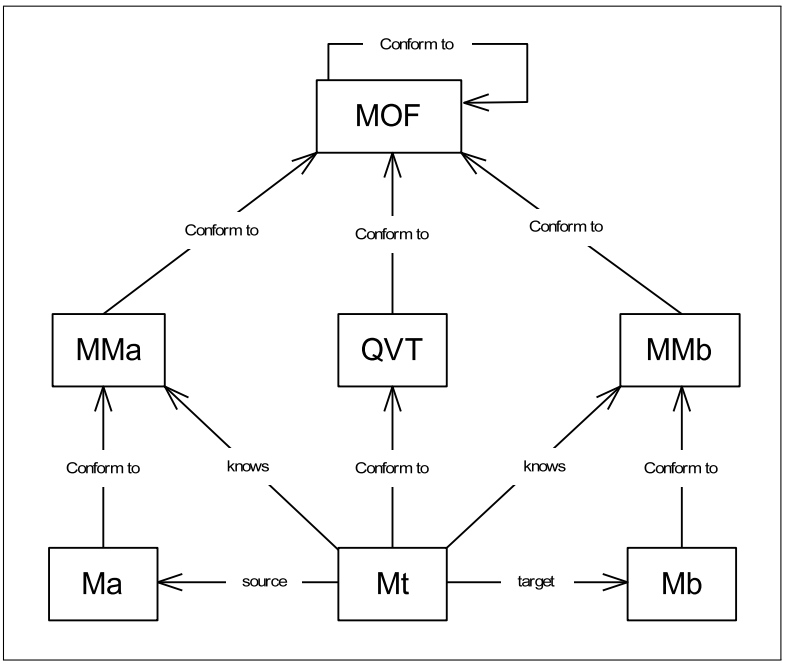

Fig. 4.2 : Contexte opérationnel de QVT.

Le langage *Relations* offre la possibilité de spécifier des transformations de modèles. Il supporte le pattern matching complexe des objets, et crée implicitement des classes de trace et leurs instances pour enregistrer ce qui s'est produit pendant une exécution de transformation (traçabilité). Dans le langage de relations, une transformation entre les modèles candidats est indiquée comme un ensemble de relations qui doivent se tenir pour que la transformation réussisse.

Le langage *Core* est un petit langage de modèles. Il est déclaratif et plus simple que le langage *Relations*. Il supporte seulement le pattern matching d'un ensemble de variables en évaluant des conditions sur ces variables par rapport à un ensemble de modèles. Il traite tous les éléments modèles sources, cibles, et modèles de trace systématiquement. Il est également aussi puissant que le langage *Relations*, et en raison de sa simplicité relative, sa sémantique peut être définie plus simplement, bien que les descriptions de transformation décrites en utilisant le *Core* soient donc plus complexes.

En plus des langages déclaratives *Relations* et *Core* qui incarnent la même sémantique à deux niveaux d'abstraction différents, QVT propose deux mécanismes pour étendre les langages

Relations et *Core* : un langage appelé *Operational Mappings* et un mécanisme pour invoquer des implémentations impératives de transformations non standard du *Black-Box*.

Le langage *Operational Mappings* est spécifié d'une manière standard afin de fournir des implémentations impératives afin d'étendre le langage *Relations*. Il fournit des constructions OCL qui permettent un style plus procédural, et une syntaxe concrète qui semble bien familière aux programmeurs impératifs. Des opérations de mappings peuvent être employées pour implémenter une ou plusieurs relations à partir des spécifications de relations.

Le mécanisme *Black-Box* permet d'intégrer et d'exécuter du code externe au cours de l'exécution d'une transformation. Il permet ainsi à des algorithmes complexes d'être codés en n'importe quel langage de programmation avec une attache de MOF. Comme il permet l'utilisation des bibliothèques spécifiques de domaine pur. Par exemple, mathématique, ingénierie, bio-science, et beaucoup d'autres domaines qui ont de grandes bibliothèques qui codent des algorithmes spécifiques aux domaines qui sont difficiles, ou impossible à exprimer en utilisant OCL.

3 Le langage de Transformation ATL

Le langage ATL (ATLAS Transformation Language) [2, 46, 48] est défini dans le contexte des langages de transformation par méta-modèles présenté dans la (Fig. 4.3). L'opération de transformation est dirigée par un programme ATL nommé *mma2mmb.atl*. Ce programme transforme le modèle source *Ma* en un modèle cible *Mb*. Il est lui-même un modèle. Les modèles *Ma*, *Mb* et le programme sont conformes aux méta-modèles *MMa*, *MMb* et *ATL* respectivement. Ces méta-modèles sont conformes au méta-méta-modèle *MOF*.

3.1 Structure globale des programmes de transformation

En ATL, une transformation s'appelle un module. Un module contient une en-tête, un ensemble d'importation de bibliothèques et de fonctions, un ensemble de helpers et de règles de transformation.

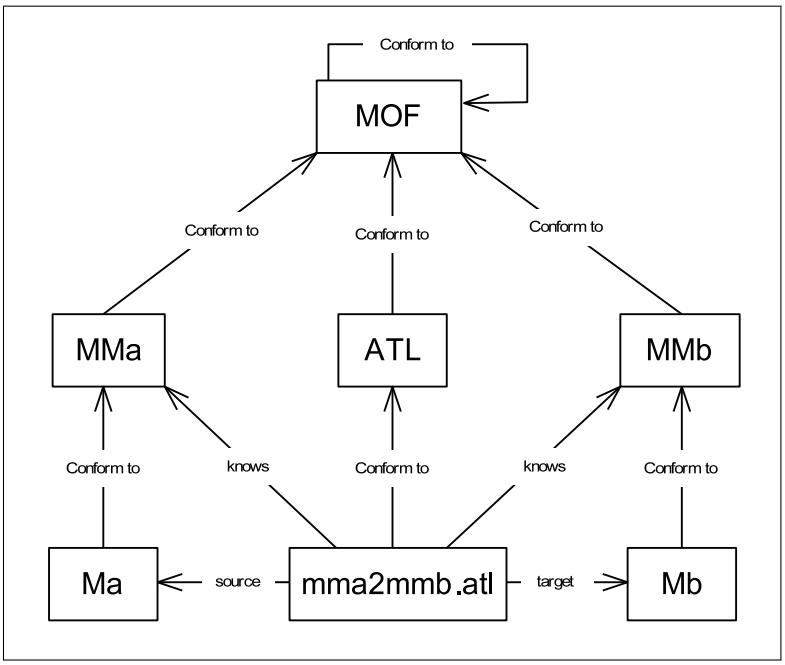

Fig. 4.3 : L'approche de transformation par ATL.

L'en-tête donne le nom du module de la transformation et déclare les modèles sources et cibles. Le mot-clé *create* indique les modèles cibles. Le mot-clé *from* indique les modèles sources. Les modèles sources et cibles sont respectivement conformes à leurs méta-modèles.

```
module module_name;
create output_models [from|refining] input_models;
```

3.1.1 Helpers

Les fonctions ATL sont appelées *helpers*. Un *helper* peut être vue comme une méthode Java. Il sert à factoriser du code ATL. Il est basé sur le langage OCL. Un helper peut être appelé à partir de n'importe quel point d'une transformation ATL. Il existe deux types de helpers : Opération (*helper fonctionnel* référencié en général par *helper*) et Attribut (*helper attribut* référencié par attribut). Un *helper* peut être spécifié dans le contexte d'un type OCL (par exemple, String ou Integer) ou d'un type source (venant de l'un des méta-modèles sources).

Les helpers opération et attribut sont utilisés pour définir des opérations dans le contexte d'un élément de modèle ou du module de transformation. Le rôle principal des helpers est de réaliser la navigation des modèles sources. Ils ont un nom, un contexte et un type. Leur valeur est définie par une expression OCL. Ils peuvent être récursifs et retournent toujours la même valeur pour un contexte et un ensemble d'arguments donnés. Les helpers opération peuvent avoir des paramètres à la différence des helpers attribut qui sont définis sans paramètres.

```
helper [context context_type] def:helper_name(parameters)
                                  :return_type=exp;
```

3.1.2 Règles de transformation

La règle de transformation permet d'exprimer une transformation de modèles en ATL. Les règles ATL peuvent être déclaratives, appelées *Matched rules* ou impératives *Lazy rules* et *Called rules*.

3.1.2.1 Règles déclaratives (matched rules) : Une *matched rule* permet de spécifier la manière avec laquelle les éléments du modèle cible sont générés à partir des éléments du modèle source. Donc, une *matched rule* spécifie : (i) l'élément du modèle source à comparer, (ii) le nombre et le type des éléments du modèle cible à générer et (iii) la manière d'initialiser les éléments du modèle cible à partir des éléments du modèle source. Une règle est conforme à la syntaxe suivante :

```
rule rule_name {
  from   in_var : in_type [( condition)]
  [using {var1 : var_type1 = init_exp1;
                  ...
          varn : var_typen = init_expn; }]
  to
          out_var1 : out_type1 (bindings1),
          out_var2 : distinct out_type2
                     foreach(e in collection)(bindings2),
                     ...
```

```
        out_varn : out_typen (bindingsn)
[do  { statements }]
}
```

Chaque règle est identifiée par son nom. Elle doit être unique dans un module de transformation. Une *matched rule* est composée d'un motif source et d'un motif cible. Le motif source d'une règle définit un élément du modèle source et une garde sous la forme d'une expression OCL booléenne. Le motif cible est composé d'un ensemble d'éléments. Chacun de ces éléments définit un type cible et un ensemble d'affectations appelées *bindings*. Un *binding* fait référence à une propriété (attribut ou référence) du type cible et spécifie une expression dont la valeur est utilisée pour initialiser la propriété. La section optionnelle *using* permet de déclarer des variables locales. Ces variables peuvent être utilisées dans la section *using* elle-même comme dans les sections *to* et *do*. Chaque variable est identifiée par son nom et son type, et initialisée par une expression OCL. La dernière section d'une règle matched est le bloc impératif *do*. Le bloc impératif contient une séquence d'instructions impératives. Il peut être utilisé à la place de ou en combinaison avec un motif cible, dans des règles déclaratives ou impératives. Les instructions disponibles en ATL sont celles communément offertes par les langages impératifs (instruction d'affectation, instruction conditionnelle, instructions de boucles, etc.). Ce bloc peut être utilisé pour initialiser certains attributs des éléments du modèle non encore initialisés.

3.1.2.2 Règles impératives (Lazy rules) : Elles sont définies comme les règles matched mais elles sont déclenchées par d'autres règles. Elles sont appliquées sur chaque tuple autant de fois qu'elles sont référencées par les autres règles.

3.1.2.3 Règles impératives (Unique Lazy rules) : Elles sont aussi déclenchées par d'autres règles. Mais elles ne sont appliquées qu'une seule fois sur chaque tuple.

3.1.2.4 Règles impératives (Called rules) : ATL définit les *called rules* afin de permettre la génération d'éléments du modèle à partir du code impératif. Ce type de règles peut être explicitement déclenché à partir d'un bloc impératif à l'exception de la règle *entrypoint called rule*. Une

called rule est conforme à la syntaxe suivante :

```
[entrypoint]rule rule_name (parameters){
    [using {var1 : var_type1 = init_exp1;
                    ...
            varn : var_typen = init_expn; }]
    to
            out_var1 : out_type1 (bindings1),
            out_var2 : distinct out_type2
                       foreach(e in collection)(bindings2),
                    ...
            out_varn : out_typen (bindingsn)
    [do  { statements }]
    }
```

Les règles *Called rules* sont similaires à des procédures. Elles sont invoquées par leur nom et peuvent prendre des arguments. Elles ne possèdent pas de motif source. La règle *entrypoint called rule* n'a pas besoin d'être appelée comme les autres mais elle est invoquée au début de l'exécution de la transformation.

ATL et QVT partagent certaines caractéristiques communes, comme ils ont d'abord partagé le même ensemble d'exigences définies dans QVT RFP. Les deux langages ATL et QVT ont un contexte opérationnel similaire montré dans la figure (Fig. 4.4), un programme de transformation *Mt* qui se traduit par la transformation automatique du modèle *Ma* en un modèle *Mb*. Ces trois modèles sont conformes aux méta-modèles *MMt*, *MMa* et *MMb* respectivement. Ces derniers sont conformes au méta-modèle *MOF*. *MMt* correspond à la syntaxe abstraite des langages de transformation. QVT et ATL fournissent leur propre méta-modèle définissant leurs syntaxes abstraites et se placent dans le contexte de *MMt*. Ils sont tous deux conformes au *MOF*.

Les deux langages QVT et ATL peuvent être alignés sur plusieurs catégories (abstraction, paradigmes, directivité, cardinalité, etc.) [45, 47]. Les résultats de l'analyse ont permis de réaliser une certaine interopérabilité entre les deux langages. Cette interopérabilité se traduit par l'exécution des programmes ATL sur les moteurs de QVT et réciproquement des programmes QVT sur la

machine virtuelle ATL. Par conséquent, nous avons choisi d'exprimer les règles de transformation de modèles dans le langage de transformation ATL [46]. Dans la section suivante nous présentons la mise en œuvre de l'approche de dérivation du comportement d'un système par transformation des modèles des exigences décrite dans le chapitre 3.

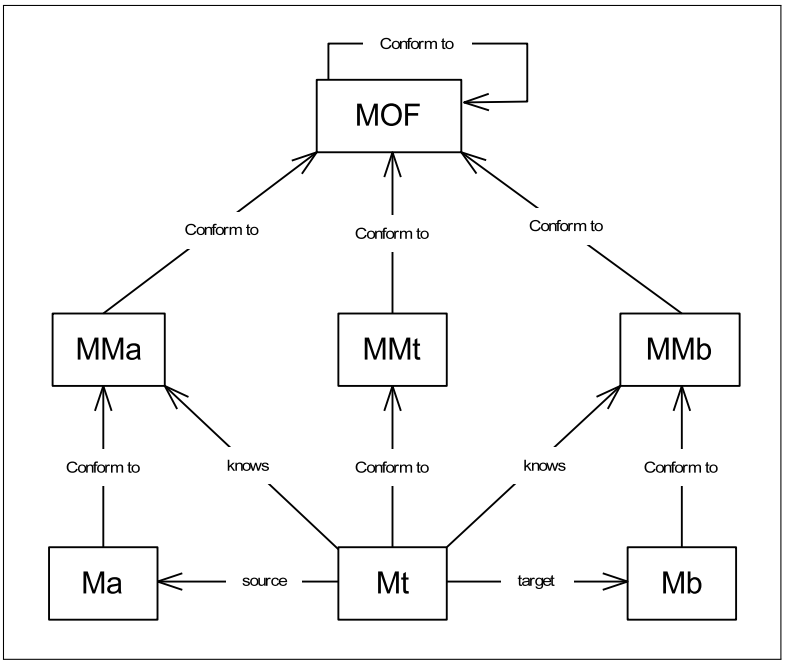

Fig. 4.4 : Le processus de transformation de Modèles.

4 Mise en œuvre de l'approche de transformation des modèles des exigences

La mise en œuvre du processus de dérivation adopté (Fig. 3.1) exige :

(i) la représentation du méta-modèle des exigences, du méta-modèle cible et du modèle des exigences décrivant le comportement global du système instance du méta-modèle des exigences ; et

(ii) l'application des règles de transformation de modèles spécifiées dans le langage ATL sur

le modèle source. Cette phase est gérée par l'algorithme de transformation proposé dans le chapitre 3. Ce processus génère des machines à états-transitions décrivant le comportement de chaque rôle du système. Le modèle du comportement dérivé est conforme au méta-modèle cible.

Nous présentons ci-dessous le code de quelques règles à titre d'exemple et plus précisément les règles de transformation pour les cas de chorégraphie suivants : séquence Strong, Weak et structure de choix. On distingue deux catégories de règles ; celles définies au niveau d'une sous-collaboration et celles définies pour une collaboration composée. Ces dernières règles sont similaires aux premières, mais elles sont définies d'une manière récursive et font appel à d'autres règles afin d'achever la transformation. Ceci est dû à la composition des collaborations conformément au méta-modèle des exigences.

Règle 1 : If $r \in TR(C_1)$ then $Transform(C_1, S_1, Send(Flowm(r, r', S_2)))$ $\forall r' \in (SR(C_2) - r)$;

Le code ATL (Listing 4.1) de cette règle au niveau d'une sous-collaboration est exécuté si et seulement si les helpers *IsinTermine()* et *CibleIsStrong()* sont évalués à vrai. Ces helpers déterminent si le rôle est terminateur dans la collaboration source. Au niveau d'une collaboration, un code similaire pour la règle est défini (Listing 4.2) mais dans ce cas, la règle fait appel à d'autres règles afin d'achever la transformation de la collaboration.

```
rule CollaborationS2StateT {

from s:MMS!Sub_collaboration (s.IsinTermine() and s.CibleIsStrong())

to    r: MMt!CompositeState(Name<-s.Name, outgoing<-s.Getoutgoing(),
      incoming<-s.Getincoming(), states<-Si, transition<-ti, states<-st,
      states<-sc, transition <-t, transition <-tf, states<-Sf ),
   Si: MMt!InitialState(outgoing<-ti),
   ti: MMt!Transition ( source <-Si, target <- sc),
   sc: MMt!State (action<-act, incoming <-ti, outgoing <-t),
   act: MMt!DomainAction(Name<-s.Name),
   t: MMt!Transition (source<-sc, target<-st),
   st: MMt!State (incoming<-t, outgoing<-tf),
   tf: MMt!Transition (source<-st, target<-Sf),
    Sf:MMt!FinalState (incoming<-tf)

do {thisModule.i<-thisModule.i+1; sc.Name<-'S'+thisModule.i.toString();
   thisModule.i<-thisModule.i+1; st.Name<-'S'+thisModule.i.toString();
```

```
    thisModule.j<-thisModule.j+1; ti.Name<-'T'+ thisModule.j.toString();
    thisModule.j<-thisModule.j+1; t.Name<-'T'+ thisModule.j.toString();
    thisModule.NameCol <- s.Name;
    st.action<-s.GetRolesInit()   ->select(e| e.Name <> thisModule.role)
                                  ->collect(x| thisModule.SendFlowm(x));
  thisModule.j<-thisModule.j+1; tf.Name<-'T'+thisModule.j.toString()}}
```

Listing 4.1 : Code ATL de la Règle 1 au niveau d'une sous-collaboration.

Cette règle consiste à :

- Générer un état composite S dans le modèle RolderivBeh(r) conforme au méta-modèle cible MM_t dont S.Name= C_1.Name ;
- Générer (un état initial) dans S ;
- Accomplir la transformation des actions de la sous-collaboration C_1 ;
- Générer les actions d'envoi de messages de coordination aux rôles initiateurs de la collaboration C_2 dans S (à l'exception de soi même). L'ensemble des rôles initiateurs de la collaboration C_2 est calculé par le helper *GetRolesInit()* ;

```
helper context MMs!Action def: GetRolesInit():
OrderedSet(MMs!Role)=self.ColCibleS().GetSRCol();
```

- Générer les transitions pour relier les différents états dans l'état composite S ;
- Générer (un état final) dans S ;
- Connecter S à ses états prédécesseurs et successeurs.

```
rule Collaboration2StateT {

from s:MMS!collaboration (s.IsTermineCol())

to   r:  MMt!CompositeState(Name<-s.Name, outgoing<-s.Getoutgoing(),
     incoming<-s.Getincoming(), states<-Si, transition<-ti, states<-st,
     states<-sc, transition <-t, transition <-tf, states<-Sf ),
   Si: MMt!InitialState(outgoing<-ti),
```

```
    ti: MMt!Transition ( source <-Si, target <- sc),
    sc: MMt!State( action<-act,incoming <-ti,outgoing <-t,
         states<-s.controlnodes,   states<-s.Getsub(),
         transition<- s.Getrelation()),
    t: MMt!Transition (source<-sc,target<-st),
    st: MMt!State (incoming<-t,outgoing<-tf),
    tf: MMt!Transition (source<-st,target<-Sf),
    Sf:MMt!FinalState (incoming<-tf)

do {thisModule.i<-thisModule.i+1;sc.Name<-'S'+thisModule.i.toString();
    thisModule.i<-thisModule.i+1;st.Name<-'S'+thisModule.i.toString();
    thisModule.j<-thisModule.j+1;ti.Name<-'T'+ thisModule.j.toString();
    thisModule.j<-thisModule.j+1;t.Name<-'T'+ thisModule.j.toString();
    thisModule.NameCol <- s.Name;
    st.action<-s.GetRolesInit()   ->select(e| e.Name <> thisModule.role)
                                  ->collect(x| thisModule.SendFlowm(x));
  thisModule.j<-thisModule.j+1;tf.Name<-'T'+thisModule.j.toString()}}
```

Listing 4.2 : Code ATL de la Règle 1 au niveau d'une collaboration.

Cette règle consiste à :

- Générer un état composite S dans le modèle RolderivBeh(r) conforme au méta-modèle cible MM_t dont S.Name= C_1.Name ;
- Générer (un état initial) dans S ;
- Accomplir la transformation de la collaboration C_1 en déclenchant la transformation de ses composantes ;
- Générer les actions d'envoi de messages de coordination aux rôles initiateurs de la collaboration C_2 dans S (à l'exception de soi même). L'ensemble des rôles initiateurs de la collaboration C_2 est calculé par le helper *GetRolesInit()* ;
- Générer les transitions pour relier les différents états dans l'état composite S ;
- Générer (un état final) dans S ;
- Connecter S à ses états prédécesseurs et successeurs.

Règle 2 : If $r \in SR(C_2)$ then $Transform(C_2, S_2, Receive(Flowm(r', r, S_2)))$ $\forall r' \in (TR(C_1) - r)$;

Le code ATL (Listing 4.3) de cette règle au niveau d'une sous-collaboration est exécuté si et seulement si les helpers *IsinInitial()* et *SourceIsStrong()* sont évalués à vrai. Ces helpers déterminent si le rôle est initiateur dans la collaboration cible.

```
rule CollaborationS2StateI {

from s:MMS!Sub_collaboration (s.IsinInitial() and s.SourceIsStrong())

to   r: MMt!CompositeState(Name<-s.Name, outgoing<-s.Getoutgoing(),
      incoming<-s.Getincoming(), states<-Si, transition<-ti, states<-st,
      states<-sc, transition <-t, transition <-tf, states<-Sf ),
    Si: MMt!InitialState(outgoing<-ti),
    ti: MMt!Transition ( source <-Si, target <- sc),
    sc: MMt!State (incoming <-ti, outgoing <-t),
    t: MMt!Transition (source<-sc, target<-st),
    st: MMt!State (action<-act, incoming<-t, outgoing<-tf),
    act: MMt!DomainAction(Name<-s.Name),
    tf: MMt!Transition (source<-st, target<-Sf),
    Sf:MMt!FinalState (incoming<-tf)

do {thisModule.i<-thisModule.i+1;sc.Name<-'S'+thisModule.i.toString();
    thisModule.NameCol <- s.Name;
    st.action<-s.GetRolesTermin()->select(e| e.Name <> thisModule.role)
                                 ->collect(x| thisModule.ReceptFlowm(x));
    thisModule.i<-thisModule.i+1;st.Name<-'S'+thisModule.i.toString();
    thisModule.j<-thisModule.j+1;ti.Name<-'T'+ thisModule.j.toString();
    thisModule.j<-thisModule.j+1;t.Name<-'T'+ thisModule.j.toString();
    thisModule.j<-thisModule.j+1;tf.Name<-'T'+thisModule.j.toString()}}
```

Listing 4.3 : Code ATL de la Règle 2 au niveau d'une sous-collaboration.

Cette règle consiste à :

- Générer un état composite S dans le modèle RolderivBeh(r) conforme au méta-modèle cible MM_t dont S.Name= C_2.Name ;
- Générer (un état initial) dans S ;
- Générer les actions de réception de messages de coordination provenant des rôles terminateurs de la collaboration C_1 dans S (à l'exception de soi même). L'ensemble des rôles terminateurs de la collaboration C_1 est calculé par le helper *GetRolestermin()* ;

```
helper context MMs!Action def: GetRolesTermin():
OrderedSet(MMs!Role) = self.ColSourceP().GetTRCol();
```

- Accomplir la transformation des actions de la sous-collaboration C_1 ;
- Générer les transitions pour relier les différents états dans l'état composite S ;
- Générer (un état final) dans S ;
- Connecter S à ses états prédécesseurs et successeurs.

Les opérations d'envoi et de réception de messages au niveau des règles 1 et 2 sont réalisées par l'appel à des Lazy rules *SendFlowm* et *ReceptFlowm* respectivement (Listing 4.4) :

```
lazy rule SendFlowm {

from s:MMs!Role

to actm: MMt!SendSignalAction ( message <- mes),
   mes: MMt!Message (Name<-'Flowm',
        Expediteur<-thisModule.role, Recepteur<-s.Name,
        Collaboration <-thisModule.NameCol)}


Lazy rule ReceptFlowm {

from s: MMs!Role

to actm: MMt!AcceptSignalAction ( message <- mes),
   mes: MMt!Message (Name<-'Flowm', Expediteur<-s.Name,
    Recepteur<-thisModule.role, Collaboration <-thisModule.NameCol)}
```

Listing 4.4 : Code ATL des règles d'envoi et de réception de messages Flowm.

Règle 3 : If $r \in (PR(C_1) - TR(C_1))$ or $r \in (PR(C_2) - SR(C_2))$ then $Transform(C_i, S_i)$ $\forall i = 1,2;$

Le code ATL de cette règle (Listing 4.5) au niveau d'une sous-collaboration est similaire aux codes des règles 1 et 2 sauf qu'il ne comporte pas d'actions d'envoi ou de réception de messages. Il est exécuté si et seulement si le helper *IsinParticipCol()* (Listing 4.6) est évalué à vrai. Ce helper détermine si le rôle est un rôle participant ne faisant pas partie des rôles initiateurs ou terminateurs des collaborations source ou cible.

```
rule CollaborationS2StateP {

from s:MMS!Sub_collaboration (s.IsinParticipCol())

to r:   MMt!CompositeState(Name<-s.Name, outgoing<-s.Getoutgoing(),
     incoming<-s.Getincoming(), states<-Si, transition<-ti,
     states<-sc, transition <-tf, states<-Sf ),
    Si: MMt!InitialState(outgoing<-ti),
    ti: MMt!Transition ( source <-Si, target <- sc),
    sc: MMt!State (action<-act, incoming <-ti, outgoing <-tf),
    act: MMt!DomainAction(Name<-s.Name),
    tf: MMt!Transition (source<-sc, target<-Sf),
    Sf:MMt!FinalState (incoming<-tf)

do {thisModule.i<-thisModule.i+1;sc.Name<-'S'+thisModule.i.toString();
    thisModule.j<-thisModule.j+1;ti.Name<-'T'+ thisModule.j.toString();
    thisModule.j<-thisModule.j+1;tf.Name<-'T'+thisModule.j.toString()}}
```

Listing 4.5 : Code ATL de la Règle 3 au niveau d'une sous-collaboration.

```
helper context MMs!Collaboration def:ISinParticipCol(): Boolean =
if not self.IsRacine() then
  (self.IsinPR() and not self.IsinTermine() and self.CibleIsStrong())
       or
  (self.IsinPR()and not self.IsinInitial() and self.SourceIsStrong())
       or
  (self.IsWeak() and self.IsinPR() )
else false
endif;
```

Listing 4.6 : Code ATL du helper IsinParticipCol().

Règle 4 : If $r \in (PR(C_1)$ and $PR(C_2))$ then $Transform(Controlflow, Transition)$;

Le code ATL de cette règle (Listing 4.7) est exécuté si et seulement si le helper *IsTransform()* est évalué à vrai. Ce helper détermine si le controlflow est transformable, autrement dit si le rôle participe dans les deux collaborations source et cible.

```
rule Controlflow2Transition{from s: MMs!Controlflow (s.IsTransform())
to   r: MMt!Transition( source<- s.source, target<- s.target)
do {thisModule.j<-thisModule.j+1; r.Name<-'T'+ thisModule.j.toString()}}
```

Listing 4.7 : Code ATL de la règle 4.

Règle 5 : If $r \in PR(C_i)$ then $Transform(C_i, S_i)$ $\forall i = 1,2;$

La règle 5 possède le même code que la règle 3 parce que sa condition de déclenchement est incluse dans le helper *IsinParticipCol()*.

Règle 6 : If $r \in SR(C_i)$ then $Transform(C_i, S_i, Send(Choicem(r, r', S_i)))$

$$\forall r' \in (PR(C_{i'}) - PR(C_i)) \qquad \forall i, i' = 1,2 \text{ et } i \neq i';$$

Le code ATL (Listing 4.8) de cette règle au niveau d'une sous-collaboration est :

```
Rule   CollaborationS2StateSPR {

from C: MMs!Collaboration( C.Choice_C12() )

to S: MMt!CompositeState (Name<- C.Name, outgoing<-C.outgoing,
     incoming<-C.incoming, states<-Si, transition<-ti, states<-sc,
     transition<-t, states<-st, transition <-tf, states <-Sf ),

         Si: MMt!InitialState(outgoing <- ti),
   ti: MMt!Transition ( source <-Si, target <- sc),
   sc: MMt!State(action<-act, incoming<-ti, outgoing <-t),
   act: MMt!DomainAction (Name<-C.Name),
   t: MMt!Transition ( source <-sc, target <- st),
   st: MMt!State ( incoming <-t, outgoing<-tf),
   tf: MMt!Transition ( source <-st, target <- Sf),
   Sf: MMt!FinalState (incoming <-tf)

do {thisModule.i<-thisModule.i+1; sc.Name<-'S'+thisModule.i.toString();
   thisModule.i<-thisModule.i+1; st.Name<-'S'+thisModule.i.toString();
   thisModule.NameCol<-C.Name;
   st.action <-C.GetRolesC2()->collect(x|thisModule.SendChoicem(x));
   thisModule.j<-thisModule.j+1; ti.Name<-'T'+thisModule.j.toString();
   thisModule.j<-thisModule.j+1; t.Name<-'T'+thisModule.j.toString();
  thisModule.j<-thisModule.j+1; tf.Name<-'T'+thisModule.j.toString();}}
```

Listing 4.8 : Code ATL de la règle 6 au niveau d'une sous-collaboration.

Ce code est exécuté si et seulement si le helper *Choice_C12()* est vrai. Ce helper (Listing 4.9) détermine si le rôle est un rôle initiateur dans la sous-collaboration C_1.

```
helper context MMs!Action def: Choice_C12(): Boolean =

if self.SourceIsDN1() then if self.IsinPR()then
                           if self.IsinInitial() then true
                           else false
                           endif
                      else false
                      endif
else false
endif;
```

Listing 4.9 : Code ATL du helper Choice_C12().

Cette règle consiste à :

- Générer un état composite S dans le modèle RolderivBeh(r) conforme au méta-modèle cible MM_t dont S.Name= C_1.Name ;
- Générer (un état initial) dans S ;
- Accomplir la transformation des actions de la sous-collaboration C_1 ;
- Générer les actions d'envoi de messages de coordination *Choicem* aux rôles ne participant pas dans C_1 mais participant dans C_2 au niveau de S (à l'exception de soi même). L'ensemble de ces rôles est calculé par le helper *GetRolesC2()* ;

```
helper context MMs!Action def: GetRolesC2():OrderedSet(MMs!Role)
  = self.GetSecond().GetPRCol()->asSet()->
    select (r | not(self.GetFirst().GetPRCol()->includes(r)));
```

- Générer les transitions pour relier les différents états dans l'état composite S ;
- Générer (un état final) dans S ;
- Connecter S à ses états prédécesseurs et successeurs.

Règle 7 : If $r \in PR(C_i)$ and $r \notin SR(C_i)$ then $Transform(C_i, S_i)$ $\forall i = 1,2$;

La règle 7 possède le même code que la règle 5 avec une condition de déclenchement distincte. Elle est déterminée par les helpers *ChoicePR_C12()* et *ChoicePR_C21()* exprimée par le code du (Listing 4.10).

```
helper context MMs!Action def:ChoicePR_C12():Boolean=
if self.SourceIsDN1() then
     if self.IsinPR() then if not self.IsinInitial() then true
                           else false
                           endif
     else false
     endif
else false
endif;


helper context MMs!Action def:ChoicePR_C21():Boolean=
if self.SourceIsDN2() then
     if self.IsinPR() then if not self.IsinInitial() then true
                           else false
                            endif
     else false
     endif
else false
endif;
```

Listing 4.10 : Code ATL des helpers ChoicePR_C12() et ChoicePR_C21().

Règle 8 : If $r \in (PR(C_i) - PR(C_{i'}))$ $\forall i, i' = 1,2$ and $i \neq i'$ then

$$Transform(C_{i'}, S_{i'}, Receive(Choicem(r', r, S_{i'}))) \qquad \forall r' \in SR(C_{i'});$$

La règle 8 (Listing 4.11) est exécutée si et seulement si le helper *ChoiceN_C12()* ou *ChoiceN_C21()* est vrai. Ces helpers déterminent si le rôle ne participe pas dans les collaborations C_1 ou C_2.

```
Rule CollaborationS2StateNPR {

from C: MMs!Sub_Collaboration (C.ChoiceN_C12() or C.ChoiceN_C21() )

to   S: MMt!CompositeState (Name<- C.Name, outgoing<-C.outgoing,
     incoming<-C.incoming, states<-Si, transition<-ti, states <-sc,
     transition<-t, states<-st, transition <-tf, states <-Sf ),
```

```
    Si: MMt!InitialState(outgoing <- ti),
    ti: MMt!Transition ( source <-Si, target <- sc),
    sc: MMt!State(incoming<-ti,outgoing <-tf),
    tf: MMt!Transition ( source <-sc, target <- Sf),
    Sf:MMt!FinalState (incoming <-tf)

do {thisModule.i<-thisModule.i+1;
 sc.Name<-'S'+thisModule.i.toString(); thisModule.NameCol<-C.Name;
 sc.action <-C.GetSRCol->collect(x|thisModule.ReceiveChoicem(x));
 thisModule.j<-thisModule.j+1; ti.Name<-'T'+thisModule.j.toString();
thisModule.j<-thisModule.j+1; tf.Name<-'T'+thisModule.j.toString();}}
```

Listing 4.11 : Code ATL de la règle 8 au niveau d'une sous-collaboration.

Cette règle consiste à :

- Générer un état composite S dans le modèle RolderivBeh(r) conforme au méta-modèle cible MM_t dont S.Name= C_i.Name ;
- Générer (un état initial) dans S ;
- Générer les actions de réception des messages de coordination *Choicem* provenant des rôles initiateurs de la collaboration calculés par le helper *GetSRCol()* ;

```
helper context MMs!Action def: GetSRCol():OrderedSet (MMs!Role)
= if self.oclIsTypeOf(MMs!Sub_Collaboration) then
  self. GetSRColX() else self.GetSRColY()   endif;
```

- Générer les transitions pour relier les différents états dans l'état composite S ;
- Générer (un état final) dans S ;
- Connecter S à ses états prédécesseurs et successeurs.

Règle 9, 10 et 11 :

If $r \in PR(C_i)$ then $Transform(Controlflow, Transition)$ $\forall i = 1,2;$

If $r \in PR(C_i)$ then $Transform(DecisionNode, ChoiceState)$ $\forall i = 1,2;$

If $r \in PR(C_i)$ then $Transform(MergeNode, JunctionState)$ $\forall i = 1,2;$

Les règles 9,10 et 11 (Listing 4.12) ne sont exécutées que si leurs conditions sont évaluées à vrai à savoir les helpers *IsTransformC()*,*IsTransform1()* et *IsTransform2()* respectivement.

```
rule Controlflow2Transition {
from s: MMs!Controlflow (s.IsTransformC())
to r: MMt!Transition( source <- s.source, target <- s.target)
do {thisModule.j<-thisModule.j +1;
          r.Name<-'T'+ thisModule.j.toString()}}


rule DNode2State {
from s: MMs!DecisionNode (s.IsTransform1())
to r: MMt!ChoiceState(Name <-'Decision State',outgoing<-s.outgoing)}


rule MergeNode2State {
from s: MMs!DecisionNode (s.IsTransform2())
to r: MMt!ChoiceState(Name <-'Junction State',outgoing<-s.outgoing)}
```

Listing 4.12 : Code ATL des règles 9, 10 et 11.

Les autres cas de chorégraphie présentés dans le méta-modèle des exigences à savoir la boucle tant que Strong et Weak ainsi que la structure parallèle sont mis en œuvre par des règles déclaratives telles que les précédentes.

5 Conclusion

Dans ce chapitre, nous avons présenté la mise en œuvre de l'approche de dérivation du comportement présentée dans le chapitre 3 de cette thèse. Cette approche consistait à réaliser la transformation du modèle des exigences en modèles conceptuels avec dérivation du comportement des différents rôles composant le système. Au début de ce chapitre, nous avons décrit les langages de transformation de modèles, leurs principes ainsi que leurs caractéristiques et en particulier celles

exhibées dans les langages QVT (Query/ View/ Transformation) et ATL (atlas Transformation Language).

La mise en œuvre de l'approche de dérivation proposée a été réalisée sur la plateforme Eclipse avec le langage de transformation de modèles ATL (Atlas Transformation Language). La mise en œuvre de la transformation du modèle des exigences vers le modèle conceptuel exige :

(i) la représentation du méta-modèle source, du méta-modèle cible et du modèle source ;

(ii) l'application des règles de transformation de modèles spécifiées dans le langage ATL sur le modèle source. Cette phase génère les modèles résultats.

Nous avons présenté le code ATL de quelques règles de transformation nécessaires pour les cas de chorégraphie séquence Strong, séquence Weak et structure de choix exprimés dans le méta-modèle des exigences. Ces règles permettent la dérivation du comportement des différents rôles impliqués dans le comportement global du système dans le contexte de l'approche de dérivation décrite dans le chapitre 3. Une mise en œuvre similaire est réalisée pour les autres cas de chorégraphie.

Les comportements dérivés des composants du système s'exécutent dans un environnement distribué. Un important aspect dans le développement des systèmes distribués est d'assurer la conformité de la conception du système avec sa spécification, autrement dit dans notre approche de dérivation, les comportements dérivés doivent être validés et vérifiés avant leur déploiement. Du fait, nous proposons dans le chapitre suivant une approche de vérification dirigée par les modèles pour valider les comportements automatiquement dérivés d'un système distribué. L'objectif de cette approche est d'augmenter les performances et la qualité de service du système développé.

Chapitre 5

Une vérification formelle des comportements dérivés d'un système distribué

1 Introduction

La dérivation automatique du comportement d'un système distribué à partir de sa spécification des exigences globales est une étape importante dans le développement du systèmes logiciels complexes. Elle permet aux développeurs d'obtenir une conception du système à partir de sa spécification des exigences globales. Pour assurer la conformité de cette conception avec la spécification globale du système, les comportements dérivés doivent être validés et vérifiés avant leur déploiement. En fait, les méthodes formelles sont des outils puissants permettant de vérifier l'exactitude logique des systèmes concurrents à différents niveaux de leur cycle de vie. La vérification de modèles est l'une de ces méthodes et est particulièrement bien adaptée à la vérification automatisée de systèmes à états finis. Dans ce chapitre, nous présentons une approche de vérification de modèles basée sur une transformation de modèles pour valider le comportement dérivé d'un système distribué. Cette approche dirigée par les modèles vérifiera si le système dérivé se comporte correctement par rapport à sa spécification globale, le but est d'augmenter les performances et la QoS du sys-

tème. Cette approche permet au développeur de raisonner sur un modèle du système global plutôt que sur le système lui-même.

Les concepteurs de logiciels savent par expérience que, très probablement, tout système logiciel ne fonctionne pas parfaitement après la première compilation réussie, ni la deuxième ni la troisième. Parfois, il faut un certain temps pour découvrir comment un logiciel apparemment correct peut échouer de façon subtile. Les petits défauts peuvent se cacher pendant des années et apparaître au moment le plus gênant lorsqu'ils sont le moins attendus. Cependant, il existe des cas dans lesquels ces défauts ne sont pas acceptables, tels que dans les applications de surveillance médicale ou les applications économiques critiques ; les défauts logiciels peuvent entraîner des pertes de vie ou causer des dommages économiques importants. Étant donné qu'une grande partie de notre monde est actuellement contrôlée par des applications logicielles, le défi le plus important consiste à trouver les moyens et les outils afin de rendre ces applications plus fiables. La vérification de tels systèmes implique une validation de leur conception. Cette vérification vérifie si la conception du système satisfait à ses exigences. (Si ce n'est pas le cas, alors il est souhaitable de le savoir au début du processus de conception !) Ces tâches, la vérification du système et la validation de la conception, peuvent être accomplies en utilisant les techniques de simulation et de test.

Les techniques éprouvées de simulation et de test sont des outils de débogage très utiles pour les premières étapes de la conception et de la vérification d'un système. Ils impliquent de vérifier le comportement du système sur un grand nombre d'entrées prévues. Cependant, tout au long de l'affinement d'un système, les bugs restants deviennent moins nombreux et plus difficiles à découvrir. Une importante lacune dans le processus d'utilisation des techniques de la simulation et/ou des tests pour la vérification et la validation est qu'il n'y a aucun moyen de savoir quand tous les bugs du système ont été trouvés. En d'autres termes, les tests et la simulation peuvent être utilisés pour démontrer la présence de bugs mais pas leur absence. Tout simplement, il a été prouvé que les tests et la simulation ne peuvent pas être utilisés pour garantir un très haut niveau de fiabilité dans un délai réaliste [21]. Pour certains systèmes, cette limitation est un risque acceptable. Pour ces systèmes, il suffit de réduire le niveau de bugs au-dessous d'une certaine tolérance mesurable. Pour d'autres systèmes, tels que les systèmes financiers ou les systèmes de surveillance médicale, dont la fiabilité est essentielle car la défaillance est potentiellement catastrophique et non tolérée. Il est absolument nécessaire dans ce cas que le système respecte ses spécifications en examinant

tous les comportements possibles, y compris ceux qui sont inattendus. Cette assurance peut être accomplie de manière fiable en utilisant des méthodes formelles.

Les méthodes formelles sont des outils puissants utilisées par les ingénieurs logiciels pour vérifier l'exactitude logique des systèmes concurrents. La vérification formelle a été utilisée avec succès pour vérifier divers systèmes tels que le contrôle du trafic aérien, le pilotage automatique, la conception de processeurs, les systèmes de survie et de nombreux autres systèmes de sécurité [96]. Bien qu'il existe un éventail de techniques différentes pour la vérification formelle, la vérification de modèles est l'une des méthodes formelles les plus puissantes et est particulièrement bien adaptée à la vérification automatisée des systèmes à états finis. La vérification de modèles nécessite des algorithmes sophistiqués basés sur la théorie des automates et la logique pour vérifier les modèles, mais pas les programmes [41]. Les modèles sont des descriptions de haut niveau d'un système. Ces modèles représentent fidèlement le système, tout en restant suffisamment concis pour permettre de vérifier son exactitude. La vérification de modèles est la méthode de vérification préférable pour deux raisons. Premièrement, il est beaucoup plus efficace et rentable d'effectuer la vérification le plus tôt possible dans le processus de conception du système, évitant ainsi la possibilité de découvrir une erreur qui nécessite une refonte du système achevé. Deuxièmement, il est plus simple et plus facile de raisonner sur un modèle du système plutôt que sur le système lui-même parce que le modèle ne comprend que les caractéristiques pertinentes du système et parce que le modèle est plus facile à construire et à reconcevoir [96].

L'approche de dérivation de comportement présentée dans le chapitre 3 se focalise sur les premières phases de développement d'un système distribué, en particulier l'obtention d'une conception du système à partir de sa spécification des exigences globales. Elle est basée sur l'approche IDM pour dériver le comportement des composants du système en transformant le modèle des exigences globales en modèle de conception. Le modèle des exigences globales du système décrit le comportement fonctionnel d'un système donné de manière abstraite. Le modèle de conception représente le comportement local de chaque composant du système. Les comportements dérivés pour les composants du système s'exécutent dans un environnement distribué. Les aspects de concurrence rendent ces applications difficiles à développer et à vérifier. Le défi le plus important consiste à trouver les moyens et les outils nécessaires afin de rendre ces comportements dérivés plus fiables. Cet objectif consiste à vérifier si le système dérivé en question se comporte tel qu'il a été conçu

pour se comporter. Toutes ces raisons ont motivé notre travail pour l'utilisation d'une approche dirigée par les modèles pour faciliter la vérification du comportement des composants du système.

Ce chapitre présente une approche dirigée par les modèles (IDM) pour vérifier et valider le processus de dérivation. Il s'agit de vérifier si la collaboration des comportements dérivés satisfait la spécification globale du système initial. Si ce n'est pas le cas, il est souhaitable de le savoir dès le début du processus de conception. Cette approche IDM décrit un processus de vérification basé sur les modèles pour vérifier et analyser la cohérence logique du comportement du système dérivé. Le défi pour les concepteurs est de développer des modèles qui représentent fidèlement le système, alors que dans notre cas, ces modèles sont automatiquement dérivés du comportement global du système. Il n'y a pas de temps à consacrer à la conception de ces modèles pour s'assurer qu'ils décrivent suffisamment le système. Le processus de vérification dirigé par les modèles est principalement basé sur une transformation automatique des modèles dérivés en modèles spécifiques requis par un vérificateur de modèles afin d'effectuer la vérification.

2 L'approche de vérification dirigée par les modèles

Le développement d'applications distribuées est une tâche de plus en plus complexe. La construction et la maintenance de tels systèmes nécessitent l'existence de méthodologies de développement explicitement énoncées. De telles méthodologies devraient fournir des techniques et des approches qui traitent de la dérivation automatique du comportement des composants du système. En outre, le défi le plus important consiste à trouver les moyens nécessaires qui permettent de garantir que ce comportement dérivé fonctionne sans erreurs. Toutes ces raisons ont motivé notre proposition d'une approche dirigée par les modèles (IDM) qui permet d'abord de dériver le comportement de chaque composant du système et ensuite vérifier cette dérivation. Le processus de vérification effectue une vérification formelle des comportements dérivés des différents composants du système. La figure (Fig. 5.1) montre une vue d'ensemble de l'approche de dérivation et de vérification dirigée par les modèles.

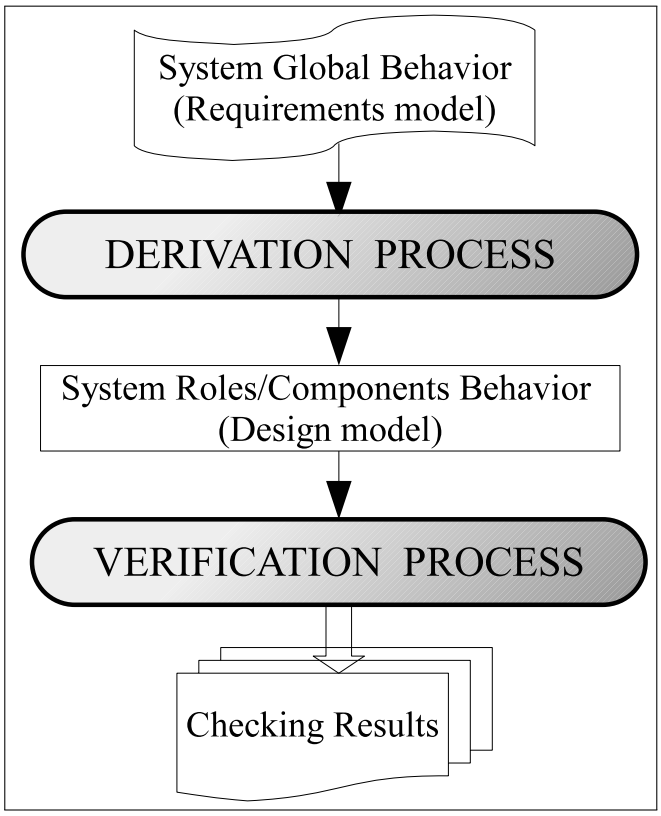

Fig. 5.1 : L'approche de vérification dirigée par les modèles.

Le comportement de chaque composant du système est obtenu en transformant le modèle global des exigences du système. Le processus de vérification garantit que la collaboration des comportements dérivés doit satisfaire le modèle des exigences initial. La vérification de modèles est l'une des méthodes formelles les plus puissantes utilisées pour la vérification des systèmes concurrents et distribués.

La première étape de la vérification de modèles consiste à décrire un modèle du système à vérifier. Le défi consiste à concevoir un modèle qui ne décrit que les caractéristiques pertinentes du système entier, et qu'il soit suffisamment détaillé pour représenter fidèlement le système. Les travaux présentés dans [25, 26, 88, 96, 119] sont axés sur la vérification et les techniques utilisant les automates pour réaliser la vérification. Ils ne se sont pas intéressés par la création du modèle à vérifier. Dans l'approche de dérivation du comportement proposée dans le chapitre 3, le concepteur de logiciel n'a aucun effort à consacrer à la conception du système. Le modèle de conception est

automatiquement obtenu à partir de la spécification des exigences. La spécification des exigences décrit le comportement global du système. Pour représenter le comportement global du système, nous utilisons les collaborations UML comme principaux blocs d'activités pour la construction du modèle des exigences. Une collaboration donnée décrit la structure d'éléments collaboratifs (rôles), chacun remplissant une fonction spéciale, qui accomplissent collectivement la fonctionnalité souhaitée. Les collaborations sont très appropriées pour modéliser les exigences car elles fournissent un cadre structurel pour de telles exigences, qui incarnent à la fois le comportement de chaque rôle et les interactions entre les rôles. Cela permet de décrire les exigences globales d'un système donné d'une manière abstraite.

Cependant, une fois le modèle est créé, ce modèle peut provoquer la création d'un nombre astronomique d'états lors de la vérification du modèle (le problème d'explosion d'états), qui ne peut pas être pris en charge par la mémoire de l'ordinateur. Les traitements réels ne peuvent pas avoir un nombre infini d'états distincts, pour la simple raison que la mémoire de l'ordinateur est finie. Le problème d'explosion d'états provoque l'arrêt de la vérification du modèle. Ce problème peut être atténué pendant la phase de modélisation en réduisant l'espace d'états aussi que possible. La réduction de l'espace d'états est obtenue en modélisant uniquement les aspects pertinents du système [26].

La réduction de l'espace d'états pendant l'étape de modélisation peut parfois être insuffisante et peut provoquer le problème d'explosion d'états. Pour résoudre ce problème, il est préférable d'utiliser une autre technique pour réduire l'espace d'états. Cette technique consiste à représenter indépendamment les composants du système fonctionnant simultanément pour tirer le meilleur avantage possible de la réduction de la taille du modèle du système. Cette technique de réduction consiste à décomposer le modèle global du système en plusieurs modèles concurrents ayant exactement le même comportement que le système en question. Par conséquent, il est préférable de modéliser l'ensemble du système avec un ensemble de modèles des différents sous-systèmes indépendants et communicants qu'un seul modèle. Dans notre approche, nous tirons parti du processus de dérivation qui transforme le modèle du comportement global du système en un ensemble de comportements des composant du système. Les messages de synchronisation sont inclus dans les modèles dérivés pour assurer la synchronisation et la coordination des différents composants du système. L'ensemble du système est ainsi modélisé par un ensemble de modèles communicants.

Cette décomposition automatique réduit la complexité du système pour la vérification. Seule une telle séquence de modèles doit être vérifiée. La dernière étape de l'approche consiste à procéder à la génération du code pour une plateforme spécifique une fois que les résultats de la vérification montrent que le système est fiable.

3 Processus de vérification

Les comportements dérivés du système fonctionnent dans un environnement distribué et collaborent pour atteindre le comportement global du système. Cette coopération nécessite une méthode formelle pour s'assurer que la collaboration de ces comportements doit satisfaire le modèle initial des exigences. La vérification de modèles est le processus formel par lequel une propriété comportementale désirée (une spécification) est vérifiée pour un système donné (le modèle). Cette vérification devrait examiner tous les comportements possibles qui provoquent les différentes transitions du système entre les différents états accessibles. Une fois que le modèle et la spécification du système ont été déterminés, l'étape de vérification du modèle peut être accomplie. La vérification du modèle renvoie une erreur lorsque la spécification n'est pas vérifiée pour toutes les exécutions du système. Cette erreur indique qu'un comportement erroné a été détecté. Un contre-exemple est produit, il consiste en une trace du modèle partant de l'état de départ vers l'état d'erreur dans lequel la spécification a été violée. Cette trace fournit les informations nécessaires pour réaliser un diagnostic.

Dans l'approche de dérivation proposée [37], ces modèles sont automatiquement dérivés du comportement global du système. Il n'y a aucun temps à consacrer à la conception de ces modèles et à s'assurer qu'ils décrivent suffisamment le système. Cette décomposition automatique réduit la complexité du système pour la vérification. Elle génère un ensemble de modèles décrivant plusieurs comportements concurrents dans le système qui ont exactement le même comportement que celui du système en entier, donc seule une telle séquence de modèles doit être vérifiée. Ainsi, il ne reste plus qu'à transcrire ces modèles en une forme de modèles à vérifier, qui peut facilement être convertie en langage Promela, et par la suite la vérification des modèles peut être effectuée. SPIN [41] est un outil logiciel pour la vérification de systèmes physiques. Il permet de vérifier

les modèles décrits dans le langage Promela. Comme il permet de réaliser une vérification des propriétés comportementales.

Pour atteindre cet objectif de vérification formelle, nous suggérons l'utilisation d'une approche dirigée par les modèles pour réaliser le processus de vérification d'un système distribué donné. Cette approche permettra aux développeurs d'obtenir automatiquement les modèles qui seront vérifiés. Elle transforme les comportements des composants du système (modèles de conception) en modèles spécifiques à la vérification. Un ensemble de règles est requis pour réaliser ces transformations de modèles. Enfin, un programme Promela est généré à partir de ces modèles spécifiques grâce à une transformation de type modèle-texte. Le résultat est combiné avec les propriétés comportementales pour réaliser l'analyse et la vérification du système. Ainsi, l'application de ce processus de vérification dirigé par les modèles nécessite la définition des méta-modèles appropriés à chaque niveau d'abstraction avec les transformations de modèles correspondantes. Ce processus est composé des étapes énumérées dans la figure (Fig. 5.2) :

1. Définition du méta-modèle spécifique à la vérification.

2. Établissement des règles de transformations de modèles qui régissent la transformation des concepts du méta-modèle de conception vers ceux du méta-modèle spécifique à la vérification.

3. Définition des transformations de modèles en texte qui transforment les modèles spécifiques à la vérification en un programme Promela.

4. Spécification des propriétés comportementales (propriétés d'exactitude).

5. Vérification du comportement du système, spécifiée dans Promela, avec le vérificateur de modèles SPIN.

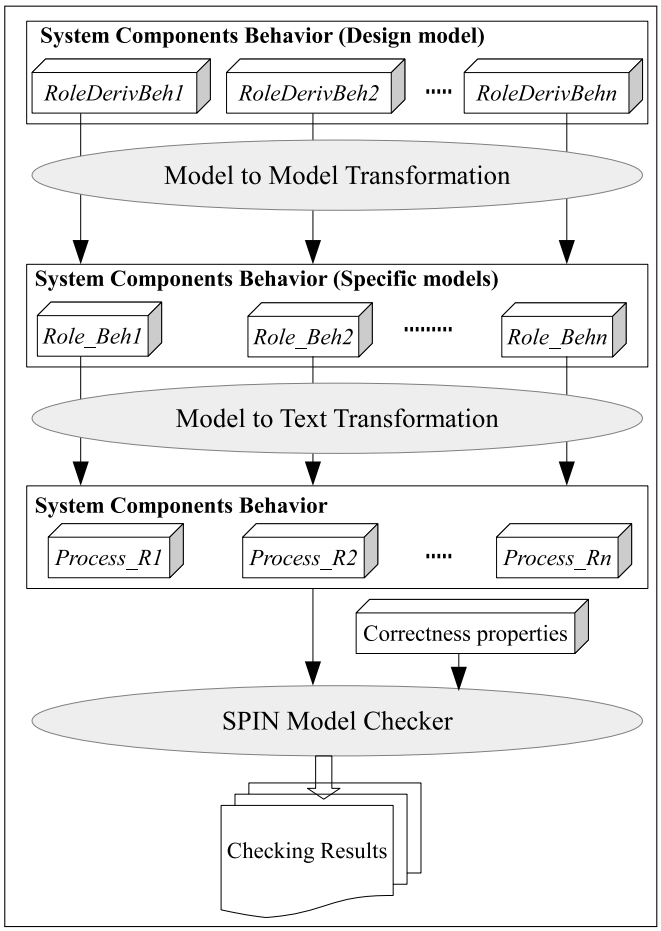

Fig. 5.2 : Le processus de vérification.

3.1 Le méta-modèle spécifique à la vérification

Tous les systèmes peuvent être modélisés en utilisant de nombreuses stratégies de modélisation pour optimiser la clarté et fournir une vue abstraite du système. Le modèle est ensuite utilisé pour réaliser une vérification formelle via le vérificateur de modèles. Indépendamment de la forme originale du modèle du système, nous le traduisons éventuellement en une sorte de machine à états finis communicante (FSM), qui est simplement un type de graphe appelé automate. Comme son nom l'indique, cette machine d'états finis est un modèle de comportement utilisant des états du système et des transitions entre ces états. Une transition est un changement d'états qui est déclenché par un événement, c'est-à-dire que les transitions mettent en correspondance des paires événement-état

avec d'autres états. Comme indiqué dans la définition, l'ensemble des états devrait être fini. De plus, il est supposé qu'il existe un ensemble fini d'événements distincts et un ensemble de transitions également fini.

Pour décrire les comportements dérivés des composants du système par un ensemble de machines à états finis communicantes, nous définissons un état pour chaque collaboration à laquelle le composant y participe. Par conséquent, chaque état du graphique est étiqueté par le nom de la collaboration. Nous considérons que la transition du comportement d'un composant d'un état à un autre est possible lorsque les événements associés à cette transition peuvent être déclenchés. Ainsi, le comportement de chaque composant est modélisé par une machine à états finis (FSM). Ce FSM comprend un ensemble d'états (Q), un ensemble d'événements (X), un ensemble d'états finaux (F) et une fonction de transition d'états. Chaque état possède des règles qui décrivent l'action de la machine à entreprendre pour chaque événement d'entrée ou de sortie, qui est représenté dans la fonction de transition d'états. La fonction de transition prend l'état actuel et un ou plusieurs événements et renvoie l'état suivant. Cette machine à états finis est formellement définie comme un quintuple (X, Q, q_0, δ, F) tel que :

- X = un ensemble fini d'événements non vide (événements d'entrée et de sortie : réception et émission des messages)
- Q = un ensemble fini d'états non vide
- q_0 = état initial, q_0 est un élément de Q
- δ = la fonction de transition d'états, définie par δ : Q x X $\rightarrow$ Q
- F = un ensemble d'états finaux où F est un sous ensemble de Q

D'après l'interprétation mathématique ci-dessus, on peut dire qu'une machine à états finis contient un nombre fini d'états. Chaque état accepte un nombre fini d'événements d'entrée et de sortie, et chaque état a des règles qui décrivent l'action du FSM pour chaque événement d'entrée ou de sortie, qui est représenté dans la fonction de transition d'états. En même temps, un événement d'entrée ou de sortie peut amener la machine à changer d'états.

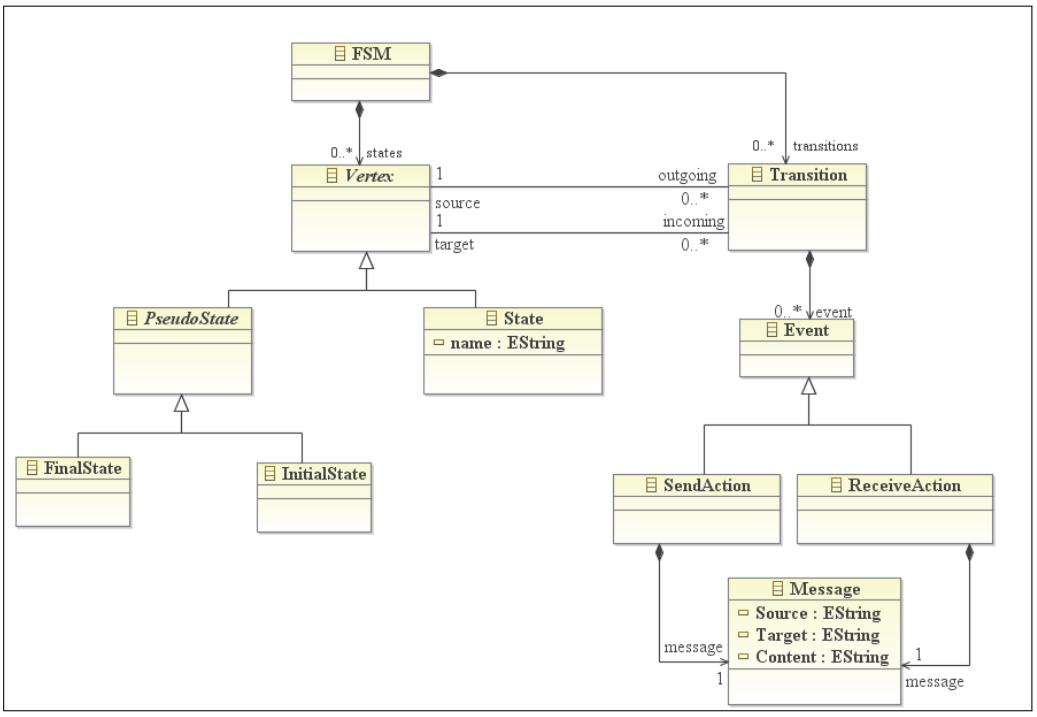

Fig. 5.3 : Le méta-modèle machine à états finis communicante (FSM).

La définition du méta-modèle spécifique à la vérification vise à décrire le comportement du système par des modèles acceptés par l'outil de vérification de modèles. Le méta-modèle machine à états finis communicante (Fig. 5.3) a été conçu pour fournir les concepts appropriés et leurs relations, qui décrivent le comportement d'un composant ou d'un rôle dans le système. Un tel méta-modèle est considéré comme le méta-modèle cible de la transformation modèle à modèle au cours du processus de vérification. Il définit le méta-modèle FSM avec ses classes et ses associations principales. Un FSM se compose de plusieurs *States* et *Transitions*. La classe *State* modélise une situation ou une phase significative du cycle de vie du composant du système. La classe *Transition* permet de spécifier le flux de contrôle (connexions) entre les classes *State*. Elle modélise une relation entre un état source et un état cible, qui représente comment le composant système peut répondre à l'occurrence d'un événement lorsque le composant est dans l'état source. Le comportement d'un composant du système est défini comme un automate dont le langage consiste en l'ensemble de tous les comportements possibles du composant du système. Le comportement d'un composant est essentiellement une séquence finie d'états du composant en question.

3.2 Transformation Modèle-Modèle

La transformation modèle-modèle (M2M) transforme automatiquement les modèles de conception issus du processus de dérivation, qui décrivent le comportement de chaque composant du système, en modèles machines à états finis. Elle est basée sur un ensemble de règles de transformation. Une règle consiste à transformer un concept décrit dans le méta-modèle de conception en un concept correspondant dans le méta-modèle spécifique à la vérification. Pour ce faire, nous avons défini une fonction appelée *Translate (Design_Concept, FSM_Concept, Event)*. Le paramètre *Event* de cette fonction représente les événements ou les actions associées aux transitions qui provoquent le changement d'états du système. Cette transformation M2M est conçue pour générer les différents concepts d'une machine à états finis pour chaque composant du système. Ces modèles sont utilisés dans la transformation de modèle à texte pour la génération automatique de code. Certaines de ces règles sont décrites ci-dessous :

Règle 1 : Chaque état composite contenant des actions (envoi et/ou réception de message) est transformé en un état dans la machine à états finis communicante. Cet état est étiqueté par le nom de l'état composite.

Règle 2 : Les actions initiales (envoi et/ou réception de message) dans un état composite sont transformées en événements associés aux transitions entrantes vers l'état correspondant.

Règle 3 : Les actions terminales (envoi et/ou réception de message) dans un état composite sont transformées en événements associés aux transitions sortantes vers l'état correspondant.

Règle 4 : Les états composites correspondant aux collaborations auxquelles le composant ne participe pas ne sont pas transformés. Alors que les messages de synchronisation inclus dans ces états composites sont associés aux transitions entrantes des états successeurs.

Les méta-modèles et les transformations de modèles décrits dans ce chapitre ont été développés en utilisant les fonctionnalités fournies par la plateforme Eclipse. La plateforme Eclipse offre une implémentation de la norme MOF (Meta Object Facility) OMG [79], appelée Eclipse Modeling Framework (EMF) [20]. Ces méta-modèles sont créés en utilisant EMOF (Essential MOF) qui permet aux concepteurs de créer, manipuler et stocker à la fois des modèles et des méta-modèles.

La conformité du modèle source avec le méta-modèle de conception est obtenue par le processus de dérivation. La conformité des modèles FSM avec le méta-modèle cible est garantie par la transformation modèle-modèle. Les règles de transformation sont exprimées dans le langage ATL (ATLAS Transformation Language) [9, 46, 49] qui fournit une solution pour les transformations de modèle à modèle.

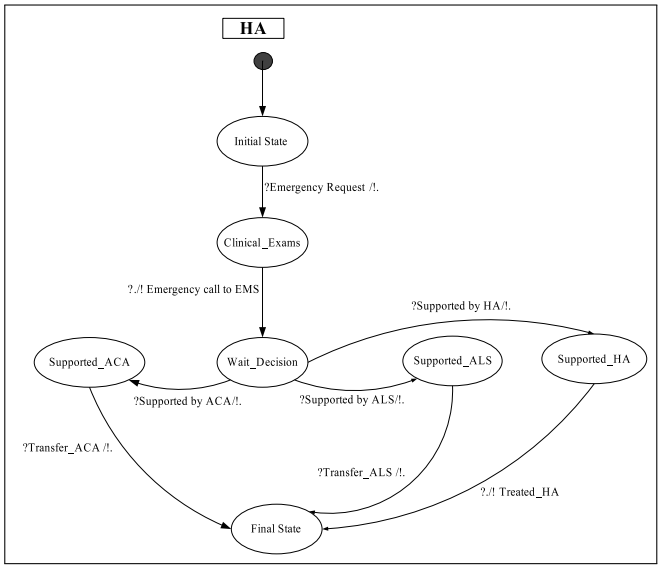

Fig. 5.4 : Le comportement du neurologue HA décrit par un FSM.

Par exemple, la figure (Fig. 5.4) décrit le comportement de l'urgentiste de l'hôpital (HA). Cette automate FSM est obtenu de la transformation du modèle de conception de la figure (Fig.3.14). Chaque état est étiqueté avec le nom de collaboration à laquelle le rôle participe. Les transitions sont étiquetées par les événements qui provoquent le changement d'états du système. Le comportement de l'urgentiste HA débute à l'état *Initial*. Lorsque le patient émet une requête *Emergency request*, le HA passe à l'état *Clinical_Exams*. A ce moment, le HA décide si cette requête nécessite plus d'investigations et un diagnostic plus poussé, il envoie un appel d'urgence au SAMU (EMS) et passe à l'état *Wait_Decision*. Lorsqu'une décision a été prise en réponse à la demande de HA, la décision *supported by HA* provoque la transition de HA vers l'état *Supported_HA*, tandis que les décisions *Supported by ALS* and *Supported by ACA* le déplacent respectivement vers les états *Supported_ALS* et *Supported_ACA*. L'exécution de la tâche par HA le fait transiter vers l'état *Final*. La réception des messages *Transfer_ALS* et *Transfer_ACA* amène HA aussi à l'état *Final*.

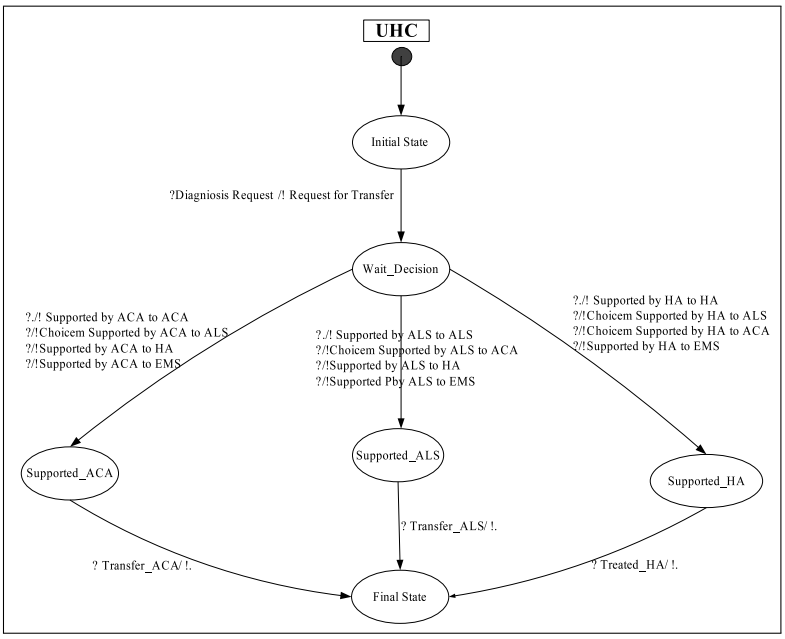

Fig. 5.5 : Le comportement du neurologue CHU décrit par un FSM.

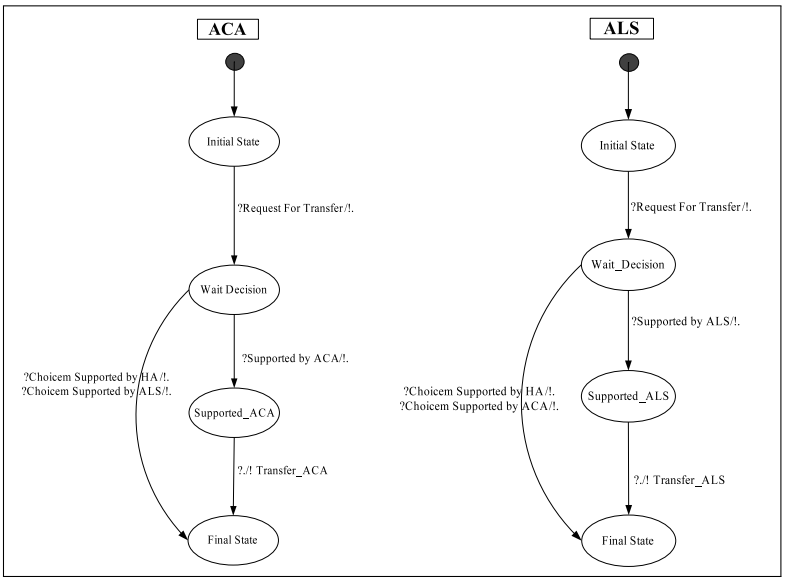

Fig. 5.6 : Le comportement du SMUR et du VLS décrit par des FSM.

Les figures suivantes (Fig. 5.5, Fig. 5.6 et Fig. 5.7) illustrent les machines à états finis communicantes qui sont générées pour les composants CHU (UHC), SMUR (ALS), VLS (ACA) et SAMU (EMS) dans l'application de télédiagnostic.

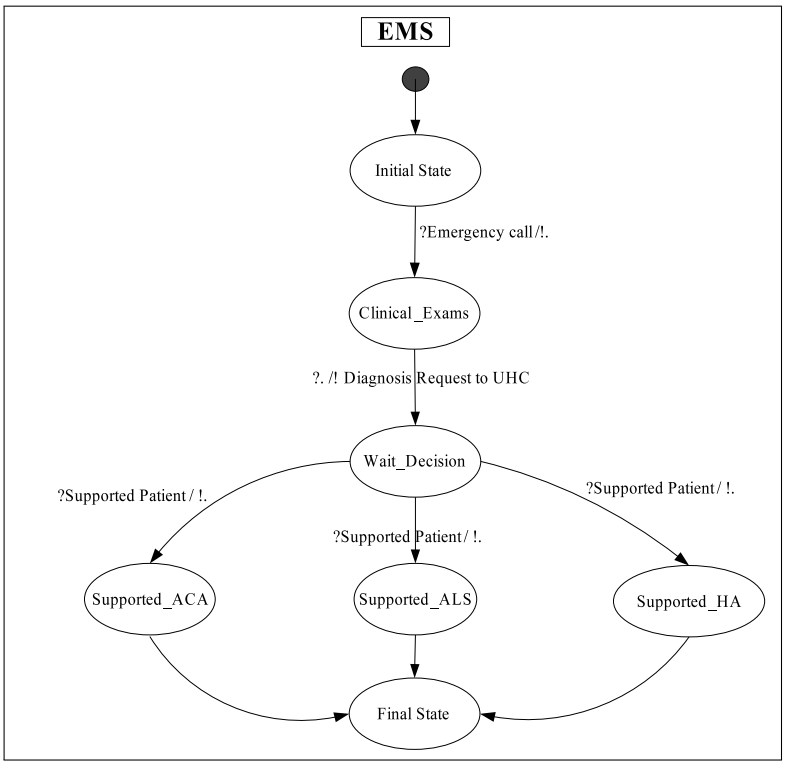

Fig. 5.7 : Le comportement du SAMU décrit par un FSM.

3.3 Transformation modèle-texte

La transformation modèle-texte (M2T) permet à générer automatiquement un programme Promela. Cette activité prend en entrée les modèles machines à états finis communicantes, décrivant les comportements du système, et génère en sortie le code source de l'application à valider par le vérificateur de modèles SPIN. SPIN est un outil logiciel pour vérifier les modèles de systèmes physiques décrits dans le langage Promela. Promela (Process Meta Language) est un langage de modélisation et de validation. Ce langage permet la création dynamique de processus concurrents. La communication entre les processus se fait via des canaux de communication par messages et peut être synchrone ou asynchrone. Promela est un langage qui fait abstraction des détails des systèmes distribués ne faisant pas partie du processus d'interaction. L'outil SPIN est utilisé pour valider des fractions de comportements du système ; considérées comme suspects.

Le programme Promela généré se compose de *processus*, de *canaux* de messages, de *variables* et d'une section principale *main* qui définit comment les actions des processus peuvent être entrelacées dans le temps. Les processus sont des objets globaux. Ils spécifient les comportements des composants du système. Les canaux de messages et les variables peuvent être déclarés globalement ou localement dans un processus. Les canaux et les variables globales définissent l'environnement dans lequel les processus s'exécutent. Ainsi, nous obtenons, après la transformation M2T, un processus qui spécifie le comportement de chaque composant du système. Les transformations M2T ont été développées en utilisant le plugin Eclipse Acceleo. Acceleo est une implémentation du langage de transformation MOF Model to Text [33]. Le listing 5.1 décrit les variables globales et les canaux de messages nécessaires, et le listing 5.2 décrit le code du processus correspondant au médecin urgentiste de l'hôpital (HA) et celui de la section principale (processus Init).

```
#define Emergency_Req_HA                1
#define Emergency_Call_EMS              2
#define Diagnosis_Req                   3
#define Transfer_ALS                    9
#define Transfer_ACA                    10
#define Treated_HA                      11
#define Wait_Request                    12
#define Clinical_Exams                  13
#define Wait_Decision_Making            14
#define Supported_HA                    15
#define Supported_ACA                   16
#define Supported_ALS                   17
#define Supported_By_HA                 18
#define Supported_By_ACA                19
#define Supported_By_ALS                20

chan HA_EMS = [1] of { mtype, byte };
chan HA_UHC = [1] of { mtype, byte };
chan HA_ACA = [1] of { mtype, byte };
chan HA_ALS = [1] of { mtype, byte };
chan UHC_ACA = [1] of { mtype, byte };
chan UHC_ALS = [1] of { mtype, byte };
chan EMS_UHC = [1] of { mtype, byte };

mtype {HA, UHC, EMS, ACA, ALS}
byte HA_State, UHC_State, EMS_State,
ACA_State, ALS_State, ACA_Alert, ALS_Alert;
```

Listing 5.1 : Les variables globales et les canaux de messages.

```
proctype HA_Act(){
byte RCVD_Msg;
HA_State= Initial_state;

do
 ::(HA_State==Initial_state)
                ->HA_Init?HA(RCVD_Msg)
                ->(RCVD_Msg==Emergency_Req_HA)
                ->HA_State=Clinical_Exams
 ::(HA_State==Clinical_Exams)
   ->if ::atomic{HA_EMS!EMS(Emergency_Call_EMS)
         ->HA_State=Wait_Decision_Making}
     fi
 ::(HA_State==Wait_Decision)
   ->HA_UHC?HA(RCVD_Msg)
   -> if
      ::(RCVD_Msg==Supported_By_HA)
           -> HA_State=Supported_HA
      ::(RCVD_Msg==Supported_By_ACA)
           -> HA_State=Supported_ACA
      ::(RCVD_Msg==Supported_By_ALS)
           -> HA_State=Supported_ALS
     fi
 ::(HA_State==Supported_HA)
    ->atomic{ HA_UHC!UHC(Treated_HA) ->HA_State=Final_State}
 ::(HA_State==Supported_ACA)
    ->HA_ACA?HA(RCVD_Msg)
    ->(RCVD_Msg==Transfer_ACA)
                ->HA_State=Final_State
 ::(HA_State==Supported_ALS)
   ->HA_ALS?HA(RCVD_Msg)
   ->(RCVD_Msg==Transfer_ALS)
         ->HA_State=Final_State
 ::(HA_State==Final_State) ->break
od}

init {
        run HA_Act();    run EMS_Act();
        run UHC_Act();   run ACA_Act();
        run ALS_Act();
        (HA_State==Wait_Request)
        ->HA_Init!HA(Emergency_Req_HA)
}
```

Listing 5.2 : Le processus HA et le processus Init.

Le vérificateur de modèles SPIN exécute le programme étape par étape. Dans chaque étape, une instruction de base exécutable est sélectionnée arbitrairement. L'outil SPIN examine sa clause d'exécutabilité pour déterminer si cette instruction est exécutable ou non. Lorsque cette condition est évaluée à vraie, la clause d'effet de l'instruction est ensuite appliquée et l'état actuel du processus est mis à jour. SPIN continue d'exécuter les instructions jusqu'à ce qu'il ne reste plus d'instructions exécutables, ce qui se produit lorsque (1) le nombre de processus actifs devient zéro et tous les processus se terminent sans aucune erreur ou (2) si un état invalide est atteint.

3.4 Propriétés comportementales (propriétés de correction)

Les systèmes distribués sont systématiquement développés à partir de leurs spécifications. Cependant, une fois que ces systèmes sont développés, il n'y a aucun moyen de vérifier si le système développé suit sa spécification. Une méthode de vérification exhaustive n'est pas pratique pour vérifier les systèmes distribués. Le problème avec une telle méthode est qu'elle génère un nombre astronomique de calculs possibles. La méthode la plus appropriée pour vérifier qu'un système suit ses spécifications doit être basée sur une notion formelle, très précise et claire pour exprimer l'énoncé des spécifications. Par conséquent, la logique est un bon moyen d'exprimer les propriétés de correction d'un système. La logique a la capacité d'exprimer les spécifications du système d'une manière concise et claire et permet ainsi l'automatisation du processus de vérification. Les systèmes concurrents impliquent nécessairement la notion de temps. Par conséquent, la logique propositionnelle n'est pas suffisamment expressive pour décrire de tels systèmes réels.

La Logique Temporelle Linéaire (LTL) a été introduite comme un outil pour raisonner sur les programmes concurrents [87]. Les logiques temporelles sont des logiques modales orientées vers la description de l'ordre temporel des événements. La notion de temps n'est pas explicitement introduite dans la description de l'ordre des événements. Les logiques temporelles sont particulièrement efficaces pour décrire les systèmes concurrents [27]. Les vérificateurs de modèles LTL suivent l'approche de la théorie des automates [118]. Dans de tels vérificateurs de modèles, le modèle du système et la négation de sa propriété de correction (exprimés en logique temporelle) sont transformés en automates d'états finis et sont exécutés simultanément. Le vérificateur de modèles recherche un chemin accepté par les deux automates. S'il en trouve un, alors ce chemin représente

un cas où le système autorise un comportement qui viole la spécification d'exactitude ; par conséquent, le modèle n'est pas correct et le comportement peut être signalé en tant que contre-exemple à la propriété comportementale. Si aucun comportement de ce type n'est détecté (c'est-à-dire qu'une propriété comportementale est vérifiée pour le système), il n'y a aucun comportement qui soit à la fois une exécution légale du modèle et satisfait également la négation de la propriété comportementale.

La Logique Temporelle Linéaire est utilisée pour spécifier et vérifier la plupart des propriétés d'exactitude (de correction) des modèles. C'est la logique formelle utilisée pour la vérification dans SPIN [41, 42]. La logique Temporelle Linéaire (LTL) raisonne sur les traces linéaires à travers le temps. À chaque instant, il n'y a qu'une seule chronologie réelle qui peut se produire. Traditionnellement, cette chronologie est définie comme débutant maintenant, à l'étape actuel, et progressant infiniment dans le futur. LTL étend la logique propositionnelle avec les opérateurs temporels. Les formules du calcul propositionnel sont composées de propositions atomiques et des opérateurs (non = !, et = &&, ou = ||, implique = $\rightarrow$ et équivalent = $\leftrightarrow$). Les formules LTL sont composées d'un ensemble fini de propositions atomiques et d'opérateurs qui incluent les opérateurs du calcul propositionnel ainsi que les opérateurs temporels. Les opérateurs temporels sont (toujours = $\Box$, éventuellement = $\diamond$, jusqu'à = $\mathscr{U}$). LTL est idéale pour la vérification des systèmes qui fonctionnent dans un environnement dynamique tel que les programmes concurrents. LTL nous permet de spécifier naturellement les systèmes distribués en fonction de leur comportement continu.

Les propriétés comportementales (spécifications) sont les formules logiques temporelles que nous définissons pour décrire les comportements souhaités que le système doit avoir. Pour atteindre cet objectif, nous décrivons un ensemble de spécifications pour spécifier un comportement cohérent du système. Ils sont fournis au vérificateur de modèles pour vérifier que le système a la liste des comportements émergents décrits par notre ensemble de spécifications.

Pour l'application de télédiagnostic en neuroscience présenté dans le chapitre 3, nous avons décrit différents types de comportements, qui spécifient les propriétés comportementales (propriétés de correction). Il y a deux types de propriétés fondamentales : les propriétés de sécurité et les propriétés de vivacité. Ces catégories ont été introduites à l'origine par Leslie Lamport [60]. Une propriété de sécurité exprime le sentiment que "quelque chose de mauvais n'arrive jamais". Le système a toujours un bon comportement. Une propriété de vivacité exprime le sentiment que

"quelque chose de bien doit éventuellement arriver". Le système doit avoir éventuellement un bon comportement. La vivacité est définie comme la capacité d'un programme à s'exécuter en temps opportun. Alors que les propriétés de sécurité raisonnent pour atteindre des états spécifiques, les propriétés de vivacité raisonnent sur le flux de contrôle entre les états.

Les propriétés de correction pour l'application de télédiagnostic en neuroscience qui décrivent les comportements souhaités que le système doit avoir sont :

- Lorsqu'un patient présentant un accident vasculaire cérébral est admis à l'hôpital, le SAMU doit être contacté.
- Lorsque le SAMU est sollicité, le patient doit être examiné au préalable par le médecin urgentiste.
- Avant d'appeler le neurologue CHU, la création du dossier médical du patient doit être réalisée.
- Lorsque le diagnostic est initié par le neurologue CHU, les moyens de transfert SMUR et VLS doivent être informés par la décision d'un transfert probable.
- Lorsque le neurologue est sollicité, il doit impérativement fournir un diagnostic (une réponse).
- Si le patient est traité localement par le médecin urgentiste, les ressources de transfert doivent être libérées.
- Le SAMU doit être informé par la décision du neurologue.
- Le choix d'un type de transfert libère automatiquement l'autre.
- Si une décision de transfert est prise, le patient doit être préparé pour le transfert par le médecin urgentiste.

Certaines de ces propriétés exprimées dans des formules en logique temporelle linéaire sont répertoriées dans le listing 5.3.

```
#define Cas_AVC (HA_State==Clinical_Exams)
#define Exams_Achieved (HA_State== Wait_Decision_Making)
#define EMS_Informed (EMS_State== Medical_Record)
#define EMS_Solicit_UHC (EMS_State== Wait_Decision)
#define UHC_Decision (UHC_State== Decision_Making)
#define ACA_Preparation (ACA_State== Wait_Decision)
#define ALS_Preparation (ALS_State== Wait_Decision)
#define local_Traitement (HA_State== Supported_HA)
#define SupportedBy_ALS (ALS_State== Choicem_Supported_ALS)
#define SupportedBy_ACA (ACA_State== Choicem_Supported_ACA)
#define Patient_Preparation_ACA (HA_State== Supported_ACA)
#define Patient_Preparation_ALS (HA_State== Supported_ALS)
#define UHC_Decides_HA (UHC_State==Supported_HA)
#define UHC_Decides_ACA (UHC_State==Supported_ACA)
#define UHC_Decides_ALS (UHC_State==Supported_ALS)
#define UHC_Free (UHC_State== Final_State)


#define Term_State ((HA_State == Final_State) &&(UHC_State == Final_State)
&&(EMS_State==Final_State)&&(ACA_State==Final_State) &&(ALS_State == Final_State))

#define Channels_Empty ((len(HA_EMS)==0)&&(len(HA_UHC)==0)
    &&(len(HA_ACA)==0)&&(len(HA_ALS)==0)
    &&(len(UHC_ACA)==0)&&(len(UHC_ALS)==0)&&(len(EMS_UHC)==0))


LTL prop1 {[](Cas_AVC -> <> EMS_Informed)}

LTL prop2 {[] (Exams_Achieved -> <> EMS_Informed)}

LTL prop3 {[](EMS_Informed -> <> UHC_Decision)}

LTL prop4_1 {[] (UHC_Decision -><>ACA_Preparation)}

LTL prop4_2 {[] (UHC_Decision -><>ALS_Preparation)}

LTL prop5 {(EMS_Solicit_UHC -> <> (UHC_Decides_HA||UHC_Decides_ACA||UHC_Decides_ALS))}

LTL prop6_1 {[](local_Traitement -> <> !(SupportedBy_ALS))}

LTL prop6_2 {[](local_Traitement -> <> !(SupportedBy_ACA))}

LTL prop7 {[](UHC_Decision -> <> EMS_Solicit_UHC)}

LTL prop8_1 {[](SupportedBy_ALS -> <> !(SupportedBy_ACA))}
```

```
LTL prop8_2 {[](SupportedBy_ACA -> <> !(SupportedBy_ALS))}

LTL prop9_1 {[] (UHC_Decides_ACA -> <> Patient_Preparation_ACA)}

LTL prop9_2 {[] (UHC_Decides_ALS -> <> Patient_Preparation_ALS)}

LTL prop10 {[] ((local_Traitement||SupportedBy_ACA ||SupportedBy_ALS) -> <> UHC_Free)}

LTL prop11 { Term_State -> Channels_Empty}

LTL prop12 { SupportedBy_ACA -> (ACA_Alert==Alerted)}

LTL prop13 { SupportedBy_ALS -> (ALS_Alert==Alerted)}

LTL prop14 {UHC_Decision -> (Data_Rec==Recorded)}

LTL prop15 {EMS_Informed -> (Clinical_exams==Performed)}
```

Listing 5.3 : Les propriétés comportementales de l'application de télédiagnostic.

3.5 Vérification de modèles (SPIN)

La vérification exhaustive est une méthode peu pratique pour vérifier l'exactitude des programmes concurrents. Elle génère un grand nombre de calculs, ce qui ne peut pas être réalisé avec un espace mémoire fini. Par conséquent, de telles vérifications exhaustives doivent être effectuées par un outil logiciel efficace qui utilise une quantité minimale de mémoire. Cet outil doit établir avec certitude si un comportement donné ne comporte pas d'erreurs.

SPIN est peut-être le principal exemple d'un tel outil. C'est un vérificateur de modèles qui a été développé depuis de nombreuses années par Gerard J. Holzmann dans [41, 42]. Conçu à l'origine pour vérifier les protocoles de communication, il est depuis devenu l'un des plus puissants outils vérificateurs de modèles. Il est appliqué dans l'industrie pour résoudre des problèmes réels dans le développement de systèmes logiciels distribués à grande échelle. Cet outil a été largement utilisé pour la vérification des systèmes dans de nombreuses applications, y compris les logiciels de systèmes d'exploitation et les protocoles de communication. SPIN est particulièrement bien adapté pour vérifier les systèmes concurrents et distribués qui sont basés sur l'entrelacement d'instructions atomiques. SPIN est un outil logiciel pour analyser et vérifier la cohérence logique

des systèmes concurrents. Le système à vérifier est décrit dans un langage de modélisation appelé Promela (Process Meta Language).

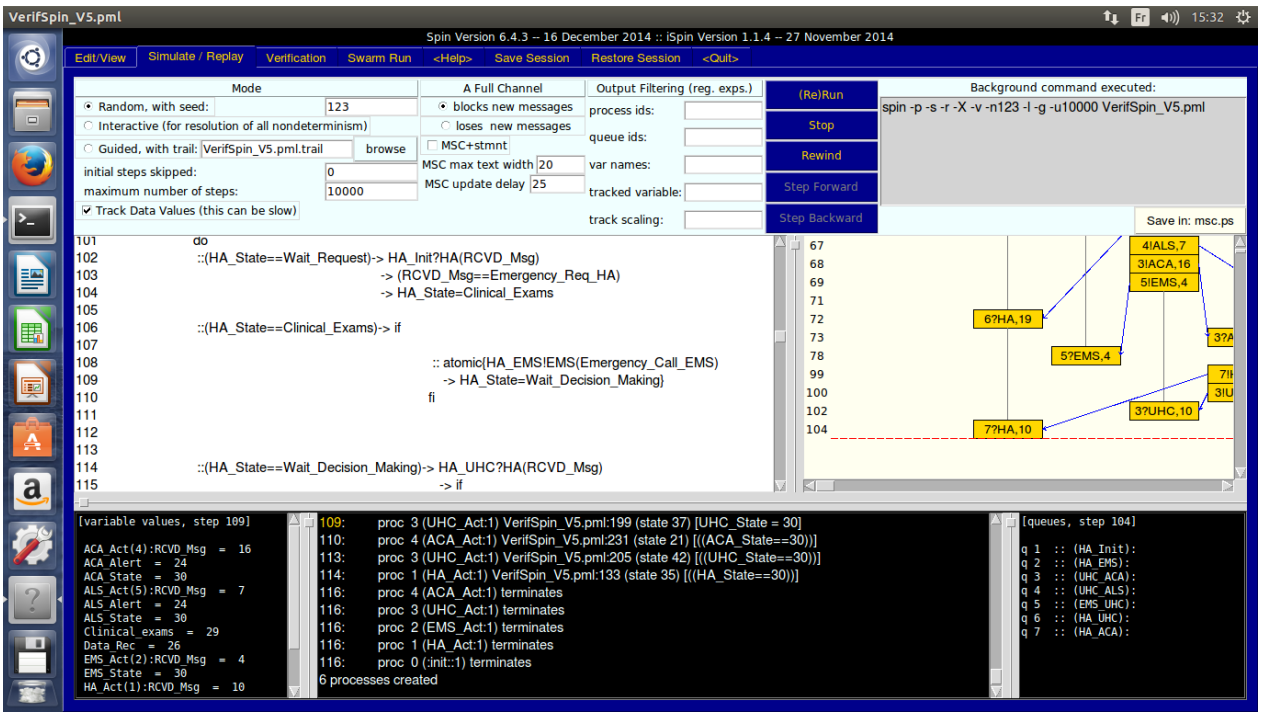

Fig. 5.8 : Les résultats de la simulation aléatoire.

Le programme Promela décrivant l'application de télédiagnostic en neuroscience est généré automatiquement par le processus de transformation modèle-texte. Il s'agit d'un ensemble de processus où chaque processus spécifie le comportement d'un composant du système. SPIN effectue des simulations aléatoires de l'exécution du système de télédiagnostic. Pendant la simulation et la vérification, SPIN vérifie l'absence d'interblocages, de réceptions non spécifiées et de code non exécutable. La simulation de l'exécution de l'application de télédiagnostic dans un environnement concurrent par SPIN n'a détecté aucune erreur. Tous les processus terminent leur exécution sans erreurs, sans interblocages et sans cycles indéfinis. Les résultats de la simulation sont présentés dans la figure (Fig. 5.8). SPIN n'a également trouvé aucune erreur lors de la vérification des composants du système sans les propriétés de correction (safety verification). Les résultats de cette vérification sont montrés dans la figure (Fig. 5.9).

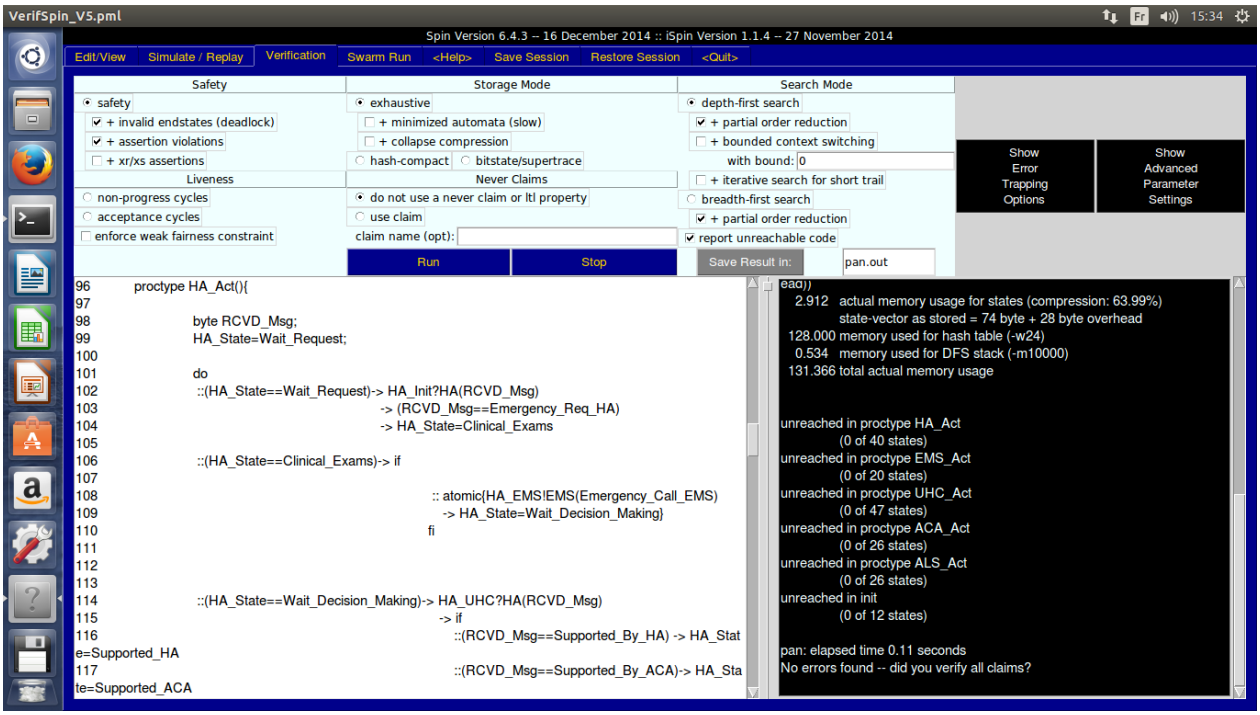

Fig. 5.9 : Les résultats de la vérification sans propriétés de correction.

La figure (Fig. 5.10) montre un scénario de la simulation aléatoire du système de télédiagnostic effectuée par SPIN. Dans ce scénario, le médecin urgentiste HA est informé par l'admission d'un patient avec un accident vasculaire cérébral. Le médecin urgentiste contacte le SAMU (EMS) au moyen d'un *emergency call* pour l'informer de l'admission d'urgence d'un patient qui présente des symptômes suspectant un accident vasculaire cérébral. Au niveau de système d'informations du SAMU, le médecin régulateur contacte le neurologue CHU (UHC) en envoyant un message *diagnostic request*. Le neurologue CHU informe le SMUR (ALS) et le VLS (ACA) d'un éventuel transfert du patient et initie un processus pour établir un diagnostic. Dans ce scénario, le neurologue prend la décision de transférer le patient de toute urgence au Centre Hospitalier Universitaire par le biais d'une ambulance non équipé VLS. Il informe le VLS, le HA et le SMUR de son choix. Enfin, le VLS envoie un message *transfer_ACA* pour informer le médecin urgentiste HA et le neurologue CHU que le transfert a été effectué. Ce scénario est illustré dans la figure (Fig. 5.11) avec un diagramme de séquence.

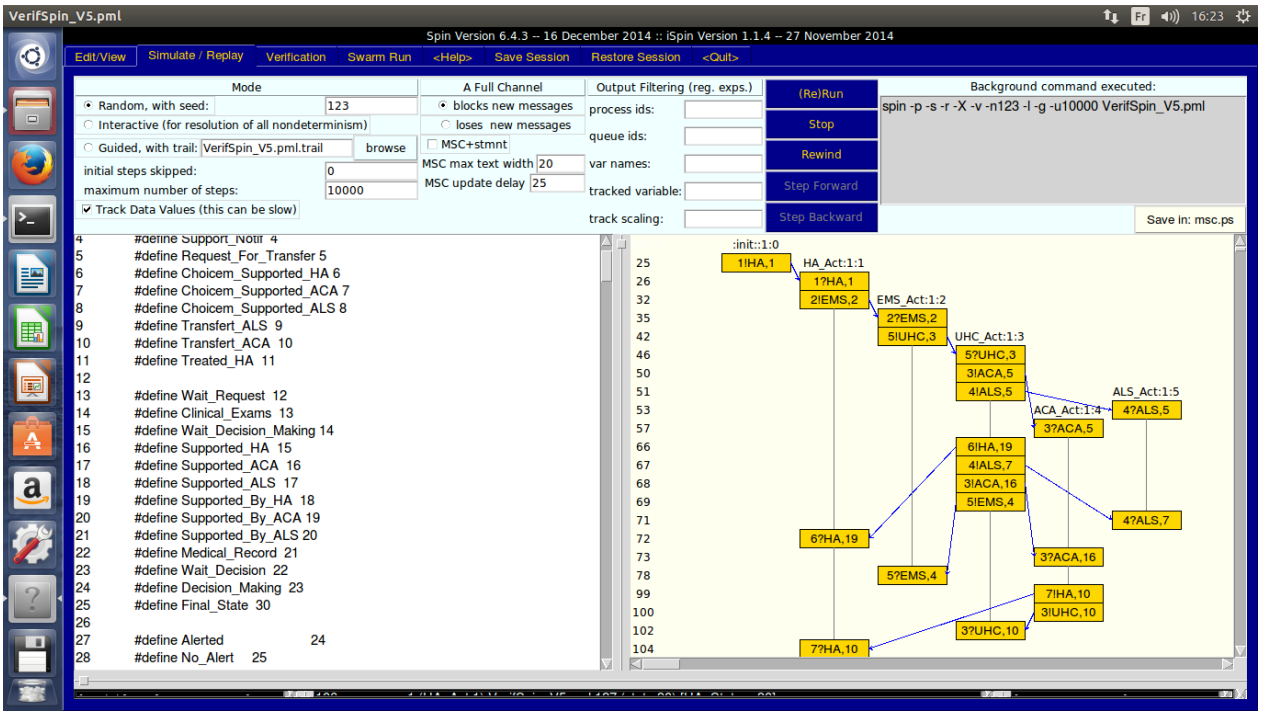

Fig. 5.10 : Un scénario de la simulation aléatoire de l'application de télédiagnostic.

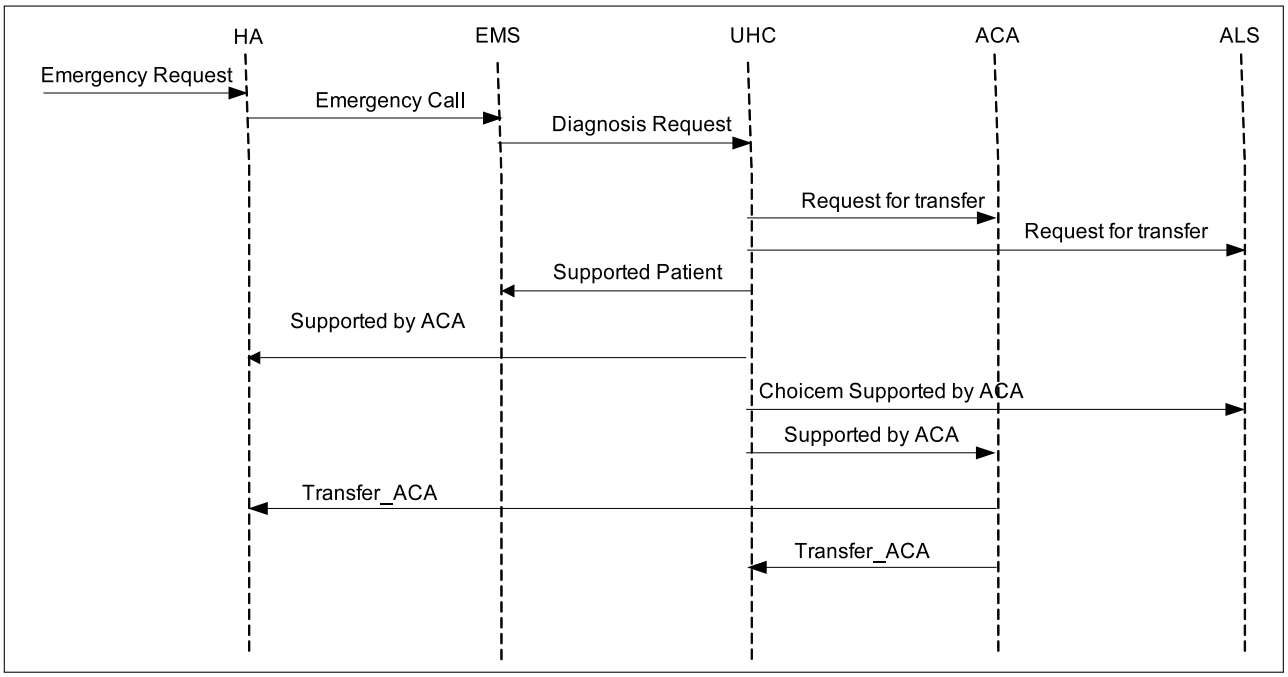

Fig. 5.11 : un scénario de l'application de télédiagnostic.

SPIN peut également effectuer une vérification efficace et exhaustive des propriétés d'exactitude du système. Au cours de la vérification, SPIN vérifie si les propriétés d'exactitude exprimées dans les formules en logique temporelle linéaire (LTL) sont valides pour le modèle du système. Pour l'application de télédiagnostic en neuroscience, nous combinons le programme Promela issu du processus M2T avec les propriétés de correction exprimées dans LTL. Ensuite, nous effectuons la vérification des propriétés d'exactitude décrites au paragraphe 3.4. Le vérificateur de modèles SPIN montre que les propriétés spécifiées sont valides pour les comportements dérivés du système. Les résultats de la vérification sont illustrés par la figure (Fig. 5.12).

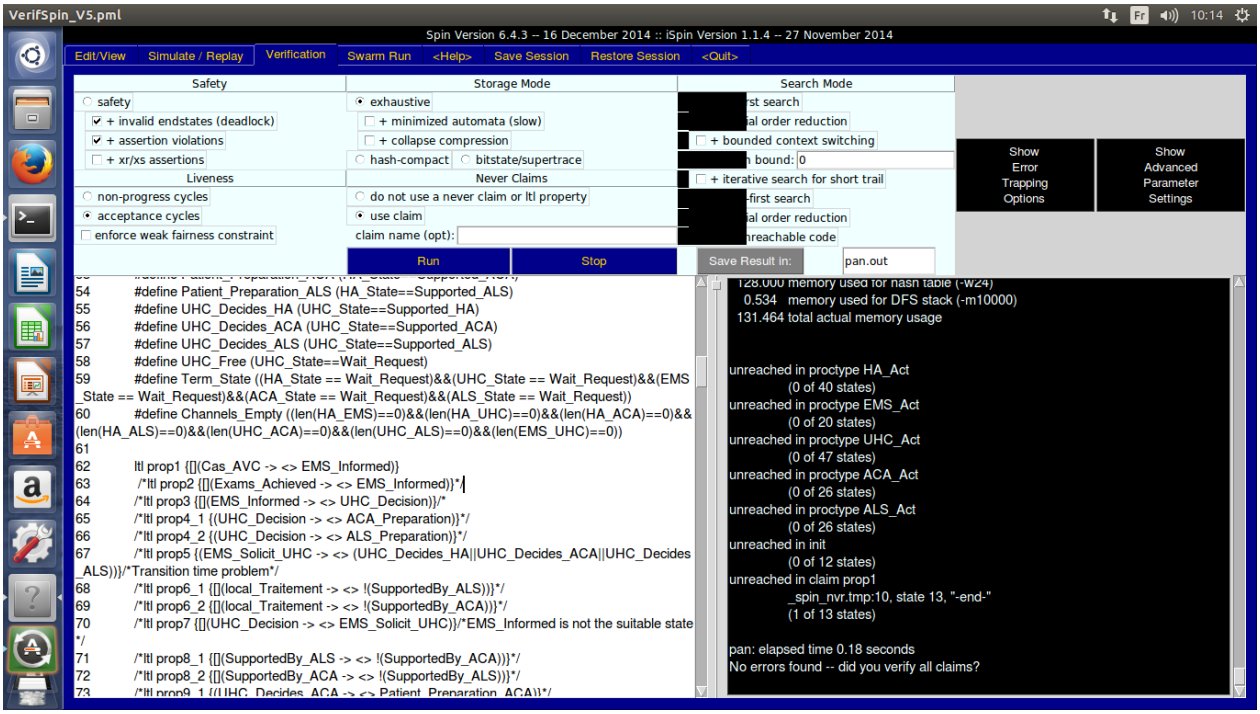

Fig. 5.12 : Les résultats de la vérification des propriétés comportementales.

4 Conclusion

Le travail présenté dans ce chapitre propose une approche dirigée par les modèles pour vérifier et valider les comportements dérivés d'un système distribué. La vérification et la validation des comportements dérivés permettent de s'assurer que la collaboration de ces comportements satis-

fait les exigences initiales du système. Cette approche étend et complète l'approche de dérivation et fournissent à eux deux une méthodologie de développement fiable pour le développement et la maintenance des applications distribuées. En outre, l'utilisation d'une approche d'ingénierie logicielle pour prendre en charge un tel processus de dérivation et de vérification peut conduire à des modèles indépendants de toute plateforme d'exécution. Ce processus de vérification dirigé par les modèles est principalement basé sur la transformation automatique des modèles dérivés du système en modèles spécifiques requis par un vérificateur de modèles. Un programme Promela est ensuite généré à partir de ces modèles spécifiques par une transformation modèle-texte. Le programme Promela issu est combiné avec les propriétés d'exactitude pour réaliser l'analyse et la vérification du système. Les propriétés sont définies sur le comportement global du système. Elles sont exprimées en logique temporelle linéaire (LTL).

Notre approche a été validée à travers une application réelle qui consiste en un système distribué collaboratif dédié au diagnostic à distance pour les cas d'urgence en neuroscience. Ce système a été utilisé pour tester et illustrer les concepts discutés dans l'approche proposée. La simulation aléatoire des comportements dérivés du système effectuée par le vérificateur de modèles SPIN, nous a permis d'explorer plusieurs scénarios d'exécution du système. Les résultats ont montré que les exécutions de ces comportements dans un environnement distribué ne génèrent aucun message inattendu et que tous les messages émis sont reçus. De plus, l'exécution ne mène à aucun blocage, elle se termine toujours de manière sûre et ne génère pas d'erreur d'exécution. La vérification des propriétés d'exactitude par SPIN a montré l'absence d'erreurs et que les comportements dérivés satisfont la spécification globale du système. Toutes les propriétés d'exactitude exprimées sur le comportement global du système sont valides pour les comportements dérivés. Cette vérification nous a permis de valider le processus de dérivation. En conclusion, le processus de dérivation et sa vérification constituent une approche robuste et fiable pour la construction et la maintenance des systèmes distribués.

Chapitre 6

Une approche dirigée par les modèles pour la conception des réseaux WBSN pour la surveillance médicale

1 Introduction

Un réseau de capteurs corporels sans fil (Wireless Body Sensor Network (WBSN)) est un réseau de capteurs sans fil attachés au corps humain afin de permettre la surveillance des paramètres vitaux du corps humain et son environnement. La conception et le développement de tels systèmes pour la surveillance médicale (health monitoring) ont récemment reçu beaucoup d'attention, dans les domaines de la recherche et de l'industrie. Cet engouement est principalement motivé par des soins de santé très coûteux et par les progrès récents accomplis dans le développement de dispositifs miniatures pour la surveillance médicale. L'existence d'une approche de conception explicite devrait être très bénéfique pour le développement et la maintenance de ces systèmes.

Dans ce chapitre, nous présentons un système de surveillance médicale préventive. Ce système est basé sur une architecture composée de nœuds hétérogènes et réalise à la fois une surveillance quotidienne et continue des malades dans un hôpital ainsi que des contrôles spécifiques.

Nous avons défini un modèle pour décrire le comportement global du système WBSN. L'approche de dérivation du comportement dirigée par les modèles est utilisée pour obtenir le comportement de chaque nœud dans le système WBSN à partir du comportement global du système. Cette approche permet aux développeurs d'obtenir une conception du système WBSN à partir de la spécification globale de ses besoins. En plus de l'approche de dérivation du comportement, nous utilisons l'approche de vérification dirigée par les modèles pour valider et vérifier les résultats du processus de dérivation. Cette approche de vérification est nécessaire pour vérifier si le système obtenu après la transformation se comporte comme prévu.

2 Les réseaux de capteurs corporels sans fil (WBSN)

Les systèmes de surveillance médicale sont dédiés à la surveillance de l'état de santé d'un patient. La demande croissante de solutions pour la surveillance médicale est motivée par le coût élevé des soins de santé, la croissance et le vieillissement de la population et l'augmentation du nombre de patients atteints de maladies chroniques dans le monde. En raison de ces facteurs, les soins de santé traditionnels ne peuvent pas fournir l'évolutivité et la scalabilité souhaitées. Par conséquent, il existe un besoin de solutions de surveillance et suivi médicaux performantes, peu coûteuses et efficaces.

Un réseau de capteurs sans fil (WSN) se compose d'un certain nombre de capteurs sans fil muni d'une infrastructure de communication. Les capteurs mesurent et enregistrent différents paramètres à divers endroits. Généralement, de telles mesures comprennent des mesures environnementales (par exemple, la qualité de l'air, la température et l'humidité) et des mesures des fonctions vitales du corps humain (par exemple, les signaux du cœur et du cerveau). Un WSN est capable de détecter, de traiter et de communiquer. Les données collectées par les capteurs sont transmises à une station de base ou à un ordinateur, pour aider à analyser et à réagir en fonction des conditions observées dans un environnement. Les réseaux de capteurs sans fil (WSNs) ont ouvert la voie à des progrès dans divers aspects de la surveillance. De nombreuses applications de surveillance basées sur les WSN ont été développées dans différents domaines d'application pour collecter des données sur l'environnement surveillé. Ces applications incluent, sans s'y limiter, les applications

militaires [123], les applications de surveillance du climat [83], les applications dans les réseaux sous-marins [67] et les applications de surveillance médicale [91].

Les systèmes de surveillance médicale utilisant les réseaux de capteurs sans fil ont été conçus pour assurer une surveillance des fonctions vitales du corps humain. Les réseaux de capteurs corporels sans fil (WBSN) pour la surveillance médicale permettent un suivi constant des paramètres de santé des personnes. Ils peuvent comprendre différents types de capteurs : capteurs attachés au corps humain et capteurs ambiants qui sont déployés autour du patient. Généralement, les conditions surveillées par les systèmes WBSN incluent le diabète, l'arythmie cardiaque, l'apnée du sommeil, l'asthme et les signaux ou paramètres physiologiques (tels que la tension artérielle, la température corporelle, l'activité cardiaque, la respiration, la saturation en oxygène et l'activité cérébrale). Ces paramètres peuvent être surveillés à distance par les médecins et les infirmiers sans affecter les activités des patients. En fait, ces systèmes ont considérablement réduit les erreurs humaines, permettant une meilleure compréhension de l'origine des maladies.

Divers réseaux de capteurs corporels sans fil (WBSNs) ont été développés ces dernières années. Ils visent à surveiller une maladie spécifique ou un ensemble de signaux physiologiques des patients à leur domicile [54, 69, 122], comme ils peuvent effectuer la reconnaissance quotidienne de l'activité physique et la surveillance de l'activité cardiaque [99]. Ces systèmes doivent satisfaire à certains critères tout en fonctionnant sous des limitations de ressources matérielles importantes [5, 90] :

1. La sécurité et la confidentialité des données médicales mesurées doivent être garanties par le système ; et

2. La consommation d'énergie doit être minimisée pour augmenter la durée de vie du système.

La conception des systèmes WBSN est une tâche très complexe. Les concepteurs de tels systèmes devraient prendre en compte ces limitations lors de la construction de systèmes de plus en plus fiables. Cependant, la majorité des études dans ce domaine sont axées sur des problèmes de mise en œuvre, et elles abordent rarement une méthodologie pour la construction et la maintenance de tels systèmes. La plupart de ces systèmes sont développés en sélectionnant d'abord la

plateforme cible la plus appropriée et, ensuite, son système d'exploitation spécifique et son langage de programmation. L'existence d'une architecture explicite devrait être très bénéfique pour la construction et la maintenance des grands systèmes WBSN.

Les travaux présentées dans [44, 101] proposent l'utilisation du système data-MULE. Les études de [52, 129] utilisent le schéma de transport de messages. Dans ces deux approches, le Data-MULE et les message ferries recueillent les données provenant des nœuds capteurs tout en se déplaçant autour du réseau. Ces approches sont conçues uniquement pour fournir le service de relais de messages dans les réseaux. Dans le même temps, un récepteur mobile peut consommer les données collectées pour ses propres fins ou mettre les données à la disposition d'utilisateurs distants. Dans [89, 121], un ou plusieurs récepteurs mobiles se déplacent à travers le réseau WSN pour collecter les données à partir des nœuds capteurs statiques déployés dans la zone du réseau.

Les réseaux WSN sont très efficaces pour prendre en charge diverses applications. Diverses publications dans ce domaine de recherche ont été proposées pour surveiller les paramètres vitaux du corps humain à la maison, pour les patients âgés ou dans un centre de santé. Pour un suivi continu et en temps réel, les auteurs de [63] ont développé une chemise intelligente qui mesure les signaux d'ECG (électrocardiogramme). La chemise est composée de tissus conducteurs pour obtenir le signal du corps à partir d'électrodes et de capteurs pour la surveillance des données médicales. Les données observées et mesurées sont transmises dans un réseau ad hoc pour la surveillance à distance. Un appareil Android est utilisé pour analyser les signaux ECG à partir d'un terminal de surveillance mobile, comme mentionné dans [39]. Dans [34, 124, 127, 128], des applications de réseaux de capteurs sans fil pour la surveillance médicale à domicile sont présentées. Ils assurent un suivi médical continu du patient à domicile sans restreindre ses activités et ses mouvements. Ces études sont basées sur l'utilisation d'un collecteur de données statique pour collecter les informations. Tous ces travaux sont axés uniquement sur les problèmes de mise en œuvre et d'implémentation.

Dans [62], les auteurs ont proposé un système de surveillance médicale basé sur un réseau de capteurs sans fil pour la collecte et la diffusion de données de capteurs médicaux. Ce système permet le stockage, la corrélation et la diffusion des données ainsi que la gestion des alertes en temps opportun, lorsque les paramètres sont violés. La mise en œuvre et l'analyse d'une application e-Health basée sur WSN a été décrite dans [125]. Les auteurs de [68] illustrent la conception et la

mise en œuvre d'un système de surveillance médicale en ligne. L'architecture de ce système est basée sur des dispositifs intelligents et des réseaux de capteurs sans fil pour l'analyse en temps réel de divers paramètres des patients.

Le travail présenté dans cette thèse contribue à la conception et la validation des applications de surveillance médicale basées sur les WBSNs. Nous proposons d'abord une architecture composée de nœuds hétérogènes pour construire un système de surveillance médicale préventive. Le système de surveillance médicale proposé [38] est conçu pour effectuer une surveillance quotidienne et continue ainsi que des contrôles spécifiques pour les patients dans un hôpital. C'est une architecture WBSN basée sur un collecteur de données mobile. Le collecteur mobile est utilisé pour économiser les ressources (consommation d'énergie) des capteurs. Les nœuds capteurs ne peuvent être réveillés que lorsqu'ils sont en présence du collecteur de données. Cette solution a été étendue par une présence humaine (l'infirmier). Cette présence humaine peut ajouter de la valeur à cette architecture en analysant les données collectées et en prenant des décisions urgentes. Deuxièmement, l'approche d'ingénierie dirigée par les modèles (IDM) pour la dérivation est adoptée pour dériver le comportement des composants du système WBSN (capteurs, collecteurs de données, etc.) à partir des exigences globales du WBSN pour différents scénarios de fonctionnement de l'hôpital. Le niveau des exigences et sa transformation au niveau conceptuel constituent la première étape de la génération d'une conception de qualité d'un système complexe.

Les travaux de [1, 59, 64, 120] présentent une approche MDE pour le développement d'applications WSN. Ils ne traitent que la conception et le développement des WSN statiques. Ces études se concentrent uniquement sur les transformations de modèles et leurs résultats et ne fournissent pas les outils et les moyens pour vérifier les transformations. En contraste à ces études, nous utilisons une approche dirigée par les modèles pour dériver le comportement WBSN à partir de sa spécification globale et vérifier si le système dérivé se comporte correctement selon sa spécification globale, afin d'augmenter les performances du système et la qualité de service (QoS). La vérification valide la conception du système dérivé en vérifiant si elle répond aux exigences du système et en rendant le système plus fiable. Il est très souhaitable d'effectuer une vérification le plus tôt possible dans le processus de conception du système, évitant ainsi la découverte possible d'erreurs dans un système achevé qui nécessiterait une nouvelle conception.

3 Architecture du WBSN pour la surveillance médicale

Un hôpital est composé de plusieurs étages. Chaque étage comprend un certain nombre de chambres de patients. Il est structuré comme indiqué sur la figure (Fig. 6.1). Les capteurs sont rattachés au corps du patient et déployés autour dans son environnement. Ils peuvent être classés en deux types : les capteurs médicaux et les capteurs environnementaux. Les capteurs médicaux surveillent les paramètres vitaux du patient, tandis que les capteurs environnementaux surveillent les paramètres de la chambre, incluant la température ambiante, les niveaux d'oxygène et au-delà. Un infirmier équipé d'un nœud sans fil, par exemple un assistant personnel numérique (PDA), un téléphone intelligent ou un PC de poche, collecte les données mesurées par les capteurs. Ce collecteur de données mobile peut soit consommer les données collectées pour son propre usage (afficher les informations physiologiques sur une interface utilisateur) ou les transmettre à un centre médical. Les médecins consultent le centre médical pour afficher le statut du patient et prendre les décisions appropriées.

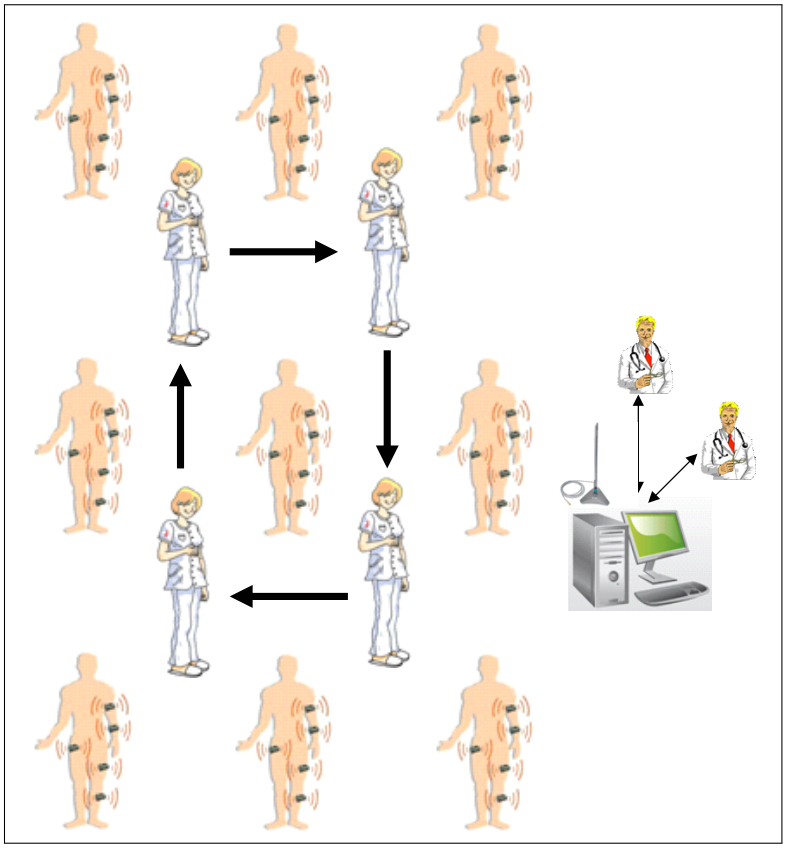

Fig. 6.1 : Architecture d'un étage du WBSN.

Il y a quatre utilisateurs de base dans cette architecture : les patients, les infirmiers, les médecins et les techniciens.

1. Les patients sont équipés de capteurs. Ces capteurs sont des capteurs médicaux et environnementaux. Les capteurs médicaux sont attachés au patient pour mesurer ses paramètres vitaux, tandis que les capteurs environnementaux sont déployés autour du patient dans divers endroits de la chambre. Ces capteurs produisent des valeurs brutes de données qui sont transmises au collecteur de données par des liaisons sans fil. Prisent ensemble, ces valeurs présentent la situation en temps réel du patient à tout moment. Les nœuds capteurs avec le collecteur de données forment un réseau de capteurs du corps (BSN) dans une topologie en étoile.

2. Les infirmiers sont équipés d'un collecteur de données mobile. Ce collecteur de données mobile collecte les données mesurées à partir des capteurs et les convertit en métadonnées significatives. Les données du capteur contiennent uniquement les valeurs des paramètres mesurés, sans aucune information sur l'identité du patient ni le temps et la date de la mesure. Le collecteur de données mobile ajoute des valeurs, telles qu'un identifiant unique, le type du paramètre surveillé, l'heure et l'unité de mesure, pour rendre l'information significative. Ensuite, il transmet les métadonnées au centre médical. La communication entre le BSN et le centre médical peut être accomplie en utilisant WLAN, GSM, ou d'autres systèmes, qui peuvent offrir une large couverture. L'utilisation d'un dispositif intelligent pour collecter les données peut également servir à effectuer un traitement local des données afin d'analyser l'état de santé du patient.

3. Les médecins consultent le centre médical pour afficher le statut du patient. Ils prescrivent des médicaments ou fournissent des suggestions au patient, qui sont en corrélation avec les valeurs des paramètres mesurés par les capteurs.

4. Les techniciens consultent le centre médical pour détecter toute défaillance de capteurs. Ils répondent à une alarme ou au dysfonctionnement d'un capteur.

L'architecture du système est définie par trois couches en fonction de la fonctionnalité des composants intervenant dans chaque couche. La figure (Fig. 6.2) illustre les trois couches utilisées

dans cette architecture :

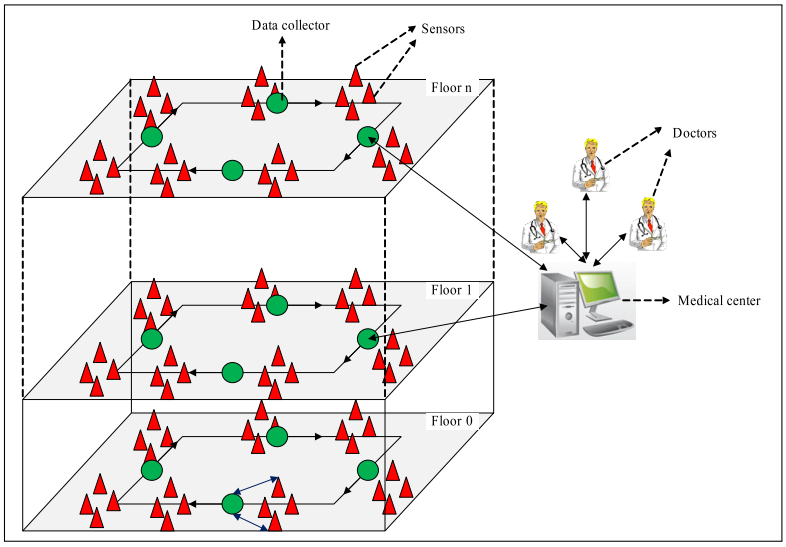

Fig. 6.2 : Architecture du WBSN pour la surveillance médicale.

1. La couche de perception : La première couche se trouve au bas de la hiérarchie, elle se compose de capteurs médicaux et environnementaux qui recueillent des données en temps réel. Les capteurs médicaux sont utilisés pour analyser l'état de santé du patient en mesurant divers paramètres vitaux du corps humain. Ces capteurs comprennent le capteur de pression artérielle, le module ECG et le thermomètre ...etc. Les capteurs environnementaux surveillent l'environnement du patient et assurent que le patient se trouve dans un environnement sain. Ces capteurs comprennent des capteurs de détection de gaz, des capteurs de température, ...etc. Les données accumulées par les capteurs sont collectées par le collecteur de données, qui les convertit en informations significatives. Le collecteur de données ajoute des informations aux mesures pour distinguer les valeurs brutes ; par exemple, il ajoute un identifiant unique du patient pour distinguer quel rapport est pour quel patient. Ensuite, il transmet les métadonnées au centre médical. Par conséquent, le collecteur de données agit comme une passerelle dans le système entre la couche 1 et la couche 2 de l'architecture.

2. La couche de traitement (centre médical) : Cette couche stocke les antécédents médicaux du patient ainsi que les enregistrements actuels des paramètres surveillés. Ces dossiers contiennent (i) des rapports médicaux, (ii) des prescriptions médicales et (iii) des médicaments auxquels

le patient est allergique. Ce stockage joue un rôle central dans les réponses d'urgence et dans le système de surveillance hospitalière car il permet de comparer les données collectées par les capteurs avec les seuils mémorisés pour les valeurs paramétriques. Les données de cette couche sont utilisées par les médecins et les applications de service (troisième couche de l'architecture).

3. Les services de santé (utilisateurs et services) : La troisième couche du système est une couche qui offre des services pour l'utilisation des données collectées fournies par la couche de traitement. Elle se compose d'infirmiers, de médecins et de techniciens. Cette couche offre plusieurs services au patient. Il s'agit de prescrire des médicaments ou de fournir des suggestions au patient par les médecins. Ces prescriptions médicales sont faites en fonction des valeurs des données reçues des capteurs. Il y a aussi une réponse immédiate des médecins et des infirmiers en cas d'urgence et des techniciens en cas de défaillance de capteurs. Le personnel surveille ces alarmes pour chacun des patients et fournit au patient les médicaments requis pour remédier à la situation d'urgence.

L'architecture proposée minimise les erreurs de mesure en collectant les données par un infirmier équipé d'un collecteur de données. La présence d'êtres humains permet d'analyser des données mesurées, de noter l'état du patient et de réaliser une maintenance du réseau si nécessaire (par exemple, corriger une mauvaise position d'un capteur). Une assistance est directement demandée aux autres soignants lorsqu'une anomalie est observée. Dans le fonctionnement hospitalier ordinaire, les capteurs ne sont réveillés qu'en présence du collecteur de données mobile, ce qui minimise la consommation d'énergie dans le système WBSN. Dans ce cas, l'infirmier recueille les données des capteurs une fois par jour et les transmet au centre médical. Les médecins peuvent consulter les données pour observer l'état d'un patient. Dans d'autres cas, les capteurs peuvent être réveillés périodiquement pour effectuer des mesures, les stocker dans leurs mémoires privées et les transmettre par la suite au collecteur. Ainsi, le fonctionnement ordinaire du système peut être enrichi par de nouvelles contraintes, telles que le traitement des alarmes, des cas critiques et des urgences.

Le système WBSN peut donner lieu à plusieurs schémas de fonctionnement (fonctionnement hospitalier ordinaire, fonctionnement avec des mesures périodiques, fonctionnement avec traite-

ment d'alarmes, fonctionnement avec gestion des urgences, etc.). Chaque scénario décrit un comportement global du système WBSN. Un tel comportement global peut être décomposé en comportements partiels qui sont effectués par les différents composants ou rôles du système (Capteur, Collecteur de données et Docteur). Une approche de transformation automatique est nécessaire pour dériver le comportement de ces composants à partir du comportement global du système.

La nature distribuée des WBSN et les différents scénarios qu'ils pourraient avoir rendent difficile la conception et le développement de ces systèmes. De nombreuses propositions de systèmes WBSN pour les soins de santé à domicile ont été élaborées. Elles sont principalement axées sur les problèmes de mise en œuvre, et elles abordent rarement le processus de développement. L'utilisation d'une approche d'ingénierie logicielle pour réaliser la conception et la vérification des systèmes WBSN contribue à résoudre ce problème en permettant aux concepteurs de construire et de valider des applications WBSN.

La section suivante présente l'utilisation de l'approche dirigée par les modèles proposée pour dériver le comportement des composants du système WBSN en transformant le modèle global des exigences du WBSN en modèles de comportement de chaque composant du système [38]. L'approche formelle de vérification de modèles proposée dans le chapitre 5 est utilisée par la suite pour vérifier que les comportements dérivés satisfont la spécification globale du système.

4 L'approche de dérivation et vérification dirigée par les modèles pour les systèmes WBSN

L'approche dirigée par les modèles proposée dans le chapitre 3 permet de dériver le comportement de chaque composant du système. Le comportement de chaque composant du système WBSN est obtenu en transformant le modèle global des exigences du système en modèles représentant le comportement des différents composants du système. L'approche de vérification décrite dans le chapitre 5 permet d'effectuer une vérification formelle des comportements dérivés des différents composants du système. Elle permet également de garantir que la collaboration des comportements dérivés satisfait le modèle des exigences initial. La figure (Fig. 6.3) montre une vue d'ensemble de

l'approche de dérivation et de vérification dirigée par les modèles.

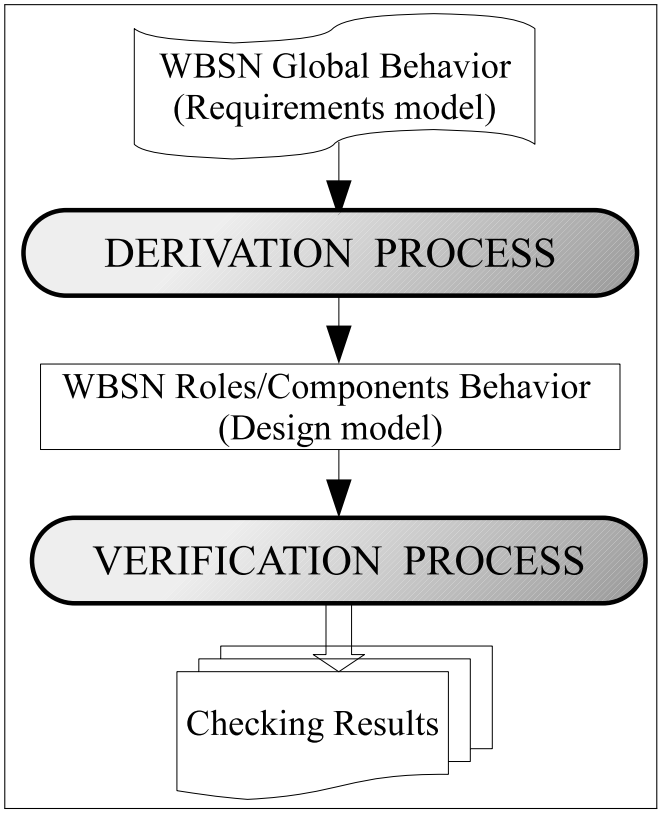

Fig. 6.3 : Approche de dérivation et de vérification dirigée par les modèles.

5 Processus de dérivation

Dans le système WBSN pour la surveillance médicale, chaque scénario de fonctionnement est exprimé avec des collaborations entre les composants du système (capteur, collecteur de données, médecin, centre médical). Dans l'approche dirigée par les modèles proposée pour la dérivation du comportement des composants d'un système, le développeur spécifie, dans la première étape du processus de dérivation, le modèle des exigences globales qui décrit le comportement global du WBSN. La figure (Fig. 6.4) représente le comportement global du WBSN pour le fonctionnement ordinaire de l'hôpital. Ce comportement est modélisé par un diagramme d'activités dont les activités principales sont des collaborations et des sous-collaborations.

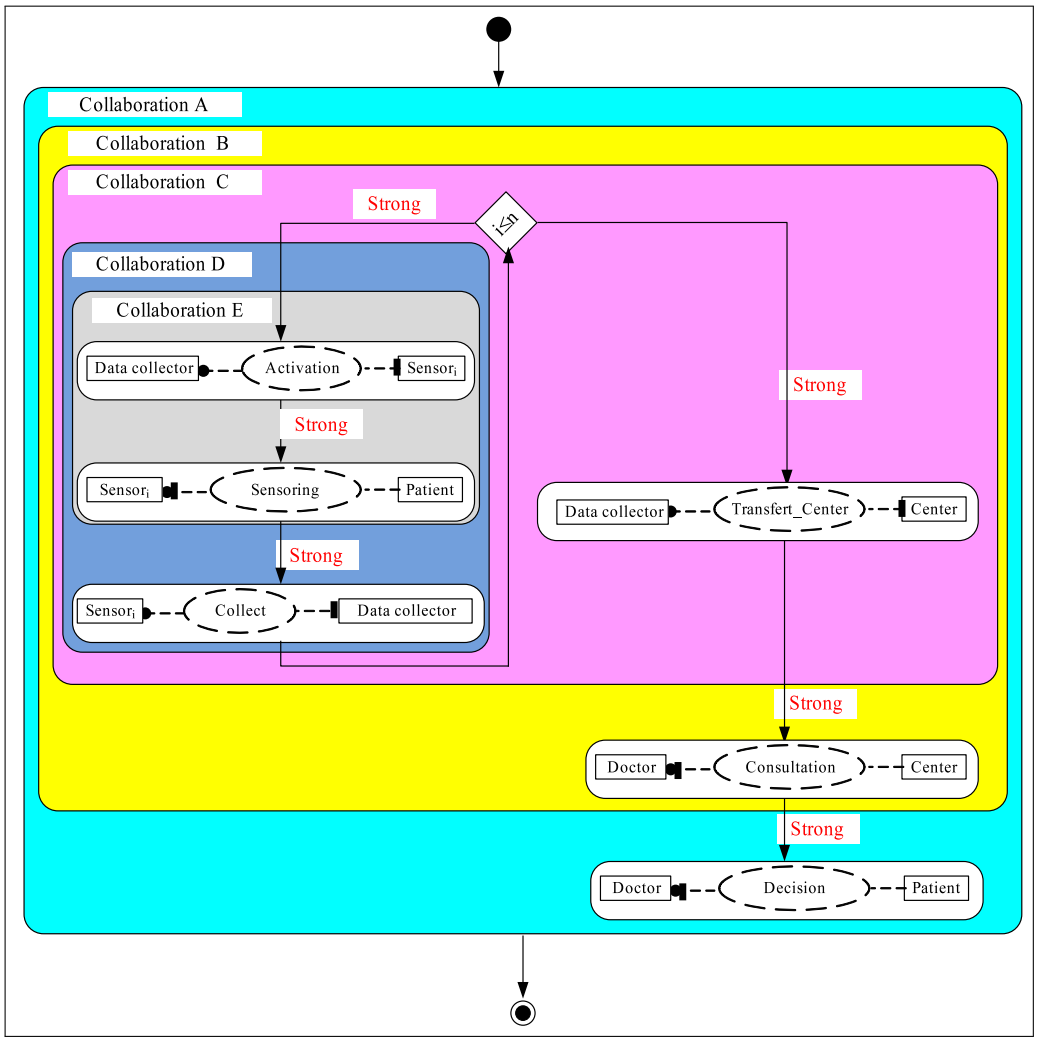

Fig. 6.4 : Comportement global du système WBSN.

La deuxième étape du processus de dérivation réalise la transformation modèle-modèle. La transformation modèle-modèle (M2M) transforme automatiquement le modèle des exigences du WBSN en modèles de conception. Le processus de dérivation intègre les messages de synchronisation dans les comportements dérivés du WBSN pour assurer la coordination entre les composants du système. Le résultat du processus de dérivation est une machine d'états UML pour chaque composant du système. Les figures (Fig. 6.6 et Fig. 6.5) décrivent respectivement le comportement du capteur et celui du collecteur de données.

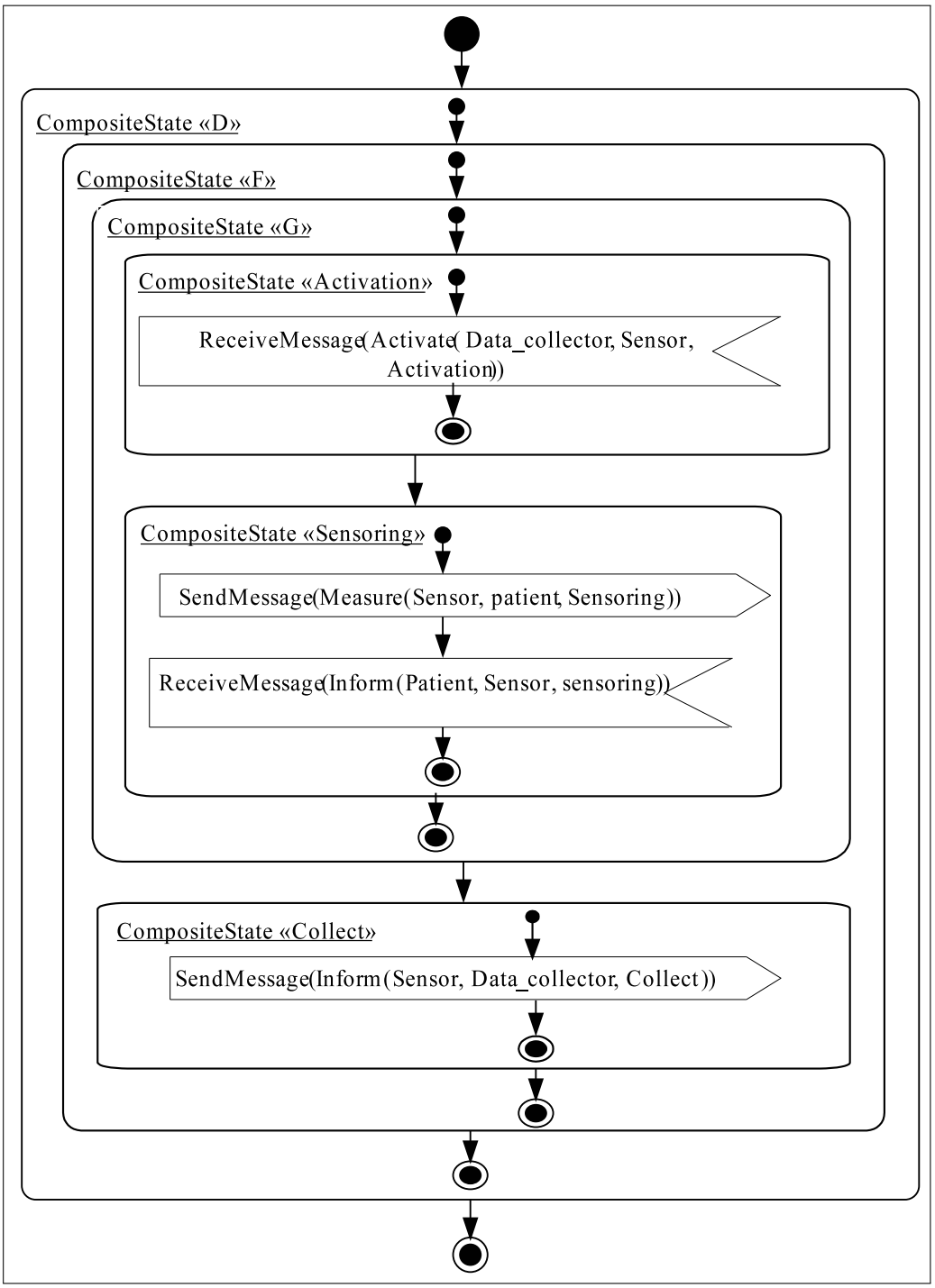

Fig. 6.5 : Comportement dérivé du capteur.

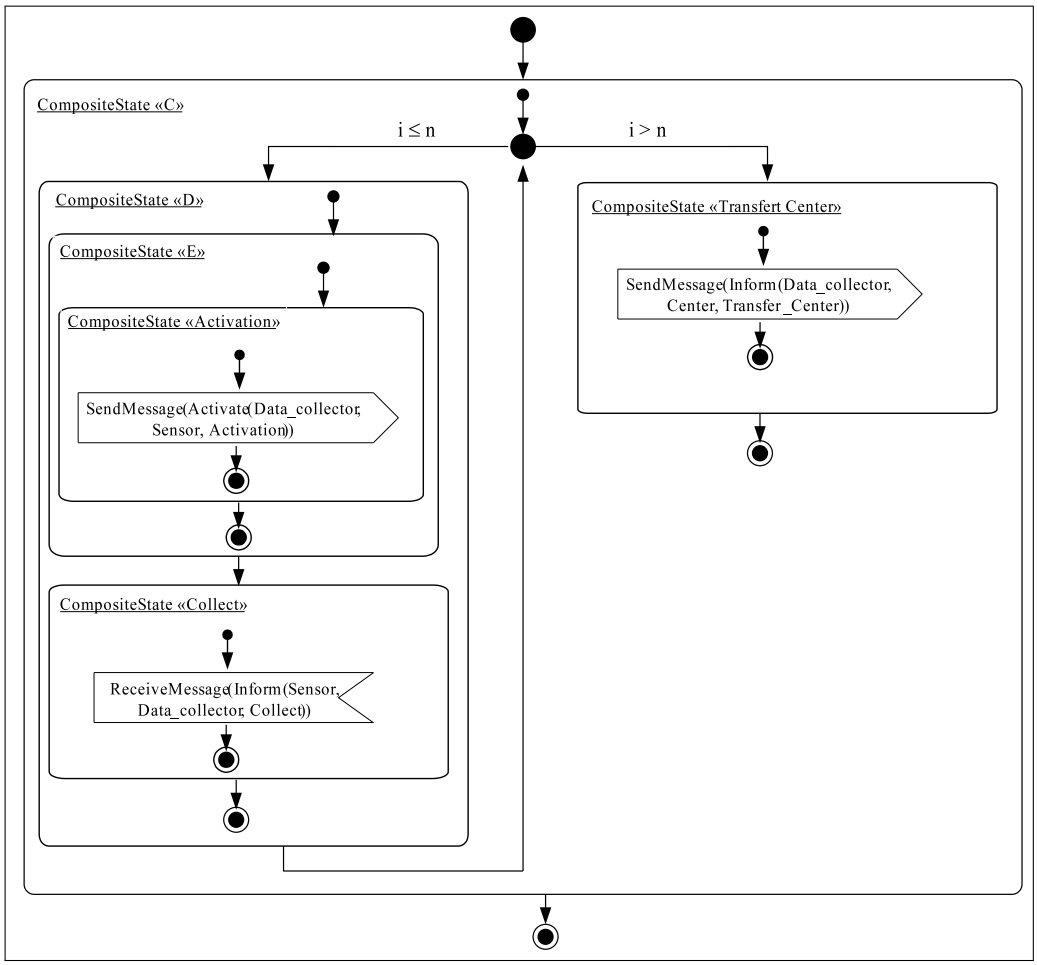

Fig. 6.6 : Comportement dérivé du collecteur de données.

6 Processus de vérification

L'approche dirigée par les modèles proposée pour la vérification des comportements dérivés des différents composants du système WBSN [38] débute par la transformation modèle-modèle (M2M). Cette transformation consiste à transformer automatiquement les modèles de conception issus du processus de dérivation, qui décrivent le comportement de chaque composant WBSN, en modèles de machines à états finis (FSM) spécifiques pour la vérification. Cette transformation M2M a été développée dans le but de générer les différents concepts d'une machine à états finis décrivant le

comportement de chaque composant du WBSN. Ces modèles sont utilisés ensuite dans le processus de la transformation de modèle à texte pour la génération automatique du code Promela. La figure (Fig. 6.7) illustre les machines à états finis communicantes qui sont générées pour les composants (capteur et collecteur de données) du système WBSN.

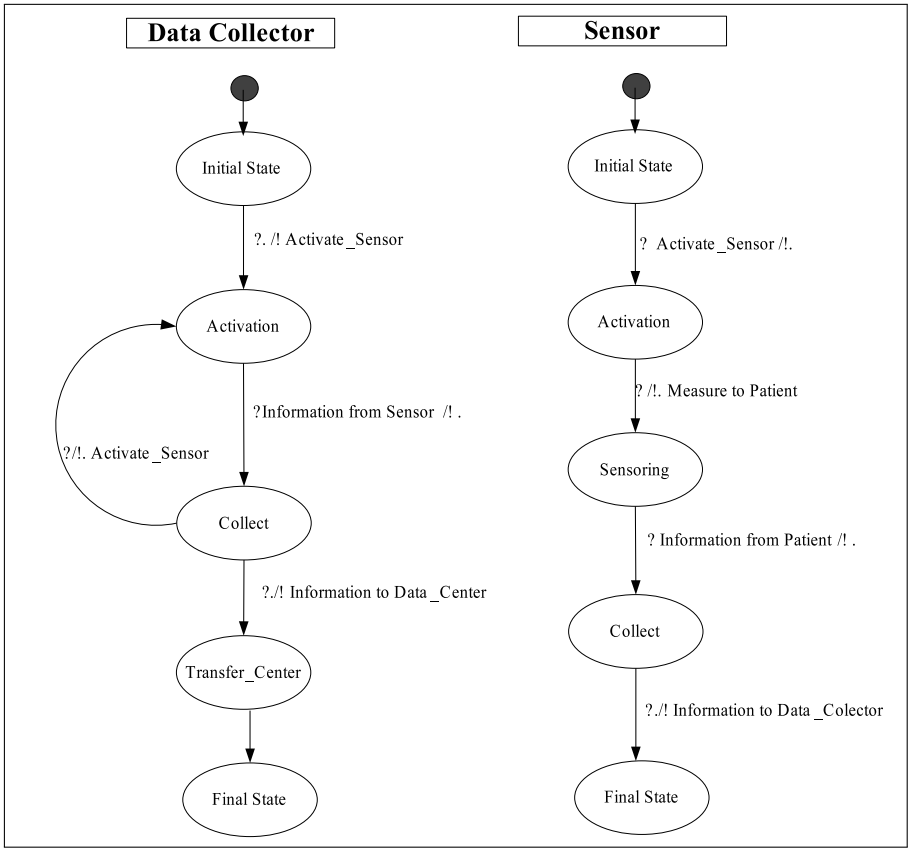

Fig. 6.7 : FSM du capteur et du collecteur de données.

L'automate de la figure (Fig. 6.7) spécifie le comportement du collecteur de données. Les états sont étiquetés avec les noms des collaborations auxquelles il y participe. Les transitions sont étiquetées par les actions qui amènent le système à changer d'états. Le comportement du collecteur de données commence à l'état *Initial_state*. Lorsque le collecteur de données envoie un message *Activate_Sensor*, il passe à l'état *Activation*. A ce stade, il attend la réception de données (*Information_data*) du capteur pour passer à l'état *Collect*. Par la suite, il décide d'activer un autre capteur et passe à l'état *Activation* si nécessaire. Une fois les données sont collectées, il transfèrent les informations au centre de données et passent à l'état *Transfer_center* et enfin à l'état *Final_state*.

La deuxième étape de l'approche consiste à réaliser une transformation modèle-texte (M2T). Cette transformation génère automatiquement un programme Promela à partir des modèles de machine à états finis communicante. Ce programme sera validé par le vérificateur de modèles SPIN. Le programme Promela généré est constitué d'un ensemble de processus où chaque processus spécifie le comportement d'un composant du WBSN. Le listing 6.1 décrit les variables globales et les canaux de messages nécessaires, et le listing 6.2 décrit le code du processus correspondant au collecteur de données et celui du processus *Init*.

```
#define Activate_sensor     1
#define Information_Data    2
#define Request_Measure     3
#define Measure_Data        4
#define Treatment           5
#define Request_Data        6
#define Data                7
#define Initial_State       12
#define Activation          13
#define Sensoring           14
#define Collect             15
#define Transfer_Center     16
#define Consultation        17
#define Decision            18
#define Final_State         30

chan Collector_Sensor = [1] of { mtype, byte };
chan Collector_Center = [1] of { mtype, byte };
chan Sensor_Patient   = [1] of { mtype, byte };
chan Doctor_Center    = [1] of { mtype, byte };

mtype {Collector, Sensor, Center, Doctor}
byte Collector_State, Sensor_State, Center_State
```

Listing 6.1 : Les variables globales et les canaux de messages.

```
proctype Collector_Act(){

byte RCVD_Msg;
Collector_State=Initial_State;

do
::(Collector_State==Initial_State)
      -> atomic{ Collector_Sensor!Sensor(Activate_sensor)
```

```
      -> Collector_State=Activation}

::(Collector_State==Activation)
     -> Collector_Sensor?Collector(RCVD_Msg)
     -> (RCVD_Msg==Information_Data)
     -> Collector_State=Collect

::(Collector_State==Collect)
     -> if
        :: atomic{ Collector_Sensor!Sensor(Activate_sensor)
       -> Collector_State=Activation}
         :: atomic{ Collector_Center!Center(Information_Data)
          ->Collector_State=Transfer_Center}
                                  fi

::(Collector_State==Transfer_Center)
     -> Collector_State=Final_State
::(Collector_State==Final_State)-> break
od
}

init {
        run Collector_Act();
        run Sensor_Act();
        run Patient_Act();
        run Center_Act();
        run Doctor_Act();
}
```

Listing 6.2 : Le processus data collecteur et du processus Init.

6.1 Propriétés comportementales du système WBSN

Les propriétés comportementales (spécifications) sont les formules logiques temporelles que nous définissons pour décrire les comportements souhaités que le système WBSN doit avoir. Un ensemble de spécifications est décrit pour spécifier un comportement cohérent du système. Il est fourni au vérificateur de modèles SPIN pour vérifier que le système a la liste des comportements émergents décrits par l'ensemble de spécifications. Pour le système WBSN dédié à la surveillance médicale, nous avons spécifié les propriétés comportementales (propriétés de correction) décrites ci-dessous :

- Le passage du collecteur de données active les capteurs pour réaliser les mesures des paramètres physiologiques et environnementaux.
- Une fois le capteur est activé, il doit effectuer la mesure d'un paramètre.
- Le collecteur de données doit collecter les informations avant de les transférer au centre médical.
- Si le capteur est activé, les données mesurées seront transmises au collecteur de données.
- Si le capteur est activé, les données mesurées seront transmises après un temps fini au centre médical.
- Le médecin ne peut fournir de traitement que si les données mesurées ont été transférées au centre médical.
- Le patient ne peut recevoir de traitement que si le médecin a consulté le centre médical
- Les capteurs doivent être déjà alertés avant de mesurer un signe physiologique ou environnemental.
- Les données mesurées doivent être transférées au centre médical avant que les médecins ne prennent une décision.
- Les données mesurées doivent être collectées avant leur transfert éventuel vers le centre médical.
- A la fin de l'exécution du système, les canaux de communication doivent être vides.

Certaines de ces propriétés exprimées dans des formules en logique temporelle linéaire sont répertoriées dans le listing 6.3.

```
#define Collector_Start (Collector_State==Initial_State)
#define Sensor_Measure(Sensor_State==Sensoring)
#define Sensor_Activation (Sensor_State==Activation)
#define Patient_Measure (Patient_State==Sensoring)
#define Collector_Collection(Collector_State==Collection)
#define Center_Transfer (Center_State==Transfer_Center)
```

```
#define Doctor_Treatment (Doctor_State==Decision)
#define Patient_Treated (Patient_State==Final_State)

ltl prop1 {[](Collector_Start -> <> Sensor_Measure)}

ltl prop2 {[](Sensor_Activation -> <> Patient_Measure)}

ltl prop3 {[](Collector_Collection -> <> Center_Transfer)}

ltl prop4_1 {(Sensor_Measure -> <> Collector_Collect)}

ltl prop4_2 {(Sensor_Measure -> <> Center_Transfer)}

ltl prop5 {(Doctor_Treatment -> <> Center_Transfer)}

ltl prop6 {[](lDoctor_Treatment -> <> Patient_Treated)}

ltl prop7 { Term_State -> Channels_Empty}

ltl prop8 { Sensor_Measure -> (Sensor_Alert == Alerted)}

ltl prop9 { Patient_Measure -> (Patient_Alert == Alerted)}

ltl prop10 {Doctor_Decision -> (Data_Rec==Recorded)}
```

Listing 6.3 : Les propriétés comportementales du système WBSN.

6.2 Vérification de modèles du système WBSN par SPIN

Le programme Promela décrivant le système WBSN est généré automatiquement par le processus de transformation modèle-texte. Il s'agit d'un ensemble de processus où chaque processus spécifie le comportement d'un composant du système. SPIN effectue des simulations aléatoires de l'exécution du système. La simulation de l'exécution du système WBSN dans un environnement concurrent par SPIN n'a détecté aucune erreur. Tous les processus terminent leur exécution sans erreurs, sans interblocages et sans cycles indéfinis. Les résultats de la simulation sont présentés dans la figure (Fig. 6.8). La figure (Fig. 6.9) montre un scénario de la simulation aléatoire du système WBSN exprimé par un diagramme de séquence. SPIN n'a également trouvé aucune erreur lors de la vérification des composants du système sans les propriétés de correction (safety verification). Les résultats de cette vérification sont montrés dans la figure (Fig. 6.10).

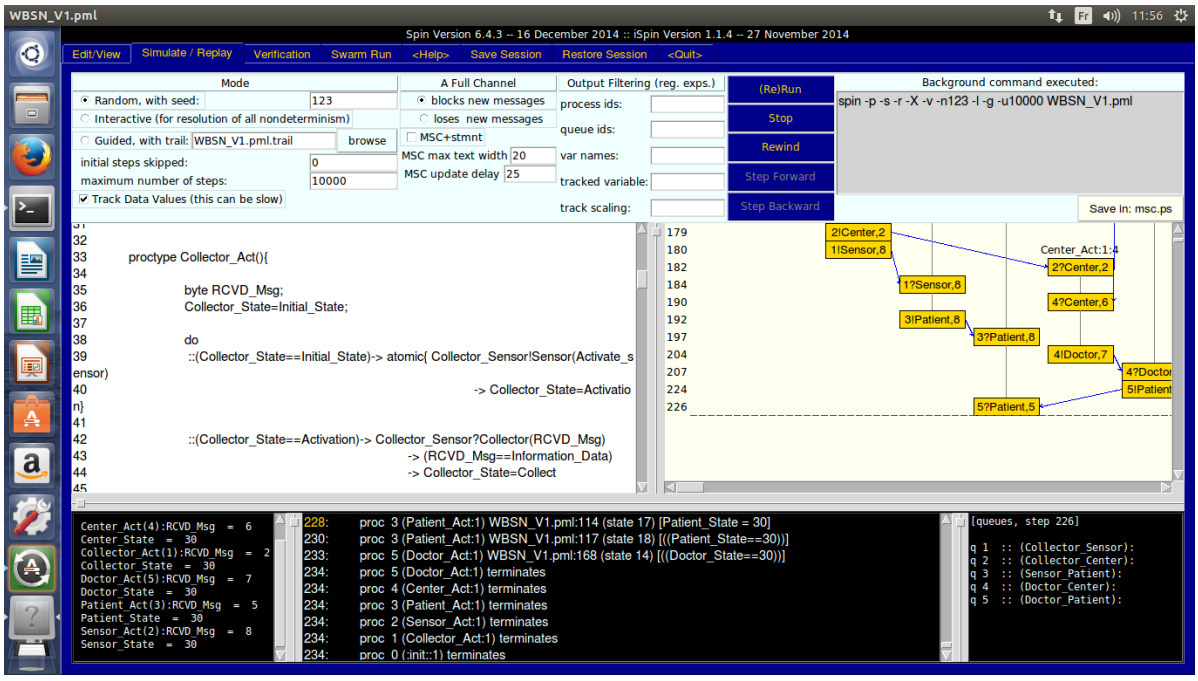

Fig. 6.8 : Les résultats de la simulation aléatoire.

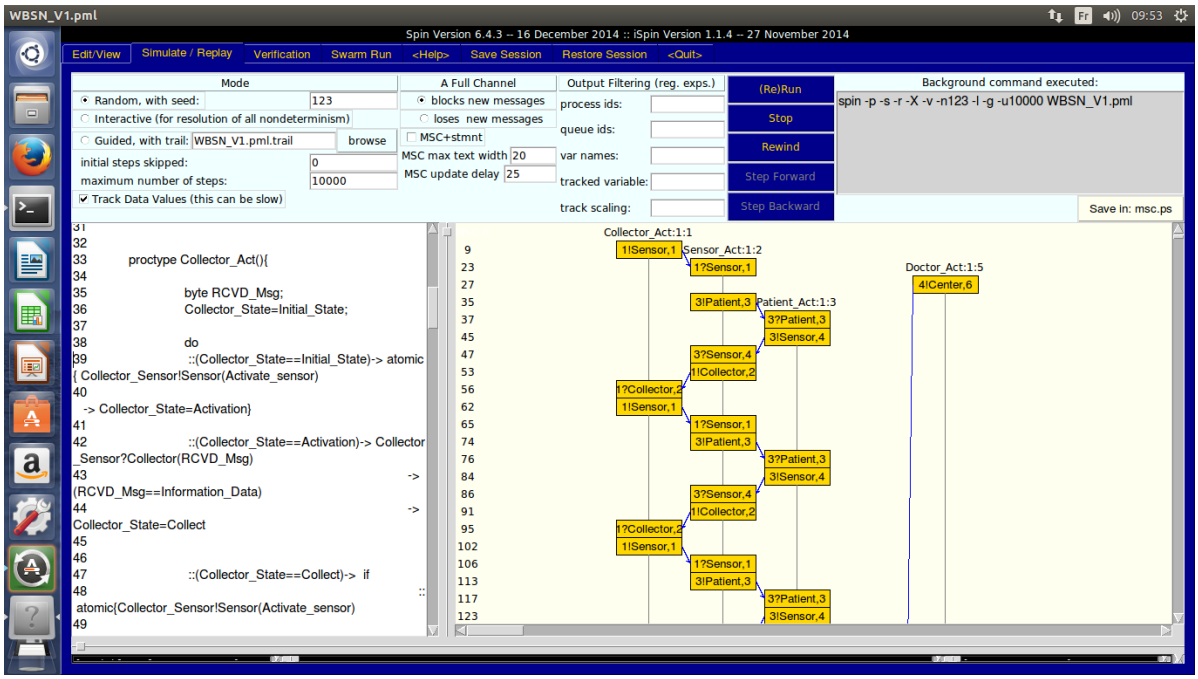

Fig. 6.9 : Un scénario de la simulation aléatoire du système WBSN.

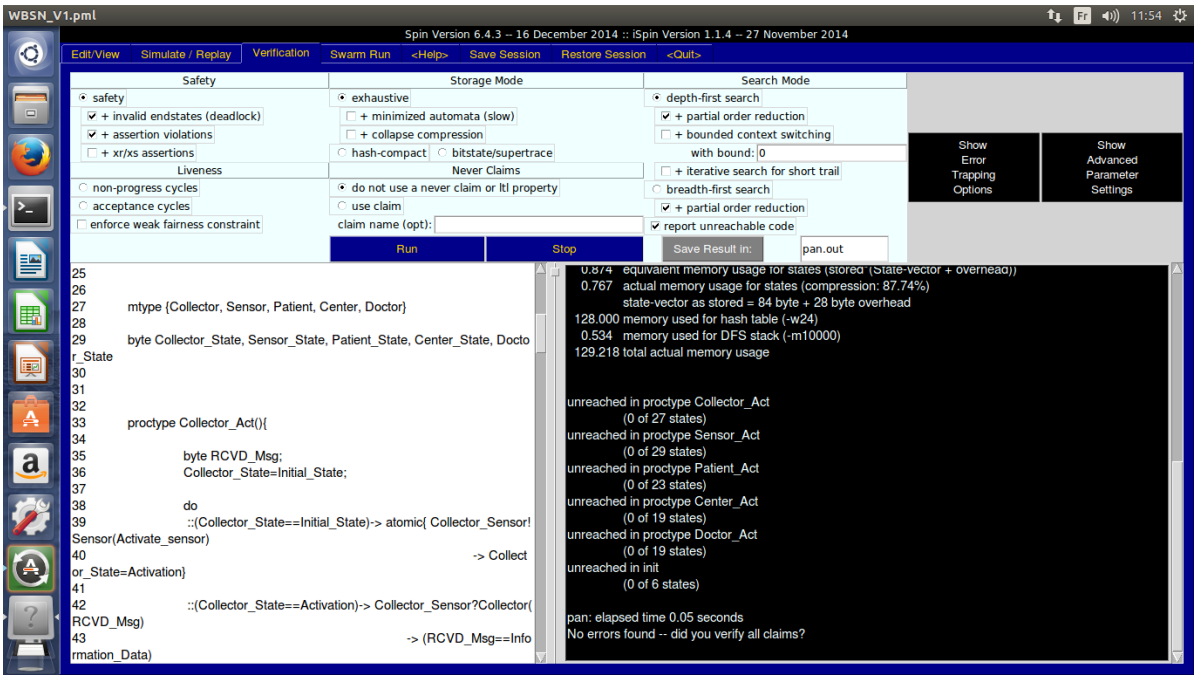

Fig. 6.10 : Les résultats de la vérification sans propriétés de correction.

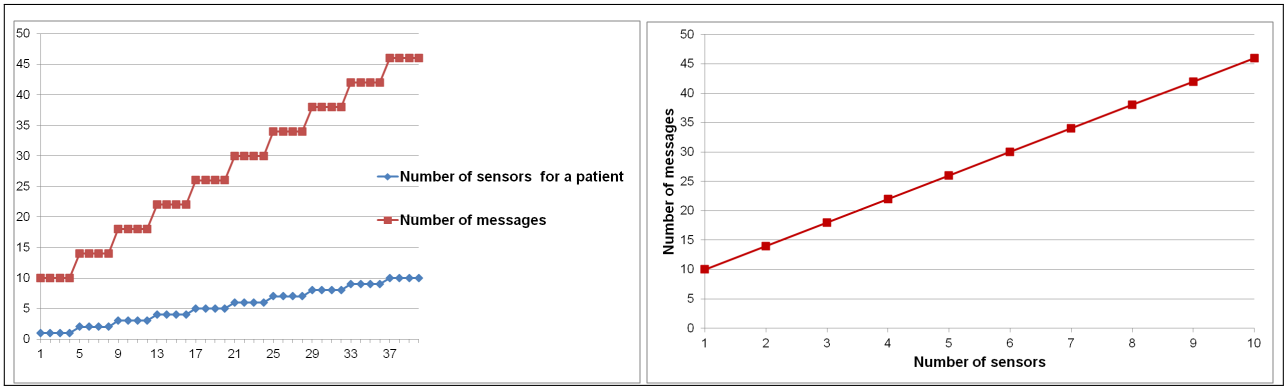

Fig. 6.11 : Les résultats de la simulation.

Nous avons réalisé plusieurs simulations aléatoires du système WBSN en faisant varier à chaque fois le nombre de capteurs attachés à un patient. Les résultats de ces simulations sont présentés dans le tableau 6.1. Ces résultats montrent que quel que soit le nombre de capteurs attribués à un patient, la simulation se termine sans erreurs dans un temps fini. Les résultats ont montré que l'exécution des comportements des composants du WBSN dans un environnement distribué ne génère aucun message inattendu et que tous les messages transmis sont reçus. Ces résultats sont

représentés par les courbes de la figure (Fig.6.11). De plus, les résultats montrent que la variation du nombre de messages en fonction du nombre de capteurs est stable. Le nombre de messages ne change pas pendant les différentes exécutions pour un nombre donné de capteurs. Il augmente avec l'augmentation du nombre de capteurs de manière constante (monotone).

Simulation	**Number of Sensors**	**Number of messages**	**Simulation**	**Number of Sensors**	**Number of messages**	**Simulation**	**Number of Sensors**	**Number of messages**	**Simulation**	**Number of Sensors**	**Number of messages**	**Simulation**	**Number of Sensors**	**Number of messages**
1	1	10	9	3	18	17	5	26	25	7	34	33	9	42
2	1	10	10	3	18	18	5	26	26	7	34	34	9	42
3	1	10	11	3	18	19	5	26	27	7	34	35	9	42
4	1	10	12	3	18	20	5	26	28	7	34	36	9	42
5	2	14	13	4	22	21	6	30	29	8	38	37	10	46
6	2	14	14	4	22	22	6	30	30	8	38	38	10	46
7	2	14	15	4	22	23	6	30	31	8	38	39	10	46
8	2	14	16	4	22	24	6	30	32	8	38	40	10	46

Tab. 6.1 : Résultats de la simulation.

La table 6.2 montre la variation du temps de la simulation en fonction du nombre de capteurs assignés à un patient. Ces résultats montrent que le temps de simulation est fini et stable pour un nombre donné de capteurs pendant toutes les exécutions (Fig.6.12).

Sensors	time (s)	Sensors	time (s)	Sensors	time (s)	Sensors	time (s)	Sensors	time (s)
1	1,98	3	2,94	5	3,91	7	4,97	9	5,93
1	1,95	3	3,03	5	3,88	7	5,01	9	5,88
1	1,89	3	3,05	5	3,9	7	4,99	9	5,86
1	1,90	3	2,99	5	3,85	7	4,93	9	5,91
2	2,46	4	3,68	6	4,45	8	5,42	10	6,38
2	2,41	4	3,73	6	4,42	8	5,38	10	6,42
2	2,33	4	3,70	6	4,48	8	5,44	10	6,39
2	2,38	4	3,63	6	4,41	8	5,4	10	6,44

Tab. 6.2 : Variation du temps de simulation.

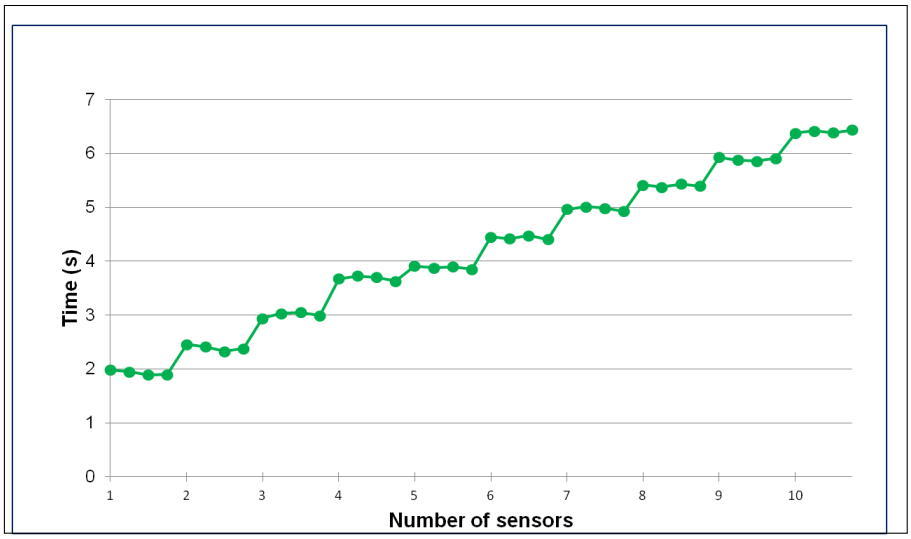

Fig. 6.12 : Variation du temps de simulation.

Au cours de la vérification exhaustive des propriétés comportementales, SPIN vérifie si ces propriétés d'exactitude exprimées dans les formules en logique temporelle linéaire (LTL) sont valides pour le modèle du système. Pour le système WBSN, nous avons combiné le programme Promela issu du processus M2T avec les propriétés de correction exprimées dans LTL. Ensuite, nous avons effectué la vérification des propriétés d'exactitude décrites au paragraphe 6.1. Le vérificateur de modèles SPIN montre que les propriétés spécifiées sont valides pour les comportements dérivés du système. Les résultats de la vérification sont illustrés par la figure (Fig. 6.13).

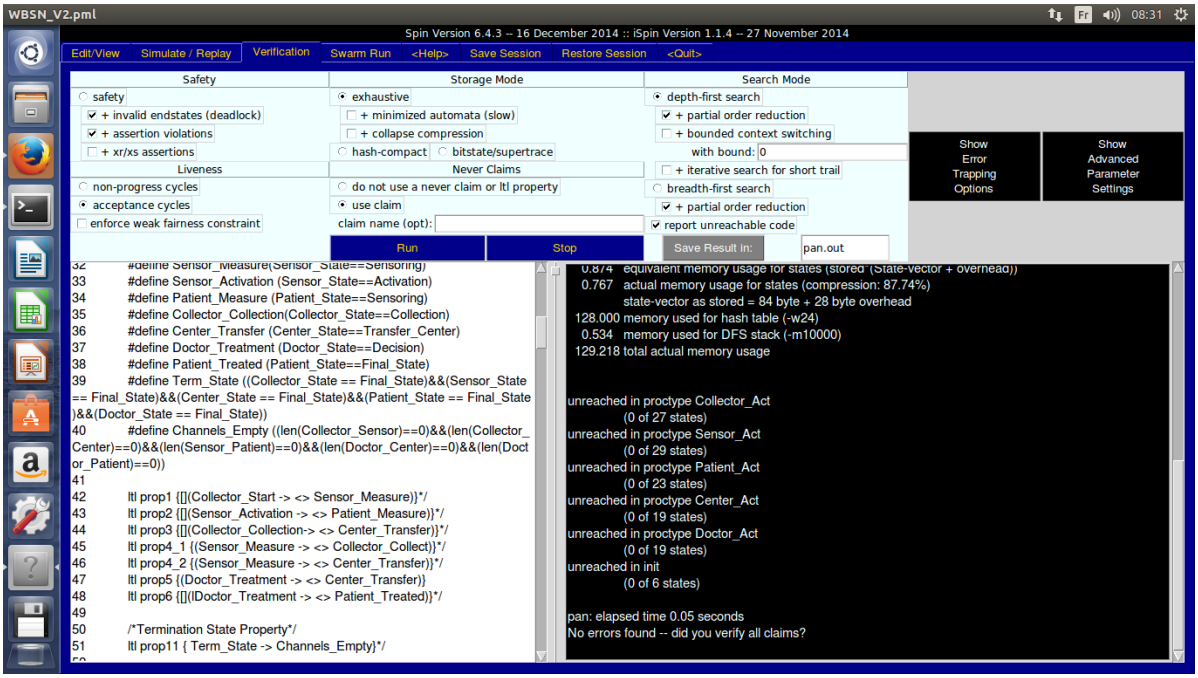

Fig. 6.13 : Les résultats de la vérification des propriétés comportementales.

7 Conclusion

Dans ce chapitre, nous avons présenté l'architecture d'un réseau de capteurs WBSN pour la surveillance médicale. Cette architecture a été conçue pour assurer le suivi des fonctions vitales dans différents scénarios de fonctionnement. L'architecture WBSN est basée sur l'utilisation de nœuds capteurs, pour mesurer les paramètres vitaux du corps humain, en combinaison avec un collecteur de données mobile, pour collecter les données mesurées et les transmettre au centre médical.

Nous avons appliqué l'approche proposée dirigée par les modèles pour dériver le comportement approprié de chaque nœud dans le système WBSN à partir du comportement global du système. Les messages de coordination sont inclus dans les comportements dérivés du système pour assurer la synchronisation globale et la coordination entre les différents composants du WBSN. Ce processus de dérivation peut jouer un rôle important dans le développement d'applications WBSN en augmentant le niveau d'abstraction et en augmentant la réutilisation du comportement dérivé sur plusieurs plateformes. Il est également possible de traiter les différents scénarios sans avoir besoin d'une nouvelle conception du système.

Nous avons utilisé l'approche proposée dans le chapitre 5 pour vérifier et valider les comportements dérivés du WBSN. La vérification et la validation des comportements dérivés permettent de s'assurer que la collaboration de ces comportements répond aux exigences initiales. Ce processus de vérification dirigé par les modèles est principalement basé sur des transformations automatiques des modèles dérivés du WBSN en modèles spécifiques requis par le vérificateur de modèles. Un programme Promela est ensuite obtenu à partir de ces modèles spécifiques au moyen d'une transformation modèle-texte. Le programme Promela généré est combiné avec les propriétés de correction pour effectuer une analyse et une vérification du système.

La simulation aléatoire des comportements dérivés du système WBSN effectuée par le vérificateur de modèles SPIN nous a permis d'explorer plusieurs scénarios d'exécution du système. Les résultats ont montré que les exécutions de ces comportements dans un environnement distribué ne génèrent aucun message inattendu et que tous les messages transmis sont reçus. L'exécution ne mène à aucun blocage, elle se termine toujours de manière sûre et ne génère pas d'erreurs d'exécution. La vérification des propriétés comportementales par SPIN a montré l'absence d'erreurs et que les comportements dérivés satisfont la spécification globale WBSN. Toutes les propriétés de correction exprimées sur le comportement global de WBSN sont valides pour les comportements dérivés. Cette vérification nous a permis de valider le processus de dérivation du WBSN. En conclusion, on peut dire que le processus de dérivation et sa vérification constituent une approche robuste et fiable pour le développement et la maintenance des systèmes WBSN.

Chapitre 7

Conclusions et Perspectives

Au cours de cette thèse, nous nous sommes intéressés à l'ingénierie des modèles et plus précisément à la transformation des modèles des exigences en modèles conceptuels. Le point de départ a été de présenter un état de l'art relatif à l'ingénierie des modèles (IDM) et les concepts manipulés par ce paradigme de l'ingénierie logicielle. Nous avons présenté l'approche MDA et ses trois niveaux d'abstraction et les techniques de transformations adoptées dans les travaux de recherche en IDM (MDE) et MDA. Par la suite, nous avons cité quelques travaux ayant utilisés l'approche IDM pour le développement des systèmes informatiques. Mais ces travaux se sont concentrés plus sur les transformations de bas niveaux dans la hiérarchie de conception d'une application à savoir le passage du niveau conceptuel vers le niveau de déploiement et plus précisément des modèles PIM en modèles PSM et des modèles PSM vers du code.

Cependant, peu de travaux se sont intéressés aux transformations de haut niveau d'abstraction en IDM, à savoir la transformation des modèles des exigences en modèles conceptuels d'une part. D'autre part, les systèmes informatiques deviennent de plus en plus complexes. Le comportement global de ces systèmes n'est pas réalisé par un seul composant mais par la collaboration d'un ensemble de composants ou de rôles. Un tel comportement peut être décomposé en un ensemble de comportements partiels réalisés par les différents composants du système. Le processus de dérivation du comportement des composants ou entités du système est très important dans le développement des systèmes distribués et les applications distribuées.

La transformation du modèle des exigences en modèles conceptuels et la dérivation du comportement des composants ou rôles d'un système sont très utiles dans la conception des systèmes informatiques. Ainsi, il est nécessaire de définir une approche de transformation pour dériver automatiquement le comportement des rôles du système à partir de la spécification globale du comportement du système, modélisé par les modèles des exigences. Un tel processus de dérivation automatique peut conduire à des systèmes valides et robustes en évitant les erreurs introduites par les activités de conception manuelle.

1 Contributions

Les travaux présentés dans cette thèse ont pour objectif de contribuer à apporter une solution dans les domaines de la transformation des modèles des exigences et la dérivation du comportement d'un système et plus particulièrement un système distribué. Les systèmes distribués ont leurs caractéristiques spécifiques, telles que l'échange de messages et les collaborations entre les composants distribués, qui doivent être prises en compte lors du processus de dérivation. Nous nous sommes intéressés aux phases initiales du développement d'un système distribué, en particulier dans le processus d'obtention de l'aspect conceptuel d'un système à partir de ses exigences globales. C'est un processus très important dans le développement de systèmes complexes et leur dérivation automatique.

Pour ce faire, nous avons proposé une approche qui traite l'automatisation du processus de transformation des méta-modèles et dérive le comportement des composants d'un système donné à partir des exigences globales du système. Les exigences définissent le comportement global du système. Elles sont modélisées en utilisant un diagramme d'activités UML étendu, tandis que le comportement des composants est dérivé sous la forme de machines à états-transitions distribuées. L'approche proposée est basée sur la définition du méta-modèle source approprié (méta-modèle des exigences), la définition du méta-modèle cible relatif au niveau conceptuel et la définition des règles qui régissent la transformation au cours du processus de dérivation.

Le processus de dérivation proposé est basé sur la transformation par méta-modèles. Ce processus est régi par un ensemble de règles de transformation. Il transforme le comportement global

du système décrit par un diagramme d'activités UML, étendu avec les collaborations, en comportements de rôles du système représentés par des machines à états-transitions UML. Les éléments de base du comportement global du système sont des collaborations entre les composants ou les rôles du système. Nous avons utilisé les collaborations parce qu'elles sont très appropriées pour modéliser les exigences. Elles fournissent un cadre structurel qui peut incarner à la fois les comportements des rôles et les interactions entre ces rôles nécessaires pour accomplir un service. Ce processus intègre les messages de coordination appropriés entre les comportements de rôles dérivés et nécessaires pour le fonctionnement cohérent du système. Une étude de cas d'application de télédiagnostic en neurosciences a été utilisée pour montrer le fonctionnement de l'approche de transformation et de dérivation du comportement des rôles du système.

Une mise en œuvre de l'approche de dérivation proposée a été réalisée sur la plateforme Eclipse avec le langage de transformation de modèles ATL (Atlas Transformation Language). La mise en œuvre pour chaque niveau de transformation, à savoir du modèle des exigences vers le modèle conceptuel ou du modèle conceptuel vers le modèle dépendant d'une plateforme exige :

(i) la représentation du méta-modèle source, du méta-modèle cible et du modèle source instance du méta-modèle source ;

(ii) l'application des règles de transformation de modèles spécifiés dans le langage ATL sur le modèle source. Cette phase génère les modèles résultats par exemple, une machine à états-transitions décrivant le comportement de chaque rôle du système. Le modèle obtenu est conforme au méta-modèle cible.

L'approche de dérivation de comportement présentée se focalise sur les premières phases de développement d'un système distribué, en particulier l'obtention d'une conception d'un système à partir de sa spécification des exigences globales. Les comportements dérivés du système s'exécutent dans un environnement distribué et collaborent pour atteindre le comportement global du système. Les aspects de concurrence rendent ces applications difficiles à développer et à vérifier. Ainsi, le défi le plus important consiste à trouver les moyens et les outils nécessaires afin de de garantir que ces comportements dérivés fonctionnent sans erreurs. La vérification de modèles est l'une des méthodes formelles les plus puissantes utilisées pour la vérification des systèmes concurrents et distribués. Dans cette perspective, nous avons proposé une approche dirigée par les modèles

pour la vérification des comportements dérivés du système. Cette approche de vérification consiste à vérifier si le système dérivé en question se comporte tel qu'il a été conçu pour se comporter.

La première étape de la vérification de modèles consiste à décrire un modèle du système à vérifier. Le défi consiste à écrire un modèle qui ne décrit que les caractéristiques pertinentes du système entier, et qu'il soit suffisamment détaillé pour représenter fidèlement le système. Dans l'approche de dérivation proposée, ces modèles sont automatiquement dérivés du comportement global du système. Il n'y a aucun temps à consacrer à la conception de ces modèles et à s'assurer qu'ils décrivent suffisamment le système. Cette décomposition automatique réduit la complexité du système pour la vérification. Elle génère un ensemble de modèles décrivant plusieurs comportements concurrents dans le système qui ont exactement le même comportement que celui du système en entier, donc seule une telle séquence de modèles doit être vérifiée.

L'approche de vérification dirigée par les modèles est principalement basée sur une transformation automatique des composants du système (modèles de conception) en modèles spécifiques à la vérification. Un ensemble de règles est requis pour réaliser cette transformation de modèles. Un programme Promela est généré à partir de ces modèles spécifiques. Le résultat est combiné avec les propriétés comportementales pour réaliser l'analyse et la vérification du système.

Le travail présenté dans cette thèse contribue également à la conception et la validation des applications de surveillance médicale basés sur les WBSNs. Nous avons proposé une architecture composée de nœuds hétérogènes pour construire un système de surveillance médicale préventive. Le système de surveillance médicale proposé est conçu pour effectuer une surveillance quotidienne et continue ainsi que des contrôles spécifiques pour les patients dans un hôpital. Nous avons défini un modèle pour décrire le comportement global du système WBSN. L'approche d'ingénierie dirigée par les modèles pour la dérivation a été adoptée pour dériver le comportement des composants du système WBSN (capteurs, collecteurs de données, etc.) à partir des exigences globales du WBSN. Elle permet d'augmenter la dynamique du système par la dérivation automatique du comportement pour les différents scénarios de fonctionnement de l'hôpital. Dans un deuxième temps, nous avons utilisé l'approche de vérification proposée pour valider et vérifier les résultats du processus de dérivation. Cette approche de vérification est nécessaire pour vérifier si le système obtenu après la transformation se comporte correctement selon sa spécification globale afin d'augmenter les performances du système.

2 Perspectives et Extensions

Plusieurs perspectives ont été identifiées nous permettant d'envisager une suite pour nos travaux. Nous énumérons quelques-unes dans les points suivants :

1. Nous proposons de compléter le processus de dérivation par le traitement des conditions au niveau des structures de choix et de répétition. Cette extension nécessitera l'introduction de nouveau messages de coordination nécessaires pour le bon fonctionnement du système. Le traitement des interruptions est très important dans le fonctionnement d'un système. Par conséquent, il est intéressant d'inclure ce traitement au niveau de notre approche de transformation pour qu'elle puisse traiter un modèle des exigences complet décrivant le comportement d'un système.

2. Dans le contexte de l'approche proposée pour la transformation du modèle des exigences en modèles conceptuels, nous nous sommes intéressés à la transformation du comportement global du système. Ce comportement est modélisé par un diagramme d'activités étendu par les collaborations. Nous proposons d'étendre l'approche de transformation au comportement interne des collaborations, notamment les interactions entres les rôles participants dans une collaboration. Cette extension consiste à développer le processus de dérivation du comportement détaillé d'une sous-collaboration. Cette dérivation sera réalisée par la transformation du diagramme de séquences décrivant les interactions entre les rôles d'une sous-collaboration. La transformation du comportement interne des collaborations permettra l'obtention d'automates d'états-transitions plus complets décrivant le comportement local des différents rôles.

3. La génération automatique de code vers différentes plateformes est très importante. Elle contribuera à renforcer l'approche et permettra de concevoir des systèmes plus robustes durant toutes les phases du processus de développement depuis l'élaboration des besoins jusqu'à le déploiement sur une plateforme d'exécution.

4. La définition d'outils graphiques pour la représentation des modèles sources permettra de faciliter la tâche à l'utilisateur pour représenter ses modèles d'une part. D'autre part, ces outils graphiques vont permettre la conversion des modèles issus des transformations sous forme XMI en modèles graphiques plus lisibles et simple à interpréter par l'utilisateur.

5. La complexité croissante des systèmes distribués ainsi que le changement des conditions d'exploitation exigent de prendre en compte ces changements pour le système déjà existant. Une solution possible que nous envisageons est l'utilisation de mécanismes pour effectuer des adaptations comportementales ou des changements dans les systèmes en cours d'exécution. Cette solution est appelée adaptation dynamique. Généralement, les modifications sont introduites à un niveau élevé (niveau de spécification des exigences) avant d'être propagées aux composants du système existant. Dans les travaux futurs, nous prévoyons d'étendre ce travail en incluant l'adaptation dynamique du comportement dérivé aux différents changements de fonctionnement que le système peut avoir. Nous proposons d'utiliser une approche dirigée par les modèles pour traiter les changements dynamiques dans les spécifications du système distribué et pour propager les modifications aux différents composants existants du système. Cette approche sera constituée d'un processus de détection de changement, de modification du comportement et de dérivation automatique des comportements des composants correspondants. Elle concerne principalement la capacité des systèmes logiciels à s'adapter aux changements dans leurs spécifications globales. Cette nouvelle approche consiste en une architecture de processus, un ensemble de composants pour assurer la conformité structurelle et comportementale des comportements dérivés du système après les changements, et un algorithme d'adaptation pour permettre l'adaptation dynamique. Elle traite de l'évolution dynamique d'une spécification globale d'un système et de la dérivation automatique du comportement des composants d'un système pendant son cycle de vie.

6. Notre architecture WBSN consiste en un ensemble de patients placés au hasard dans les services hospitaliers, et un ensemble d'infirmiers équipées de collecteurs de données pour collecter périodiquement les données mesurées. Le déploiement de ces collecteurs de données dans l'hôpital et leur parcours peuvent être définis de manière statique par l'administrateur. Cette stratégie peut générer des temps de collecte assez importants. Par conséquent, nous suggérons l'utilisation d'une solution formelle et distribuée pour réaliser un déploiement dynamique des collecteurs de données afin de recueillir les données mesurées. Cette solution peut assurer une charge de travail équitable entre tous les collecteurs de données. Elle vise également à identifier le parcours minimal pour chaque collecteur dans le système WBSN. Nous utiliserons une méthode formelle de jeu de coalitions afin d'établir des coalitions de patients et d'infirmiers en fonction de leurs emplacements géographiques dans l'hôpital. Le

caractère coopératif de cette solution permettra d'atteindre les performances globales du système en termes de couverture totale des patients et d'équité des soins entre les infirmiers. Cette approche de calcul de parcours distribuée sera ensuite incorporée dans le comportement du collecteur de données.

7. Dans l'architecture proposée du système WBSN, les patients sont suivi par un collecteur de données qui accumule et traite les données d'un ensemble de capteurs médicaux et environnementaux. Ces données mesurées sont relayées ensuite au centre médical. Cette architecture est conçue pour faciliter un diagnostic accéléré des maladies et pour fournir une efficacité et une précision accrues dans le processus de suivi des patients. Nous projetons d'étendre cette architecture afin de créer un aspect de sécurité dans le suivi des patients à tout moment en fournissant une attention particulière aux cas d'urgence. Le système surveillera les alarmes déployées en cas d'anomalies observées dans le système. L'existence de plusieurs anomalies permet aux personnels soignants de travailler de manière coopérative. Cette approche réduit la charge de travail qui est assignée à chacun d'eux et minimise ainsi la durée de leur intervention. Dans cet objectif, nous voulons modéliser et traiter la coopération entre les différents acteurs du système WBSN pour la gestion et le suivi des alarmes pour répondre aux cas d'urgence. Pour atteindre cet objectif, nous devrions considérer les contraintes de temps, d'espace et de disponibilité des acteurs lors d'une intervention. Nous prévoyons de minimiser le temps nécessaire aux médecins, aux infirmiers et aux techniciens pour faire face à ces situations d'urgence. Le problème consiste à déterminer la meilleure stratégie pour que les participants coopèrent les uns avec les autres en temps réel pour remédier aux anomalies constatées. Le problème peut être considéré comme un problème de théorie des jeux avec des acteurs coopératifs : les médecins, les infirmiers et les techniciens.

8. Nous prévoyons également de vérifier l'évolutivité de cette approche en testant plusieurs systèmes distribués. Cette vérification devra nous permettre de rendre cette approche dirigée par les modèles pour la dérivation et la vérification plus efficace, en termes de modèles sur lesquels elle peut raisonner, de temps et de l'espace qu'elle requiert.

Bibliographie

[1] Akbal-Delibas, B., Boonma, P. and Suzuki, J. : *Extensible and Precise Modeling for Wireless Sensor Networks*, Information Systems : Modeling, Development, and Integration, Lecture Notes in Business Information Processing, Springer Berlin Heidelberg, Vol. 20, pp. 551–562, (2009).

[2] Allilaire, F., Bézivin, J., Jouault, F., Kurtev, I. : *ATL : A model transformation tool*. Science of computer programming, Vol. 72 (1-2), pp. 31-39, (2008).

[3] Ameedeen, M., Bordbarand, B., Anane, R. : *Model interoperability via Model Driven Development*. Journal of Computer and System Sciences, Publisher : Elsevier Inc., Vol. 77, Issue 2, pp. 332-347, (2011).

[4] Amor, M., Fuentes, R., Vallecillo, A. : *Bridging the gap between agent-oriented design and implementation using mda*. Agent Oriented Software Engeneering Workshop, AAMAS'04, pp. 93-108, (2004).

[5] Anliker, U., Beutel, J., Dyer, M., Lukowicz, P. and G. Troster : *A Systematic approach to the distributed wearable systems*, The Computer Journal, IEEE Transactions, Vol. 53, No. 8, pp. 1017–1033, (2004).

[6] Appukuttan, B., Clark, T., Reddy, S., Tratt, L., Venkatesh, R. : *A Pattern based model driven approach to model transformations*. Proceeding Metamodelling for MDA 2003 : First International Workshop, (2003).

[7] Appukuttan, B., Clark, T., Reddy, S., Tratt, L., Venkatesh, R. : *A model driven approach to building implementable model transformations*. Workshop in Software Model Engineering (WiSME) 2003, (2003).

[8] Atkinson, C., Kuhne, T. : *Model-Driven Development : A Metamodeling Foundation*. IEEE Software, Vol. 20(5), pp. 36-41, (2003).

[9] The Atlas Transformation Language (ATL) project, at http ://www.eclipse.org/m2m/atl/.

[10] Beatriz, A., Silva, V., Lucena, C. : *Developing Multi-Agent Systems Based on MDA*. Proceeding of the CaiSE 05 Forum, (2005).

[11] Bertolini, D., Delpero, L., Mylopoulos, J., Nivikau, A., Orler, A., Penserini, L., Perini, A., Susi, A., Tomasi, B. : *A Tropos Model Driven Development Environment*. CAiSE 2006 Forum proceedings, (2006).

[12] Bézivin, J. : *On the unification power of models*. Software and System Modeling (SoSym), Vol. 4(2), pp. 171-188, (2005).

[13] Bézivin, J., Gerbé, O. : *Towards a Precise Definition of the OMG/MDA Framework*. Proceedings of the 16th IEEE international conference on Automated Software Engineering (ASE), (2001).

[14] Bézivin, J. : *In search of a Basic Principle for Model Driven Engineering*. Novatica/Upgrade, Vol. 2, pp. 21-24, at http ://www.upgrade-cepis.org/issues/2004/2/upgrade-vol-V-2.html, (2004).

[15] Bézivin, J. : *Les nouveaux défis des systèmes complexes et la réponse MDA de l'OMG*. Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents, at http ://www.lifl.fr/jfiadsma2002/talks/jfiadsma2002-Bezivin.pdf, (2002).

[16] Bochmann, G.V. : *Deriving component designs from global requirements*. Proceeding International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES), (2008).

[17] Booch, G., Rumbuagh, J., Jacobson, I. : *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, MA, (1999).

[18] Brandao, A., Silva, V., Lucena, C. : *A model driven approach to develop multi-agent systems*. Pontifcia Universidade Catlica do Rio de Janeiro - PUC-Rio, at http ://www.dbd.puc-rio.br/depto_informatica/05_09_brandao.pdf, (2005).

[19] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J. : *Tropos :An agent-oriented software development methodology*. Autonomous Agents and Multi-Agent Systems, pp. 203-236, (2004).

[20] Budinsky, F., Steinberg, D., Merks, E., Ellersick, R. and Grose T.J. : *Eclipse Modeling Framework*, Addison-Wesley Professional, (2003).

[21] Butler, R.W. and Finelli G.B. : *The infeasibility of quantifying the reliability of life-critical real-time software*, Journal IEEE TSE, Vol. 19, No. 01, pp. 3–12,(1993).

[22] Castejon, H., Bochmann, G.V., Brack R. : *Realizability of Collaboration-based Service Specifications*. Proceedings Asia-Pacific Software Engineering Conference (APSEC), (2007).

[23] Castejon, H., Bochmann, G.V., Brack R. : *Using Collaborations in the Development of Distributed Services*. Department of Telematics 2008, 37 s. AVANTEL Technical Report, (2008).

[24] Castejon, H., Bochmann, G.V., Brack R. : *On the Realizability of Collaborative Services*. Journal of Software and Systems Modeling, Springer., (2012).

[25] Clarke, E.M. and Emerson, E.A. : *Design and synthesis of synchronization skeletons using branching time temporal logic*, Proceedings of the Workshop on Logic of Programs, LNCS, Vol. 131, pp. 52–71, (1981).

[26] Clarke, E.M., Grumberg, O. and Long, D.E. : *Model checking and abstraction*, POPL, Proceedings of the 19^{th} ACM Symposium, pp. 343–354, (1992).

[27] Clarke, E.M., Grumberg, O. and Long, D.E. : *Verification tools for finite-state concurrent systems*, J.W. de Bakker, W.P. de Roever, G. Rozenberg (Eds.), Decade of Concurrency-Reflections and Perspectives (Proceedings of REX School), LNCS, Vol. 803, pp. 124–175, (1993).

[28] De Castro, V., Marcos, E., Vara J. : *Applying CIM-to-PIM model transformations for the service-oriented development of information systems*. Information and Software Technology, Vol. 53, pp. 87-105, at www.elsevier.com/locate/infsof, (2011).

[29] Edaoudi, R., Hamdani, M., Bouragba K., Ouzzif, M., Mountassir, H., Erradi, M. : *Architecture Formalisée d'une Application Collaborative Cas d'une Situation d'Urgence en Neurologie*. AFADL, (2010).

[30] Ehrig, H., Prange, U., Taentzer, G. : *Fundamental Theory for Typed Attributed Graph Transformation*. Graph Transformations, Second International Conference, ICGT 2004, Proceedings, LNCS, Vol. 3256, pp. 161-177, (2004).

[31] Favre, J.M. : *Towards a Basic Theory to Model Model Driven Engineering*. 3rd Workshop in Software Model Engineering, WiSME 2004, at http ://www-adele.imag.fr/ jmfavre, (2004).

[32] Garlan, D., Monroe, R., Wile, D. : *Acme : An Architecture Description Interchange Language*. Proceedings CASCON 97, (1997).

[33] Goubet, L. and Delaigue, L. : *Acceleo user guide*, at http ://help.eclipse.org/helios/topic/org.eclipse.acceleo.doc/doc/html/acceleo, (2008).

[34] Hairong Yan, Youzhi Xu, Gidlund, M. : *Experimental e-Health Applications in Wireless Sensor Networks*, WRI International Conference on Communications and Mobile Computing, Vol. 1, pp. 563–567, (2009) .

[35] Harbouche, A., Erradi, M. : *Une approche de dérivation du comportement : Cas de collaborations en Télémédecine*. WCARDN'2011- Second Workshop on Collaborative Applications for Remote Diagnosis in Neuroscience, (2011).

[36] Harbouche, A., Erradi, M., Mokhtari, A. : *A Transformation Approach for Collaboration Based Requirement Models*. International Journal of Software Engineering & Applications (IJSEA), Vol.3, No.1, (2012).

[37] Harbouche, A., Erradi, M., Mokhtari, A. : *An MDE Approach to Derive System component Behavior*. International Journal of Advanced Science and Technology (IJAST), Vol. 53, pp. 41–60, April, (2013).

[38] Harbouche, A., Djedi, N., Erradi, M., Ben-Othman, J., Kobbane, A. : *Model driven flexible design of a wireless body sensor network for health monitoring*. Computer Networks, The International Journal of Computer and Telecommunications Networking, Elsevier, Vol. 129 (2), pp. 548–571, (2017).

[39] Pei-Cheng Hii, Wan-Young Chung : *A Comprehensive Ubiquitous Healthcare Solution on an AndroidŹ Mobile Device*, Sensors, Vol. 11, pp. 6799–6815, (2011).

[40] Hoffer, J.A, George, J.F, Valacich, J.S. : *Modern system analysis and design*. Prentice Hall, (2004).

[41] Holzmann, G.J. : *The model checker SPIN*, Formal Methods in Software Practice, IEEE TSE, Vol. 23, No. 05, pp. 279–295, (1997).

[42] Holzmann G.J. : *The Spin Model Checker : Primer and Reference Manual*, Addison Wesley, Boston MA, (2004).

[43] Jacobson, I., Booch, G., Rumbaugh, J. : *The Unified Software Development Process*. Addison Wesley, pp. 463, (1999).

[44] Jain, S., Shah, R., Brunette, W., Borriello, G. and Roy, S. : *Exploiting mobility for energy efficient data collection in wireless sensor networks*, ACM/Springer Mobile Networks and Applications, Vol. 11, No. 3 , pp. 327–339, (2006).

[45] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I. and Valduriez, P. : *ATL : a QVT-like Transformation Language*. The 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, pp. 719-720, (2006).

[46] Jouault, F., Kurtev, I. : *Transforming Models with ATL*. J.-M. Bruel (Ed.) : MODELS 2005 Workshops, LNCS, Vol. 3844, pp. 128-138, (2006).

[47] Jouault, F., Kurtev, I. : *On the architectural alignment of ATL and QVT*. Proceedings of the 2006 ACM symposium on Applied computing, pp. 1188-1195, (2006).

[48] Jouault, F. : *Contribution à l'étude des langages de transformation de modèles*. Thèse de Doctorat, UFR Sciences & Techniques, Université de Nantes, France, (2006).

[49] Jouault, F., Allilaire, F., Bezivin, J. and Kurtev, I. : *ATL : a model transformation tool*, Journal Science of computer programming, Vol. 72, No ; 1-2, pp. 31–39, (2008).

[50] Judson, S. : *Pattern Based Model Transformation*. Proceedings OOPSLA '03 Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, (2003).

[51] Judson, S., Carver, D., France, R. : *A Metamodeling Approach to Model Transformation*. Proceedings OOPSLA '03 Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, (2003).

[52] Jun, H., Zhao, W., Ammar, M., Zegura, E. and Lee, C. : *Trading latency for energy in wireless ad hoc networks using message ferrying*, Proceedings of the 1st IEEE Workshop on Pervasive Wireless Networking (PWN 2005), pp. 220–225, (2005).

[53] Kaliappan, P. : *State of the Art, Model Driven Architecture*. at http ://www-rnks.informatik.tu-cottbus.de/content/unrestricted/staff/psk/MDA_Dec07.pdf, (2007).

[54] Khan, P., Hussain A. and Kwak, K.S. : *Medical Applications of Wireless Body Area Networks*, International Journal of Digital Content Technology and its Applications, Vol. 3, Nř. 3, pp.185–193, (2009).

[55] Karlos, M., Drosdova, M. : *Analytical Method of CIM to PIM Transformation in Model Driven Architecture (MDA)*. Journal of Information and Organizational Sciences, Vol. 34 Nř1, pp. 89-99, (2010).

[56] Kleppe, A., Warmer, J., Bast, W. : *MDA Explained - The Model Driven Architecture : Practice and Promise*. Addison-Wesley, (2003).

[57] Kruger, I., Mathew, R. : *Component synthesis from service specifications*. Proceedings Intl. Dagstuhl Workshop on Scenarios : Models, Transformations and Tools, LNCS, Vol. 3466, (2003).

[58] Kurtev, I., Bézivin, J., Aksit, M. : *Technological spaces : an initial appraisal*. CoopIS, DOA'2002 Federated Conferences, Industrial track, Irvine, (2002).

[59] Kurschl, W., Mitsch, S. and Schoenboeck, J. :*Modeling Distributed Signal Processing Applications*, Sixth International Workshop on Wearable and Implantable Body Sensor Networks, BSN 2009, pp. 103–108, (2009).

[60] Lamport, L. : *Proving the correctness of multiprocess programs*, IEEE TSE, Vol. 3, No. 02, pp. 125–143, (1997).

[61] Lawly, M., Steel, J. : *Practical Declarative Model Transformation with Tefkat*. Satellite Events at the MoDELS 2005 Conference : MoDELS 2005 International Workshop, LNCS., Vol. 3844, pp. 139-150, (2005).

[62] Lawrence, E., Navarro, K. F., Hoang, D. and Lim, Y. : *Data Collection, Correlation and Dissemination of Medical Sensor Information in a WSN*, Fifth International Conference on Networking and Services, pp. 402–408, (2009).

[63] Lee, Y. D. and Chung, W. Y. : *Wireless sensor network based wearable smart shirt for ubiquitous health and activity monitoring*, Sensors and Actuators B : Chemical, Vol. 140, pp. 390–395, (2009).

[64] Losilla, F., Vicente-Chicote, C., Alvarez, B., Iborra, A. and Sanchez, P. : *Wireless Sensor Network Application Development : An Architecture-Centric MDE Approach*, Proceedings of the ECSA 2007, Lecture Notes in Computer Science, vol. 4758, pp. 179–194, (2007).

[65] Lucena, M., Castro, J., Silva, C., Alencar, F. and Santos, E. : *STREAM : A Strategy for Transition between Requirements Models and Architectural Models*. SAC 2011, ACM 978-1-4503-0113-8/11/03, (2011).

[66] Lucena, M., Silva, C., Santos, E., Alencar, F. and Castro, J. : *Applying Transformation Rules to Improve i* Models*. 21st Intl. Conf. on Software Engineering & Knowledge Engineering (SEKE'09), pp. 43-48, (2009).

[67] Mansour, A., Leblond, I., Hamad, D. and Artigas, F. : *Sensor Networks for Underwater Ecosystem Monitoring and Port Surveillance Systems*, Sensor Networks for Sustainable Development, M. Illias (Ed.), pp. 1–25, (2013).

[68] Mukherjee, S., Dolui, K. and Datta, S. K. :*Patient health management system using e-health monitoring architecture*, Advance Computing Conference (IACC), 2014 IEEE International, pp. 400–405, (2014).

[69] Neves, P., Stachyra M. and J. Rodrigues : *Application of wireless sensor networks to health-care promotion*, Journal of Communications Software and Systems, Vol. 4, Nř. 3, pp.181–190, (2006).

[70] Object Management Group, Inc. : *Model Driven Architecture(MDA)*. Document number ormsc/2001-07-0, Architecture Board, ORMSC. at http ://www.omg.org/docs/omg/01-07-01.pdf, (2001).

[71] Object Management Group, Inc. : *MOF 2.0 Query/ Views/ Transformations RFP*. OMG Document ad/2002-04-10, 2002. at http ://www.omg.org/cgi-bin/doc ?ad/2002-04-10, (2002).

[72] Object Management Group, Inc. : *MDA Guide Version 1.01*. Document number omg/2003-06-01. at http ://www.omg.org, (2003).

[73] Object Management Group, Inc. : *QVT-Partners. First revised submission to QVT RFP*. OMG document ad/03-08-08. at http ://www.omg.org, (2003).

[74] Object Management Group, Inc. : *UML 2.0 Superstructure Specification*. OMG Adopted Specification. at http ://www.omg.org, (2004).

[75] Object Management Group, Inc. : *Unified Modeling Language : Superstructure version 2.0, formal/05-07-04*. at http ://www.omg.org/docs/formal/05-07-04.pdf, (2005).

[76] Object Management Group : *UML 2.0 Superstructure Specification*, OMG Adopted Specification, at http ://www.omg.org, (2007).

[77] Object Management Group, Inc. : *XML Metadata Interchange*. Specification document v2.1 . at http ://www.omg.org/docs/formal/05-09-01.pdf, (2005).

[78] Object Management Group, Inc. : *MOF Query/ View/ Transformations (QVT) Adopted specification*. at http ://www.omg.org, (2005).

[79] Object Management Group : *Meta-Object Facility (MOF) 2.0 Core Specification*, at http ://www.omg.org/docs/ptc/04-10-15.pdf, (2004).

[80] Object Management Group, Inc. : *Meta Object Facility (MOF) 2.0 Core Specification, OMG Document formal/2006-01-01*. at http ://www.omg.org/cgi-bin/doc ?formal/2006-01-01, (2006).

[81] Object Management Group, Inc. : *BPMN : Business Process Modeling Notation Specification, OMG Final Adopted Specification, dtc/06-02-01*. at http ://www.bpmn.org/Documents/2006-02-01.pdf, (2006).

[82] Object Management Group, Inc. : *MOF Query/ View/ Transformations (QVT) Adopted specification*. at http ://www.omg.org, (2008).

[83] Pahuja, R., Verma, H. K. and Uddin, M. :*A Wireless Sensor Network for Greenhouse Climate Control*, Pervasive Computing, IEEE, Vol. 12, No. 2, pp. 49– 58, (2013).

[84] Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F. : *Service-Oriented Computing*. Research Roadmap. at ftp ://ftp.cordis.europa.eu/pub/ist/ docs/directorate_d/st-ds/services-research-roadmap_en.pdf, (2006).

[85] Pavon, J., Gomez-Sanz, J., Fuentes R. : *Model Driven Development of Multi-Agent System*. Model Driven Architecture - Foundations and Applications, LNCS, Vol. 4066, pp. 284-298, (2006).

[86] Perini, A., Susi, A. : *Automating Model Transformations in Agent-Oriented modelling*. Proceedings of 6th International Workshop AOSE 2005, (2005).

[87] Pnueli, A. : *The temporal logic of programs*, FOCS, Proceedings of the 18^{th} IEEE Symposium, pp. 46–57, (1977).

[88] Queille, J.P. and Sifakis, J. : *Specification and verification of concurrent systems*, Cesar, Proceedings of the 5^{th} International Symposium on Programming, LNCS, Vol. 137, pp. 337–351, (1982).

[89] Rao, J. and Biswas, S. : *Joint routing and navigation protocols for data harvesting in sensor networks*, Proceedings of the 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS 2008), pp. 143–152, (2008).

[90] Raskovic, D.,Martin, T. and Javanov, E. : *Medical monitoring applications for wearable computing*, The Computer Journal, Vol. 47, No.4, pp. 495–504, (2004).

[91] Reyer, M., Hurlebaus, S., Mander, J. and Ozbulut, O. E. : *Design of a Wireless Sensor Network for Structural Health Monitoring of Bridges*, Wireless Sensor Networks and Ecological Monitoring, Smart Sensors, Measurement and Instrumentation, pp. 195–216, Vol. 3, (2013).

[92] Rodríguez, A., Fernández -Medina, E., Piattini, M. : *Towards CIM to PIM Transformation : From Secure Business Processes Defined in BPMN to Use-Cases*. BPM'07 Proceedings of the 5th international conference on Business process management, LNCS, Vol. 4714, pp. 408-415, (2007).

[93] Rodríguez, A., Fernández -Medina, E., Piattini, M. : *A BPMN Extension for the Modeling of Security Requirements in Business Processes*. IEICE Transactions on Information and Systems, Vol. E90-D(4), pp. 745-752, (2007).

[94] Rodríguez, A., Fernández -Medina, E., Piattini, M. : *Towards obtaining Analysis-level class and use case diagrams from Business ProcessModels*. Proceedings ER Workshops 2008, LNCS, Vol. 5232, pp. 103-112, (2008).

[95] Rougemaille, S., Migeon, F., Maurel, C., Gleizes, M. : *Model Driven Engineering for Designing Adaptive Multi-Agent Systems*. International Workshop on Engineering Societies in the Agents World (ESAW 2007), Springer, (2007).

[96] Rozier, K.Y. : *Survey : Linear Temporal Logic Symbolic Model Checking*, Journal Computer Science Review, Vol. 5, No. 02, pp. 163–203, (2011).

[97] Rumbaugh, J., Booch, G., Jacobson, I. : *The Unified Modeling Language UserGuide*. Object Technologies, ed. Addison-Wesley, (1999).

[98] Sanz, J., Pavon, J. : *Methodologies for Developing Multi-Agent Systems*. Journal of Universal Computer Science, Vol. 10, pp. 359-374, (2004).

[99] Seeger, C., Buchmann, A. and Van Laerhovenmy, K. : *HealthAssistant : A Phone-based Body Sensor Network that Captures the Wearer's Exercises throughout the Day* , Proceedings of the 6th International Conference on Body Area Networks (BodyNets), Beijing, China, ISBN 978-1-936968-29-9, ACM, (2011).

[100] Seidewitz, E. : *What models mean*. IEEE Software, Vol. 20(5), pp. 26-32, (2003).

[101] Shah, R. C., Roy, S., Jain, S. and Brunette, W. : *Data mules : Modeling a three-tier architecture for sparse sensor networks*, Proceedings of the 2nd ACM International Workshop on Wireless Sensor Networks and Applications (SNPA 2003), pp. 30–41, (2003).

[102] Silva, V., Garcia, A., Brandao, A., Chavez, C., Lucena, C., Alencar, P. : *Taming Agents and Objects in Software Engineering*. Software Engineering for Large-Scale Multi-Agent Systems, LNCS, (2003).

[103] Silva, V., Choren, R., Lucena, C. : *A UML Based Approach for Modeling and Implementing Multi-Agent Systems*. Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS'04, (2004).

[104] Silva, V., Choren, R., Lucena, C. : *Using the MAS-ML to Model a Multi-Agent System*. Software Engineering for Large-Scale Multi-Agent Systems II, LNCS, (2004).

[105] Silva, V., Cortes, M., Lucena, C. : *An Object-Oriented Framework for Implementing Agent Societies*. Technical Report MCC32/04, PUC-Rio, (2004).

[106] Silva, V., Lucena, C. : *From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language*. Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, pp. 145-189, (2004).

[107] Silva, V., Choren, R., Lucena, C. : *Using the UML 2.0 activity diagram to model agent plans and actions*. Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, AAMAS '05, (2005).

[108] Silva, V., Demaria, B., Lucena, C. : *An MDA-Based Approach for Developing Multi-Agent System*. Proceedings of 3rd Workshop on Software Evolution through Transformations (SeTra 2006) at the 3rd International Conference on Graph Transformation (ICGT 2006), pp. 115-126, (2006).

[109] Silva, V., Lucena, C. : *Modeling Multi-Agent Systems*. Communications of the ACM, (2006).

[110] Soley, R. : *Model Driven Architecture (MDA)*. Draft 3.2. Object Management Group Inc., (2000).

[111] Stevens, P. : *Bidirectional Model Transformations in QVT : Semantics Issues and Open Questions*. Model Driven Engineering Languages and Systems, 10th International Conference, MoDELS 2007, LNCS, Vol. 4735, (2007).

[112] Taentzer, G. : *AGG : A Graph Transformation Environment for Modeling and Validation of Software*. Applications of Graph Transformations with Industrial Relevance, Second International Workshop, AGTIVE 2003, LNCS, Vol. 3062, pp. 446-453, (2003).

[113] Taentzer, G., Ehrig, K., Guera, E., Lara, J., Lengyel, L., Levendovsky, T., Prange, U., Varro, D., Varro-Gyapay, S. : *Model Transformation by Graph Transformation : A Comparative Study*. Proceedings of the Model Transformations in Practice (MTiP) Workshop at MoDELS 2005, (2005).

[114] Tratt, L. : *The MT model transformation language*. Technical Report TR-05-02, Department of Computer Science, King's College London, (2005).

[115] Tratt, L. : *The Converge programming language*. Technical Report TR-05-01, Department of Computer Science, King's College London, (2005).

[116] Tratt, L. : *The MT model transformation language*. Proceedings ACM Symposium on Applied Computing, SAC 2006, pp. 1296-1303, (2006).

[117] Tratt, L. : *Model transformations in MT*. Science of computer programming, Elsevier, Vol. 68(3), pp. 196-213, (2007).

[118] Vardi, M.Y. and Wolper, P. : *An automata-theoretic approach to automatic program verification*, Proceedings of the 1^{st} LICS, Cambridge, pp. 332–344, (1986).

[119] Vardi M.Y. : *Automata-theoretic model checking*, VMCAI, Proceedings of the 7^{th} International Conference, LNCS, Vol. 4339, pp. 137–150, (2007).

[120] Vicente-Chicote, C., Losilla, F., Alvarez, B., Iborra, A. and Sanchez, P. : *Appling MDE to the development of flexible and reusable Wireless Sensor Networks*, International Journal of Cooperative Information Systems, Vol. 16, No. 3/4, pp. 393–412, (2007).

[121] Wang, G., Cao, G., La Porta, G. and Zhang, W. : *Sensor relocation in mobile sensor networks*, Proceedings of the 24th IEEE Conference on Computer Communications (INFOCOM 2005), Vol. 4, pp. 2302–2312, (2005).

[122] Wang, B., Wang, L., Lin, S.J., Wu, D., Huang, B.Y., Zhang, Y.T., Yin, Q. and Chen, W. : *A Body Sensor Networks Development Platform for Pervasive Healthcare*, Proceedings of the 3rd International Conference on Bioinformatics and Biomedical Engineering (ICBBE 2009), pp.1–4, (2009).

[123] Winkler, M., Street, M., Tuchs, K. and Wrona, K. : *Wireless Sensor Networks for Military Purposes*, Autonomous Sensor Networks Springer Series on Chemical Sensors and Biosensors, Vol. 13, pp. 365–394, (2013).

[124] Xuemei, L., Liangzhong, J. and Jincheng, L. : *Home healthcare platform based on wireless sensor networks*, Proceeding of the Fifth Intl. Conf. on Information Technology and application in Biomedicine, pp. 263–266, (2008).

[125] Yan, H., Xu, Y. and Gidlund, M. :*Experimental e-health applications in wireless sensor networks*, International Conference on Communications and Mobile Computing, pp. 563–567, (2009).

[126] Yu, E. : *Modeling Strategic Relationships for Process Reengineering*. Ph.D. thesis. Department of Computer Science, University of Toronto, Canada, (1995).

[127] Zatout, Y., Campo, E. and Llibre, J-F. : *Toward hybrid WSN architectures for monitoring people at home*, Proceedings of the International ACM Conference on Management of Emergent Digital EcoSystems (ACM MEDES 2009), Lyon, France, pp. 308–314, (2009).

[128] Zatout, Y., Kacimi, R., Llibre, J. and Campo, E. : *Using Wireless Technologies for Healthcare Monitoring at Home*, Proceedings of the 4th Joint IFIP Wireless and Mobile Networking Conference (WMNC 2011), pp. 1–6, (2011).

[129] Zhao, W., Ammar, M. and Zegura, E. : *A message ferrying approach for data delivery in sparse mobile ad hoc networks*, Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2004), pp. 187–198, (2004).