Vers la génération de modéles de simulation

Makhlouf Benkerrou

Master 2 Recherche en Informatique

Vers la génération de modéles de simulation

Rapport de stage

Équipe de recherche : SuSy

.

Réalisé par :

Mr Makhlouf Benkerrou

Encadré par :

Mr Philippe Le Parc
Mr Jean Vareille

Juin 2011

Résumé

Grâce aux facilités de communication et aux progrès technologiques, nous pouvons agir n'importe où et à tout moment : nous commençons à vivre dans un monde ubiquitaire avec des relations de communication diverses. Un des problèmes fondamentaux des systèmes ubiquitaires est leur complexité de conception ; l'Ingénierie Dirigée par les Modèles (*IDM*) se présente comme une approche prometteuse pour résoudre cette complexité.

Notre étude concerne les systèmes ubiquitaires et leur étude à l'aide d'outils de simulation. Elle s'appuie sur l'existence d'un méta-modèle générique des systèmes ubiquitaires développé au laboratoire *LISyC* par l'équipe de recherche *SuSy*. Dans une approche d'ingénierie dirigée par les modèles, les modèles-instances de ce méta-modèle peuvent être traités par différentes techniques de transformation. Le travail s'inscrit dans un objectif général qui vise à transformer ces instances en des données conformes au domaine d'application d'un outil de simulation. Il consistera dans un premier temps à identifier les outils de transformation et de simulation adaptés au méta-modèle générique et aux propriétés à vérifier.

Mots clés **:** Système Ubiquitaire, Ingénierie Dirigée par les Modèles, Simulation.

Abstract

With communication facilities and technological advances, we are able to act anywhere at any time : we begin to live in a world with various ubiquitous communication links. One of the fundamental problems of ubiquitous systems is their complexity of design, Model Driven Engineering (*MDE*) is presented as a promising approach to resolve this complexity.

Our study concerns the ubiquitous systems and their study by the use of simulation tools. It relies on the existence of a generic meta-model of ubiquitous systems developed in the laboratory LISyC by the research team SuSy. In an engineering approach to model-driven, model-instances of this meta-model can be treated by various processing techniques. The work is part of a general objective is to transform these instances of data conforming to the scope of a simulation tool. It will initially identify transformation tools and simulation adapted well to the generic meta-model and properties to check.

Keywords **:** Ubiquitous System, Model Driven Engineering, Simulation.

Remerciements

D'ABORD je tiens à exprimer mes plus vifs remerciements à mes encadrants M^r Philippe Le Parc et M^r Jean Vareille pour leur disponibilité, les nombreux conseils, orientations et encouragements qu'ils ont su me prodiguer durant mon stage à l'UBO.

Je voudrais également exprimer mes sincères remerciements à M^r Amara Touil. Merci à tous ceux qui ont contribué de près ou de loin, à la réalisation de ce travail.

Table des matières

Liste des figures

Introduction générale

AVEC l'augmentation croissante de la taille et de la complexité des logiciels, les techniques de génie logiciel doivent évoluer sans cesse pour permettre de gérer leur complexité et leur qualité. La grande tendance actuelle du génie logiciel est l'utilisation de modèles pour permettre une montée en abstraction par rapport aux langages de programmation, ce qui a donné naissance à l'Ingénierie Dirigée par les Modèles (*IDM*).

Grâce aux facilités de communication et aux progrès technologiques, nous pouvons agir n'importe où et à tout moment : nous commençons à vivre dans un monde ubiquitaire avec des relations de communication diverses. La complexité de conception des systèmes ubiquitaires augmente avec les besoins des utilisateurs qui ne cessent de s'accroître et avec l'hétérogénéité des composants introduits sur le marché. Il est nécessaire pour les concepteurs de ces systèmes de trouver une méthode pour spécifier leur définition. Dans ce contexte, l'Ingénierie Dirigée par les Modèles (*IDM*), se présente comme une approche prometteuse pour relever ce défi. À partir d'un modèle exprimé à l'aide d'un DSL (*langage spécifique au domaine*), et en se basant sur l'IDM, on peut par exemple générer d'autres modèles pour la validation (*test, simulation*) ou l'implémentation.

La problématique principale abordée par le projet SuSy est l'étude et la proposition de solutions logicielles pour un fonctionnement sûr des systèmes (*robots, systèmes de production, cartes hybrides, etc.*). Il s'agit plus particulièrement d'étudier et de proposer des solutions au niveau des langages, des interfaces, des environnements informatiques et des architectures logicielles destinés à la supervision et l'exploitation de tels systèmes, afin de rendre leur fonctionnement et leur utilisation sûre et sécuritaire. Différents formalismes et approches sont utilisés : méthodes formelles des approches avec de type model-checking (*réseaux de Petri, abstraction,...*) et des approches basées sur la démonstration (prouveurs, ...), étude de performances, techniques d'ordonnancement, GEMMA (*Guide d'Étude des Modes de Marche et d'Arrêt*), test, ...

Trois actions de recherches sont abordées dans ce projet :

- Architectures logicielles et mécanismes de sûreté en robotique,
- Contrôle sécuritaire via le Web de machines matérielles,
- Sécurité des échanges de données électroniques.

• Objectif du stage

Afin d'explorer la pertinence d'un système de télé-contrôle ubiquitaire, on souhaite être en mesure de le simuler. On cherche donc à étudier son modèle simulable et son obtention à partir du DSL générique développé au sein de laboratoire de recherche LISyC à l'UBO par l'équipe SuSy.

Notre travail consiste à générer le modèle simulable en se basant sur les concepts de l'ingénierie dirigée par les modèles, pour cela nous allons utiliser une transformation en deux étapes une première qui va ajouter des éléments au modèle toute en restant conforme au même méta-modèle (*transformation endogène*) et une deuxième transformation qui manipule le modèle issu de la première transformation afin d'aboutir à un fichier XML (*transformation exogène*). Ce dernier, pourra alors être importé dans un outil de simulation, comme PtolemyII, qui fournira une interface de simulation représentant le premier modèle.

• Organisation du mémoire

Le mémoire va être organisé comme suit :

- Le premier chapitre va présenter l'étude bibliographique. Elle couvre l'ingénierie dirigée par les modèles, les systèmes ubiquitaires et de télé-contrôle, et une introduction à la simulation,
- Dans le deuxième chapitre, on va présenter les outils de modélisation et de simulation que nous avons utilisé pour la réalisation de notre travail,
- Le troisième chapitre chapitre sera consacré à donner en premier lieu le méta-modèle de base pour les systèmes ubiquitaires, puis un méta-modèle spécifique pour un cas réel qui est issu de la société Terra Nova Energy et enfin, on donnera les différentes transformations réalisées pour aboutir au modèle simulable,
- Une conclusion générale viendra clore ce travail résumant les grands points qui ont été abordés dans ce rapport, ainsi que les perspectives.

Chapitre 1

État de l'art

1.1 Introduction

Notre travail consiste à générer un modèle simulable pour les systèmes ubiquitaires en se basant sur les outils de l'ingénierie dirigée par les modèles pour le passage d'un modèle vers le simulateur. Dans ce chapitre, nous faisons un aperçu des techniques existantes pour l'IDM, les systèmes ubiquitaires et de télé-contrôle, il se termine par quelques notions de simulation et la présentation de l'outil retenu.

1.2 L'Ingénierie Dirigée par les Modèles (IDM)

1.2.1 Présentation de l'IDM

L'ingénierie dirigée par les modèles [1], en anglais Model Driven Engineering (*MDE*), est le domaine qui met à disposition de la communauté des outils, concepts et langages pour créer et transformer des modèles, afin de mécaniser les processus que les ingénieurs suivent à la main. L'IDM se concentre sur une préoccupation plus abstraite que la programmation classique ce qui permet d'obtenir plusieurs améliorations dans le développement de systèmes complexes.
Il s'agit d'une forme d'ingénierie générative dans laquelle tout ou partie d'une application est engendrée à partir de modèles [2].

1.2.2 Concept fondamentaux de l'IDM

- **Modèle et systèmes**

Un modèle est une abstraction d'un système, décrit 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 [3]. Il est donc une simplification de la réalité du système à développer qui permet de mieux le comprendre. Un système peut être décrit par différents modèles liés les uns aux autres. Le système, au sens de l'IDM, est ce que l'on désire modéliser. Il est le réel [4].

- **Méta-modèle**

Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle, c.-à-d. le langage de modélisation. Il permet de définir précisément les concepts manipulés dans les modèles ainsi que

les relations entre ces concepts.

- **Méta-métamodèle : langage de méta-modélisation**

Pour pouvoir interpréter un méta-modèle il faut disposer d'une description du langage dans lequel il est écrit i.e. un méta-modèle pour les méta-modèles. On désigne ce méta-modèle particulier par le terme de méta-méta-modèle [5]. Pour limiter le nombre de niveaux d'abstraction, le méta-méta-modèle doit avoir la capacité de se décrire lui-même.

C'est sur ces principes que se base l'organisation de la modélisation de l'Object Management Groupe(*OMG*) généralement décrite sous une forme pyramidale (*figure 1.1*).

Le monde réel est représenté à la base de la pyramide (niveau M0). Les modèles représentant cette réalité constituent le niveau M1. Les méta-modèles constituent le niveau M2. Enfin, le méta-méta-modèle, est représenté au sommet de la pyramide (niveau M3).

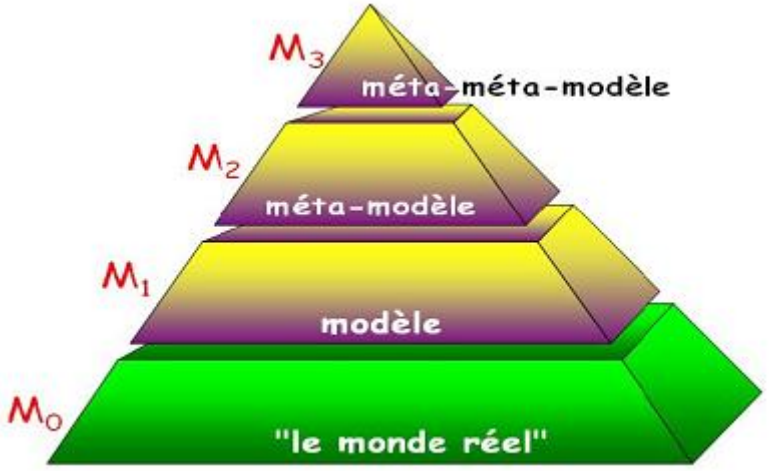

FIG. 1.1 – Pyramide de la modélisation OMG

1.2.3 DSL

Avec l'émergence de l'IDM, la création des langages spécifique au domaine (*DSL*) est devenue une partie essentielle de l'ingénierie des langages. Le concept est apparu dans la seconde moitié des années 1950 avec APT, un langage visant la programmation de machines-outils [6]. Le coût du développement d'un DSL devrait être modeste, par rapport au coût de développement d'un langage de programmation à usage général. L'article [7], propose une approche pour la construction d'une famille de DSL par la composition de plusieurs DSL.

Le *tableau 1* représente la liste de quelques langages généralement considérés comme des DSL.

DSL	Domaine d'application
BNF	Grammaire
Excel	Tableur
HTML	Pages Web
LATEX	Typographie
Make	Moteur de production de logiciels
SQL	Requêtes de bases de données

TAB. 1.1 – Exemples de langages spécifiques au domaine

1.2.4 Intérêt de transformation de modèles

D'un point de vue utilisateur, une transformation prend un ou plusieurs modèles en entrée et génère un ou plusieurs modèles en sortie [8]. Une transformation est le fruit de la combinaison de deux composantes : des règles de transformation et un moteur de transformation.La *figure1.2* illustre les principes des transformations de modèles en IDM.

Les transformations sont des processus permettant d'une part les manipulations de modèles, et d'autre part le passage d'un niveau de modèle à un autre [9]. Pour réaliser des transformations de modèles, les modèles doivent être exprimés dans un langage de modélisation étant lui-même défini à l'aide d'un méta-modèle.

On distingue deux types de transformations [10] : endogènes et exogènes. Une transformation est dite endogène si les modèles utilisés sont issus du même méta-modèle. Mais lorsque les modèles sources et cibles sont issus de différents méta-modèles, la transformation est dite exogène ou translation.

On peut subdiviser ces deux catégories de transformations :

Transformations endogènes
- Optimisation : amélioration les performances tout en maintenant la sémantique,
- Restructuration (*refactoring*) : transformation de la structure pour améliorer certains aspects de la qualité du logiciel,
- Simplification : transformation afin de réduire la complexité syntaxique.

Transformations exogènes
- Synthèse : transformation d'un certain niveau d'abstraction vers un niveau d'abstraction moins élevé,
- Rétro ingénierie : inverse de la synthèse,
- Migration : transformation d'un programme écrit dans un langage vers un autre langage du même niveau d'abstraction.

Un autre facteur important à prendre en compte dans les transformations concerne le niveau d'abstraction. On distingue les transformations horizontales et les transformations verticales.

Une transformation horizontale est une transformation où les modèles sources et cibles sont du même niveau d'abstraction. À l'opposé, dans une transformation verticale, les modèles impliqués sont de différents niveaux d'abstraction.

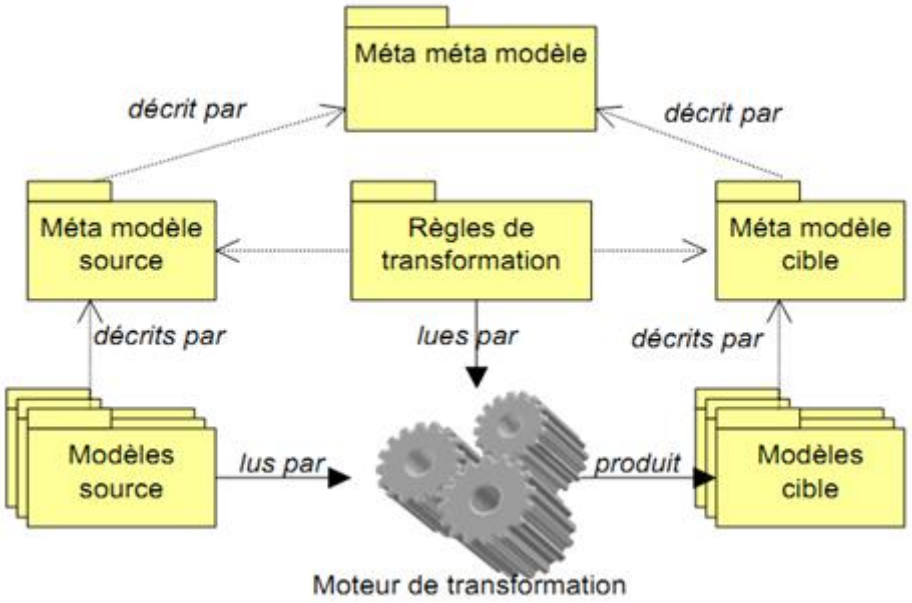

FIG. 1.2 – Principes des transformations de modèles en IDM

1.3 Systèmes ubiquitaires et systèmes de télé-contrôle

1.3.1 Systèmes ubiquitaires

Le terme «*système ubiquitaire*» a été proposé en 1988 par Mark Weiser [11]. Pour lui il est difficile de vivre et de travailler sans l'aide d'ordinateurs. Weiser considère que les systèmes basés sur les technologies de l'information et de la communication, peuvent nous soutenir dans notre vie quotidienne tout en devenant pratiquement invisible pour nous.

La notion d'ubiquité est souvent définie comme étant la possibilité d'être à plusieurs endroits en même temps. Dans le domaine des technologies de l'information, cette notion d'ubiquité peut être affinée dans, au moins, deux façons qui peuvent être opposés [12].

La première façon considère que les utilisateurs sont entourés de systèmes «*intelligents*», qui livrent l'information nécessaire. Dans ce cas, les installations informatiques sont utilisées pour permettre la disponibilité de toutes les informations dont l'utilisateur a besoin partout et à tout moment.Ce sont les informations qui sont ubiquitaires. Avec la popularité de l'informatique mobile, les applications sensibles au contexte deviennent clairement nécessaires. Ce type d'applications permet aux utilisateurs mobiles un accès universel à des services quelque soit le contexte utilisateur, y compris son environnement informatique. Les défis pour ces applications sont de faciliter la manipulation à la fois des collections et des analyses du contexte et de réagir dynamiquement à chaque

évolution pertinente dans le contexte d'exécution. L'article [13] propose une approche générique et extensible pour modéliser toute application sensible au contexte utilisant l'ingénierie dirigée par les modèles. L'illustration de cette approche est faite sur un exemple du E-commerce.

La deuxième approche permet d'obtenir des informations sur un système situé quelque part et de pouvoir agir en toute sécurité sur lui, tout en étant pas situé à proximité : dans ce cas, les installations informatiques sont utilisées pour permettre à une personne d'être «*virtuellement*» présente dans plusieurs endroits en même temps. Cette approche rentre dans le contexte de notre travail.

Suivant cette approche, on peut considérer les systèmes de télé-contrôle comme une variante de systèmes ubiquitaires en considérant la séparation géographique entre les deux éléments : le contrôleur et le contrôlé ainsi que l'échange d'information entre eux (*commande de contrôle et sa réponse*)

1.3.2 Systèmes de télé-contrôle

Il existe plusieurs travaux de recherche concernant les systèmes de télé-contrôle, on va citer ci-dessous quelques exemples.

Afin de faciliter l'étude de la commande robot manipulateur pour les utilisateurs qui ne peuvent pas accéder à la manipulation d'un robot réel, l'article [14] propose la mise en place d'une architecture à base de télé-opération pour contrôler un bras manipulateur en utilisant l'Internet.

Le succès que connaît actuellement Internet ainsi que la notion du télétravail, sont à l'origine de la réalisation du système ARITI (*Augmented Reality Interface for Telerobotic applications via Internet*) qui permet à un Opérateur Humain non expérimenté de superviser et de commander un robot depuis n'importe quelle machine connectée à Internet.L'architecture du système ARITI est basée sur le principe de Réalité Augmentée en vision indirecte (*des informations sont superposées à des images vidéo obtenues par une caméra*). L'article [15] présente une étude réalisée pour l'extension du système ARITI vers un système de télétravail collaboratif.

La tension électrique et la vitesse sont généralement contrôlées dans le processus traditionnel de télé-opération, en supposant que le *Web* se compose d'un certain nombre de systèmes de production à entrée et sortie unique. Cette hypothèse se traduit souvent par de grandes interactions qui se produisent dans le système en circuit fermé entre les boucles de contrôle. Par conséquent, le contrôle haute qualité est difficile à réaliser. Dans [16] le contrôle des processus est traité comme un problème de servomécanisme multi-variable. Deux types de contrôleur de conception : le «*contrôleur servomécanisme à contrôle parfait*» et le «*réglage*». Ces deux types sont étudiés et mis en œuvre sur une machine industrielle.

Avec le développement des e-technologies et des réseaux sous IP pour les automatismes, le contrôle des machines pourra faire appel dans le futur aux protocoles et outils développés pour le *Web*. Mais ces solutions, tout comme Internet, ne sont pas déterministes, la qualité de service n'y est pas garantie que ce soit en termes de délais d'échanges des informations ou en termes de

bande passante. L'article [17] décrit une architecture générique qui prend en compte le caractère imprévisible de la communication pour réaliser le contrôle à distance en utilisant le *Web* et le suivi des résultats de mesures collectés lors des connexions des utilisateurs distants.

1.3.3 Systèmes de sûreté

Les systèmes technologiques sont au cœur de nombreuses applications permettant d'aider l'humain dans des tâches dangereuses, trop complexes ou impossibles. Parmi ces applications, certaines peuvent blesser des humains, ou provoquer des dégâts sur les biens matériels ou l'environnement. Ces systèmes sont des systèmes à sécurité critique et leur utilisation est conditionnée par la confiance que l'humain leur accorde. Cette confiance est d'autant plus importante que l'on assiste aujourd'hui à des transferts de responsabilités de l'humain vers les dispositifs technologiques. La thèse [18] propose une approche pour appréhender la maîtrise de la sécurité d'une application de robotique de service qui se base sur le langage de modélisation UML.

L'article [19] détaille les différents enjeux liés à la sûreté de fonctionnement des systèmes informatique en proposant quelques approches pour surmonter les défauts qui peuvent subvenir lors de la création ou la mise en œuvre d'un système informatique.

1.4 Introduction à la simulation

La simulation consiste à conduire des expérimentations sur une modélisation d'un système et à interpréter les observations dans le but de prendre une décision.

1.4.1 Objectifs de la simulation

La simulation permet d'améliorer la compréhension d'un système sans devoir le manipuler réellement, soit parce qu'il n'est pas encore défini ou il n'est pas disponible, soit parce qu'il ne peut pas être manipulé directement en raison des coûts, du temps, des ressources ou du risque [4].

La simulation peut être réalisée à partir d'un scénario prédéfini ou de manière interactive. Cette dernière solution permet alors à l'utilisateur de construire progressivement la trace d'exécution à partir des événements qu'il injecte. L'utilisateur peut également voir évoluer son modèle tout au long de l'exécution et ainsi contrôler visuellement le comportement du système pour une exécution donnée [4].

1.4.2 Domaines d'applications

Historiquement, les premiers simulateurs ont été inventés pour la formation à moindre coûts des pilotes d'avion. La simulation est intégrée dans divers domaines d'applications :

- Domaine militaire,
- Nucléaire,
- Jeux,
- Et bien d'autres (*gestion d'hôpitaux, gestion de production, transports, météo, ...*).

1.4.3 L'étape de modélisation

L'étape de modélisation est la phase la plus essentielle de la simulation. Pour avoir une bonne modélisation plusieurs points doivent être abordés :

- Analyser le problème à résoudre et définir les objectifs liés au cahier des charges,
- Définir les entrées et les sorties du système,
- Définir les interactions entre les éléments et construire un modèle conceptuel,
- Définir la dynamique du système i.e. le comportement du système au cours de temps dans son environnement (*super système*),
- Etudier les éléments les plus pertinents du système.

La *figure 1.3* représente la *figure 1* de [20] dans laquelle on a ajouté la notion de *super système* qui présente le système et son environnement extérieur afin du faire une bonne analyse et de prévoir les scénarios qui peuvent engendrer des pannes dans notre système.

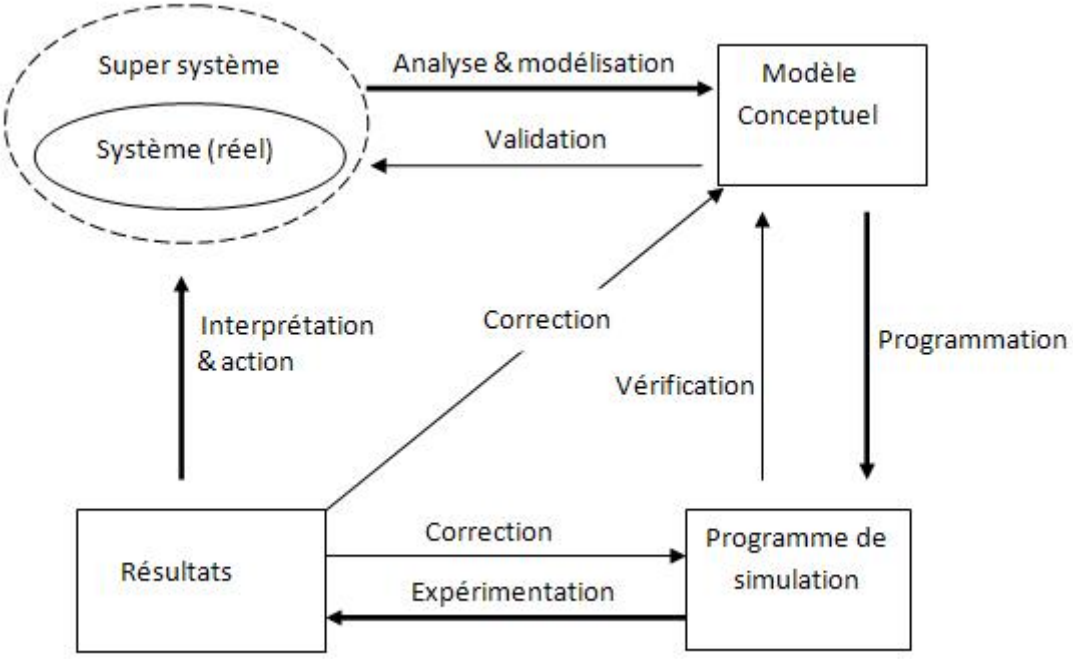

FIG. 1.3 – Méthodologie d'une simulation

1.4.4 Vérification et validation des modèles

La simulation est souvent utilisée pour savoir «*que se passerait-il si ?*». Le problème est comment avoir confiance dans le modèle ?

- **Vérification**

Cherche d'abord à répondre à la question «*Construisons-nous correctement le modèle ?*». Le but est de prouver la cohérence du modèle, de s'assurer de la bonne utilisation des moyens de modélisation et de rendre compte de la description des exigences qui ont prévalues à l'existence de ce modèle. Par définition, la vérification est la confirmation par examen et apport de preuves tangibles

(*informations dont la véracité peut être démontrée, fondée sur des faits obtenus par observation, mesures, essais ou autres moyens*) que les exigences spécifiées ont été satisfaites [21].

• **Validation**

Cherche à répondre à la question «*Construisons-nous le bon modèle ?*». La validation se définit comme la «*confirmation par examen et apport de preuves tangibles que les exigences particulières pour un usage spécifique prévu sont satisfaites. Plusieurs validations peuvent être effectuées s'il y a différents usages prévus*». Ainsi, la validation permet de lier le modèle à la réalité attendue ou perçue [21].

1.4.5 Outils de simulation

Il existe plusieurs outils de simulation comme : Labview [22], Simulink/matlab [23], Scicos/Scilab [24], PtolemyII [25], Occam [26] etc. Dans notre cas d'étude qui consiste à simuler un système ubiquitaire, on a besoin d'un simulateur qui utilise des entités communicantes, pour cela on va décrire les deux outils, PtolemyII et Occam, qui sont plus appropriés à notre cas, puis on va faire une comparaison afin de choisir entre les deux. Les autres outils de simulation sont utilisés en générale pour les problèmes scientifique et les calculs mathématiques.

• **Occam**

Occam est un langage de programmation qui a été développé par Inmos pour la programmation de ses Transputers, mais il existe des implémentations pour d'autres plates-formes. Occam est un langage procédural classique qui offre, l'exécution d'instructions séquentiellement (avec SEQ), l'exécution des instructions en parallèle (avec PAR) et même la mise en «*parallèle asynchrone*» de processus (avec ALT) pour une exécution non-déterministe d'un parmi plusieurs.

• **PtolemyII**

PtolemyII a été en cours de développement depuis 1996, il est le successeur de Ptolemy classique qu'était le premier environnement de modélisation pour mettre en œuvre les modèles multiples du calcul, hiérarchiquement combinés [27] Il est basé sur un concept central qui est l'acteur.

• **Choix d'outil**

L'efficacité d'un outil en termes de puissance de calcul pour réaliser la simulation et la capacité de s'intégrer dans un processus industriel sont des critères importants à prendre en compte pour le choix d'un outil. Les outils interprétant les résultats de la simulation à l'aide des graphes, manquent d'efficacité. Un outil de simulation par animation est donc le plus souvent utilisé pour concevoir efficacement et rapidement un premier modèle dans le but de l'améliorer et le valider par simulation.

Pour le langage Occam, la conception du modèle de simulation pour un système donné, nécessite la définition des différents processus, canaux, propriétés, modèle de calcul, etc. Du point de vue sémantique, le méta-modèle de télé-contrôle ubiquitaire met en œuvre des systèmes à entités communicantes qui peuvent correspondre aux processus communicants d'Occam. Avec ce langage on s'approche de la création de notre propre simulateur et de la définition complète des différents mécanismes de simulation. Le seul gain est celui de fondre notre approche de simulation sur l'aspect

de concurrence et des processus communicants par des canaux.

Pour le langage PtolemyII, plusieurs acteurs génériques sont présents. Ces acteurs ont fait l'objet de tests et ont été appliqués dans la conception et la simulation de plusieurs systèmes. La version VisualSense de ce langage comporte des acteurs génériques des réseaux de capteurs. Ces acteurs couvrent les notions de capteur-actionneurs et de canaux de communication filaires et sans fil. De plus, PtolemyII offre une large gamme de modèles de calcul présentés par le concept (*Directeur*) à événements discrets (*DE*), flux de données synchrone (*SDF*), rendez-vous, etc. L'aspect graphique dans PtolemyII est également un point fort qui permet d'avoir un suivi visuel de la simulation. Du point de vue sémantique, la correspondance entre les entités communicantes de notre méta-modèle et les acteurs de PtolemyII offre une bonne approche d'équivalence.

1.5 Conclusion

Au cours de ce chapitre, Nous avons commencé par le recadrage de notre sujet en évoquant l'ingénierie dirigée par les modèles en donnant ses concepts fondamentaux et les différents types de transformations. Puis l'accent a été mis sur les systèmes ubiquitaires et de télé-contrôle dont on a cité quelques travaux de recherche. Dans la dernière partie de notre état de l'art nous avons donné une introduction à la simulation et mentionné les simulateurs les plus utilisés dans différents domaines notamment ceux qui sont utilisés pour les systèmes ubiquitaires.

Le chapitre suivant va décrire les outils de modélisation qui sont Ecore pour la construction des méta-modèles et Kermeta pour les manipulations et les transformations. Pour la simulation on va présenter quelques concepts clés de ptolemyII, qui est le simulateur que nous avons choisi.

Chapitre 2

Outils de modélisation et de simulation

2.1 Introduction

Dans ce chapitre on décrit l'outil de simulation *PtolemyII* et le langage de modélisation *Ecore* qui est utilisé par notre équipe pour le méta-modèle du système ubiquitaire. Enfin nous allons donner quelques notions sur l'outil *Kermeta*, utilisé pour les manipulations et transformations des modèles.

2.2 PtolemyII

2.2.1 Historique de l'environnement PtolemyII

• **Gabriel (1986-1991) :** c'est la première version du logiciel, créé à l'Université de Berkeley (*Californie*), il est écrit en Lisp (*destiné au traitement de signal*),

• **Ptolemy classique (1990-1997) :** premier environnement de modélisation à supporter les modèles de calcul multiples combinés hiérarchiquement, il est initié par les professeurs Edouard Lee et Dave Messerschmitt, écrit en C++,

• **PtolemyII : 1996** développé par le groupe Ptolemy, il intègre la notion objet orienté acteur (*classe, sous-classe, instance*) avec possibilité d'héritage, introduit la notion de domaine polymorphisme (*où les composants peuvent être conçus pour fonctionner dans des domaines multiples*) et des modèles modaux (*où des machines à états finies sont combinées hiérarchiquement avec d'autres modèles de calcul*), ainsi qu'un modèle de temps continu.

2.2.2 Concepts de PtolemyII

PtolemyII est une plateforme basée sur un concept central qui est l'acteur. Elle permet la modélisation (*conception et simulation*) des systèmes hétérogènes. PtolemyII est doté d'un ensemble (*Framework*) de composants spécialisés et utilise XML pour la persistance des données et des composants. Parmi ces composants, nous notons : les acteurs, les ports, les relations, et les directeurs.

Les modèles sont représentés sous la forme de graphes où les nœuds représentent les entités et les arcs sont des relations. Les entités sont des acteurs et les relations sont des liaisons de communica-

tion entre les acteurs. PtolemyII offre une infrastructure unifiée permettant de supporter différents modèles de calcul, dont les implémentations sont appelées des domaines.

2.2.3 Architecture de PtolemyII

L'architecture globale de PtolemyII se compose d'un ensemble de bibliothèque (*package*) qui fournit l'appui générique pour tous les modèles de calculs. Tous les packages de PtolemyII sont regroupés sous un seul package nommé Ptolemy. Voici quelques packages de PtolemyII :

• **Ptolemy.kernel :** fournit l'architecture logicielle pour les modèles sous PtolemyII,

• **Ptolemy.actor :** regroupe les classes définissant les entités exécutables qui reçoivent et émettent des données à travers les ports, y compris la base Directeur (*étendue Ptolemy.domains*),

• **Ptolemy.domains :** comporte un ensemble de sous-packages dont chacun est l'implémentation d'un modèle de calcul (*appelé domaine*),

• **Ptolemy.data :** inclut les classes permettant l'encapsulation et la manipulation des données echangees entre les acteurs.

2.2.4 Les éléments graphiques

1. **Acteur**

   L'acteur est un agent autonome de raisonnement (*composant*) qui est la base de PtolemyII, il est une entité exécutable qui communique avec d'autres acteurs. Il possède une infrastructure neutre, indépendant du modèle de calcul, et se compose de ports et de paramètres.

2. **Acteur Composite**

   L'acteur composite est un sous-modèle (*ou partie du modèle*) du domaine de la simulation, il contient un ou plusieurs acteurs mais ne possède pas obligatoirement de directeur (*voir ci-dessous*), il a pour but d'éclairer le modèle c.à.d. de regrouper différentes parties du modèle.

3. **Port**

   Les ports sont des points de communication entre les acteurs. Ils permettent la transmission de données et de messages à travers les voies de transmission.

4. **Paramètre**

   Les paramètres permettent de configurer les différentes opérations d'un acteur, ils sont accessibles à tous les niveaux inférieurs par rapport à l'endroit où ils ont été introduits.

5. **Directeur**

   Le directeur est le moteur d'exécution du modèle, il implémente différents modèles de calcul. Tout modèle peut être soit planifiable, soit chronométré dans le temps.

Différents types de modèles peuvent être implémentés :
✓des systèmes en Temps Continu (*C.T*),
✓des systèmes réseaux (*C.I : Interaction entre Composant*),
✓des Flux de Données Synchrones *(S.D.F)*,
✓des Evénements Discrets (*D.E*),
✓des Processus Séquentiels Communicants (*C.S.P*).

Le directeur implémente et contrôle la séquence d'exécution, c.à.d. l'ordre de lancement des acteurs, la progression du temps et la taille des tampons dans la mémoire de chaque acteur.

Il peut y avoir plusieurs directeurs dans un modèle, c.à.d. un directeur extérieur qu'on appelle directeur exécutif qui contrôle l'acteur composite, et un directeur intérieur qu'on appelle directeur local, contrôle les acteurs contenus dans l'acteur composite. Ce dernier est responsable de l'exécution des composants à l'intérieur de cette entité.

2.2.5 VisualSense

La modélisation des réseaux sans fil, nécessite une représentation sophistiquée et l'analyse des canaux de communication, des capteurs, des protocoles de réseau ad-hoc, des stratégies de localisation, des protocoles de contrôle d'accès, de consommation d'énergie dans les nœuds de capteurs, etc.

VisualSense [28] est un cadre de modélisation et de simulation pour les réseaux sans fil et les réseaux de capteurs. Ce cadre de modélisation est conçu pour supporter une construction basée sur des composants de ces modèles. Il prend en charge la définition axée sur les acteurs de nœuds de réseau, les canaux de communication sans fil, des supports physiques tels que les canaux acoustiques, et sous-systèmes câblés. L'architecture logicielle se compose d'un ensemble de classes de base pour la définition de canaux et de nœuds de capteurs, une bibliothèque de classes qui fournissent certains modèles de canaux spécifiques et des modèles de nœud, et un cadre de visualisation extensible.

2.3 Environnement de modélisation

2.3.1 Ecore (*Eclipse core*)

Ecore [29] est un langage de méta-modélisation graphique de haut niveau qui défini la syntaxe abstraite d'un méta-modèle. Toutefois, la disponibilité des outils pour générer des implémentations d'un modèle Ecore signifie qu'ils sont souvent utilisés comme modèles d'application. Lorsqu'il est utilisé à cette fin, l'expressivité limitée de la syntaxe abstraite Ecore signifie que seuls les aspects de définition de données d'une application peuvent être capturés. Les syntaxes concrètes disponibles pour créer un modèle Ecore dépendent des plugins qui ont été installés dans l'espace de travail Eclipse.

2.3.2 Kermeta

Kermeta [30] est une plateforme open-source de méta-modélisation développée par l'équipe Triskell (*Université de Rennes1*). Elle dispose d'un environnement de développement basé sur EMOF (*Essential Meta-Object Facilities*) dans un environnement Eclipse. Un des objectifs de Kermeta est de permettre la manipulation des modèles, c'est-à-dire de créer, parcourir et modifier des modèles. Le langage Kermeta est une sorte de dénominateur commun des multiples langages qui, coexistent actuellement dans le paysage de l'IDM, en particulier les langages de métadonnées (*Ecore, EMOF*), de transformation de modèles (*QVT, MTL, ATL*), de contraintes et de requêtes (*OCL*) et d'actions (*Action Semantics, Xion*).

2.4 Conclusion

Dans ce chapitre nous avons présenté quelques notions clés du simulateur PtolemyII que nous allons utiliser pour la simulation, et on a mis l'accent sur le langage de modélisation Ecore ainsi que Kermeta qui sera notre langage de manipulation et de transformation de modèles afin d'aboutir au modèle simulable sous PtolemyII.

Le chapitre suivant va être consacré pour décrire notre travail qui consiste à passer d'un modèle de haut niveau vers un modèle simulable.

Chapitre 3

Modèle vers simulations

3.1 Introduction

Au lieu de créer les composants qui constituent notre système dans l'outil de simulation, on peut profiter de la généricité de ce dernier et éviter de refaire les éléments qui sont en commun entre les composants, pour cela il suffit de décrire un modèle avec un minimum d'information et construire les composants sur le simulateur automatiquement en parcourant le modèle. Cette méthode va permettre de simuler des systèmes ubiquitaires à grande échelle plus rapidement.

Ce chapitre sera consacré d'abord à présenter le méta-modèle développé au sein de l'équipe de recherche puis à décrire le méta-modèle d'un cas réel de système ubiquitaire (*Terra Nova Energy*). Ensuite on construira un modèle sous Kermeta en sachant les différents outils qui le compose à savoir (*capteur, capteur-actionneur, répéteur, BoxaNova et medium de communication*), ce modèle doit être conforme au deuxième méta-modèle qui est spécifique à Terra Nova Energy. Après avoir décrit le modèle de notre système, on passe à la transformation, cette dernière va se faire en deux étapes : la première est une transformation *endogène* et *horizontale* qui consiste à préparer notre modèle pour la simulation c.à.d. ajouter des ports pour les composants et ajouter des medium dédiés à la simulation. La deuxième étape de transformation consiste à générer un fichier XML, ce dernier donne notre interface de simulation en l'important sur PtolemyII.

3.2 Système Terra Nova Energy

Terra Nova Energy (*TNE*) développe et commercialise des logiciels internet pour permettre à ses clients (*industriels, tertiaires, et collectivités*) de mesurer, visualiser, analyser et piloter l'ensemble de leurs consommations d'énergie afin de détecter rapidement les anomalies et de réaliser ainsi des économies significatives et durables. Le choix de Terra Nova Energy comme exemple pour un système ubiquitaire est dû au fait qu'elle utilise des objets hétérogènes et des entités communicantes. Notre équipe de recherche a utilisée quelques concepts de Terra Nova Energy pour la création de méta-modèle des systèmes ubiquitaires, en tenant compte de la communication et des interactions avec les composants du système. Un système *TNE* est formé par des capteurs, des capteurs-actionneurs, des répéteurs et des BoxaNova.

• **Capteur**

Le capteur est un sous-compteur qui calcul plusieurs paramètres électriques, permet la transmission de données sans fil. Par défaut le capteur envoie un paquet toutes les 20 secondes, 3 paquets sont nécessaire pour mettre à jour tous les registres ModBus, donc on obtient des informations actualisées toutes les minutes. En cas de réseau large ou réseau avec goulot d'étranglement avec ce temps le réseau risque d'être inondé et il y aura des paquets perdu.

• **Capteur-actionneur**

En plus de la fonction de capteur il peut agir sur un dispositif matériel, par exemple la fermeture d'une vanne.

• **Répéteur**

Permet d'étendre la distance de la transmission entre les capteurs (actionneurs) et leur concentrateur.

• **Concentrateur**

C'est une passerelle qui gère le réseau sans fil et collecte les données envoyées périodiquement par les capteurs.

• **BoxaNova**

Est un boitier qui contient : un répéteur et deux concentrateurs l'un peut supporter jusqu'à 200 capteurs et l'autre peut supporter 32 capteurs-actionneur. Elle sert de passerelle entre les différentes sources d'information de comptage et les utilisateurs de ces informations.

3.3 Méta-modèle pour les systèmes ubiquitaires

Dans [12] les auteurs proposent un méta-modèle (*figure 3.1*) qui a pour but de décrire le domaine ubiquitaire. Il est basé sur les systèmes à entités communicantes.

- Un système est représenté par l'élément (*System*) et il peut contenir d'autres systèmes grâce à la relation «*ownedSystem*».
- L'élément (*Entity*) représente les entités qui s'identifient au système par la relation «*itsEntity*». Elle peut être composée d'autres entités par la relation «*ownedEntity*».

Différents concepts sont attachés à l'élément (*Entity*) :

- **Structure :** défini la nature des différentes parties physiques. Elle peut être mécanique, chimique ou biologique.
- **Task :** permet de définir les taches réalisées par les entités.
- **Behavior :** élément définissant l'organisation des différentes tâches exécutées par les entités.
- **Data :** permet de définir des propriétés des entités ou de l'information échangée entre des entités communicantes.
- **ContactInterface :** élément permettant la connexion entre les entités.
- **Message :** élément décrivant l'échange de données entre les entités.
- **Energy :** les entités communiquent entre elles et accomplissent des tâches. Ce comportement nécessite une énergie à consommer. Ce concept est représenté par l'entité Energy.
- **Timeclock :** permet de mesurer, d'enregistrer et/ou d'indiquer le temps.

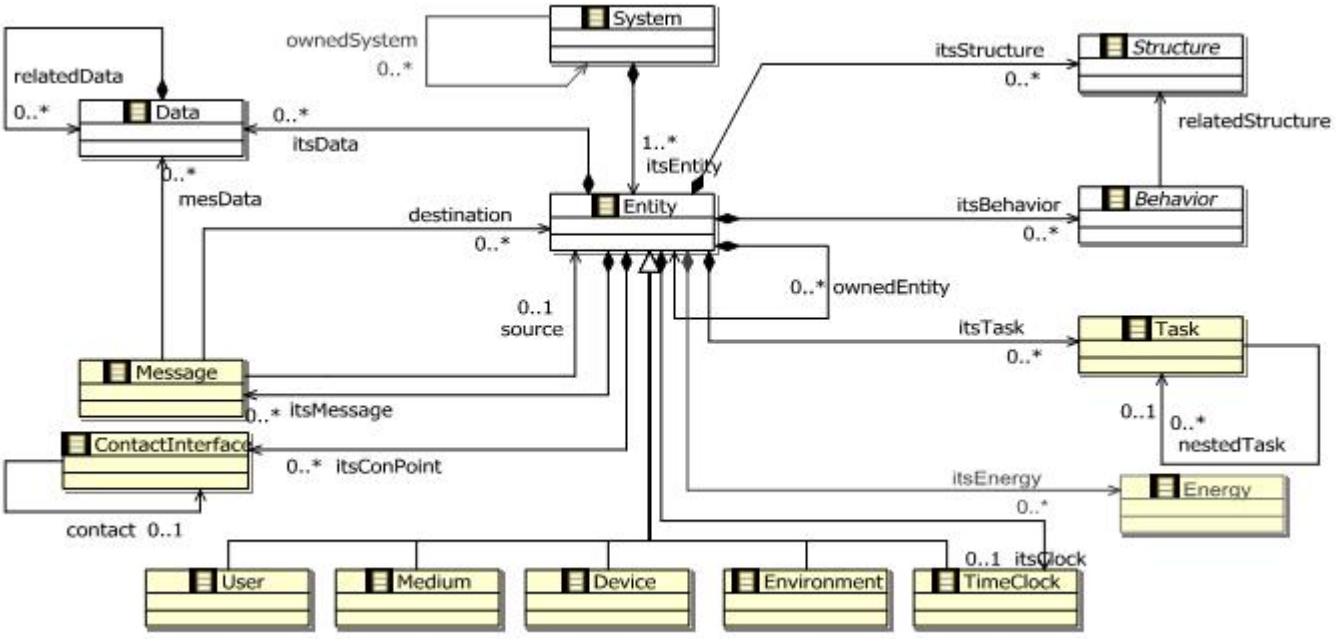

FIG. 3.1 – Méta-modèle de systèmes ubiquitaire

Afin de prendre en compte les entités classiques rencontrées dans les systèmes ubiquitaires les auteurs ajoutent quelques concepts spécialisés :

- **Device :** représente un composant physique, tel qu'un capteur, un actionneur, un routeur, une unité de calcul, etc.
- **Medium :** assure le transport des messages et de l'information entre les différentes entités.
- **User :** élément utilisé pour modéliser l'interaction humaine avec les entités du système. Il s'agit donc d'un avatar de l'utilisateur.
- **Environment :** une entité particulière modélisant les conditions environnementales du système. En effet, les différentes entités qui composent le système évoluent sous certaines conditions physiques imposant des contraintes qui affectent et influencent leurs comportements. Il est donc nécessaire de les modéliser, au moins partiellement.

3.4 Méta-modèle Terra Nova Energy

L'équipe a complété le méta-modèle pour décrire les systèmes utilisés par Terra Nova Energy. Ces systèmes TNESystems permettent l'acquisition de données pour gérer la consommation d'énergie. La composition de ces systèmes comporte des Devices de type : capteur, actionneur, routeur et concentrateur. Les entités d'un système TNE communiquent entre eux par des Mediums qui forment le ou les réseaux de communication.

La *figure 3.2* représente le méta-modèle décrivant le contexte général de Terra Nova Energy.

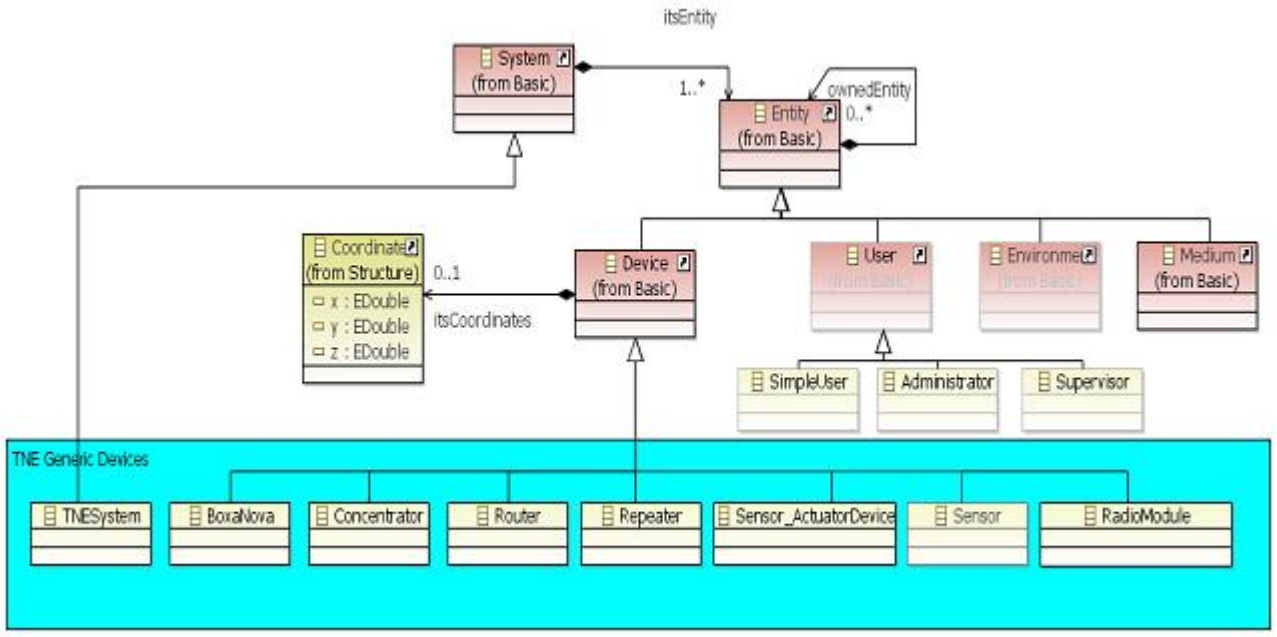

FIG. 3.2 – Méta-modèle pour TNESysteme

- L'élément «*TNESystem*» représente le système «*Terra Nova Energy*», il hérite de l'élément système qui est l'élément système de la figure 3.1.
- L'élément «*Entity*» est l'élément «*Entity*» de la figure 3.1.
- L'élément «*Coordinate*» représente les coordonnés géographiques d'un «*Device*» dans notre système ubiquitaire.
- L'élément «*User*» représente l'élément «*User*» de la figure 3.1.
- Les éléments «*SimpleUser*», «*Administrateur*» et «*Supervisor*» : représentent des éléments qui indiquent la nature de l'entité «*User*»
- Les éléments «*BoxaNova*», «*Concentrateur*», «*Router*», «*Repeater*», «*Sensor_ ActuatorDevice*» et «*Sensor*» représentent les devices de bases utilisés par Terra Nova Energy.

3.5 Principe de travail

Le contexte de notre travail consiste à créer des instances conformes au méta-modèle de la *figure 3.2*, puis de permettre le passage des instances vers le simulateur (*PtolemyII*) en utilisant des outils de transformation à l'aide de Kermeta.

La *figure 3.3* représente le contexte de notre travail.

Pour réaliser le travail, on a besoin de 3 étapes : la première consiste à représenter le système «*à la main*» pour définir l'ensemble des composants matériels et des mediums de communication qui seront utilisés et les différents endroits dans lesquels ils seront implantés. La deuxième consiste à donner la représentation de notre modèle sous Kermeta. La troisième étape permet la représentation de notre système sous une interface de PtolemyII qui sera prête pour la simulation.

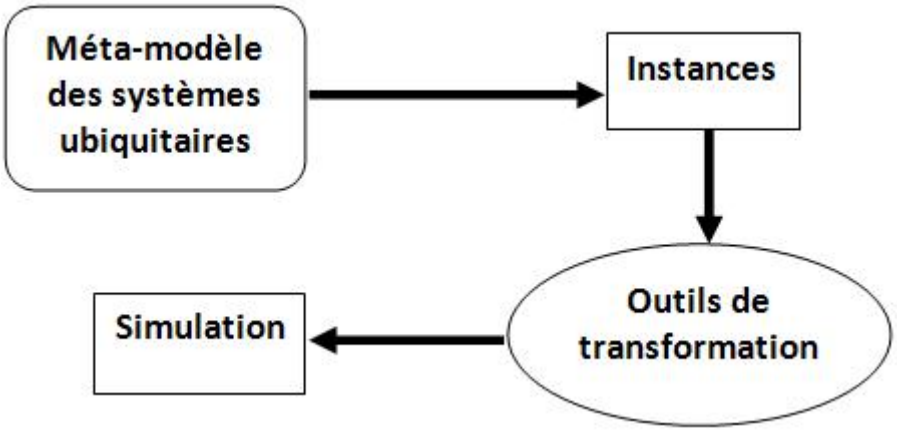

FIG. 3.3 – Contexte du travail

Avant de décrire les étapes de transformation, on commence d'abord par décrire les éléments de PolemyII qui vont représenter les composants de notre système.

3.6 Choix des éléments

Afin de représenter un système ubiquitaire de *TNE* sous un modèle simulable de PtolemyII, on a besoin d'un ensemble de composants standard de PtolemyII pour représenter les entités de notre système.

3.6.1 PowerLossChannel

Nous avons choisi «*PowerLossChannel*» (figure 3.4) parmi un ensemble d'outils de PtolemyII pour représenter l'entité «*Medium*» du méta-modèle «*Terra Nova Energy*». Il s'agit d'un modèle de canal sans fil avec une formule de propagation de puissance spécifiable. Il contient des paramètres par défaut qui sont :

- **defaultProperties :** un enregistrement qui contient deux champs {*range, power*}. Le champ range exprime la portée et le champ power exprime la puissance. Sa valeur par défaut est {*range=infinity, power=infinity*} cette valeur exprime le fait qu'il n'y a pas d'atténuations, ni de délais.
- **propagationSpeed :** la vitesse de propagation. Ceci détermine le délai entre l'émission et la réception. Par défaut il a la valeur infinie qui signifie qu'il n'y a pas de retard.
- **powerPropagationFactor :** c'est la formule par défaut pour le facteur de perte de puissance. C'est une expression qui dépend d'une variable «*distance*», qui est la valeur de la distance entre un émetteur et le récepteur.

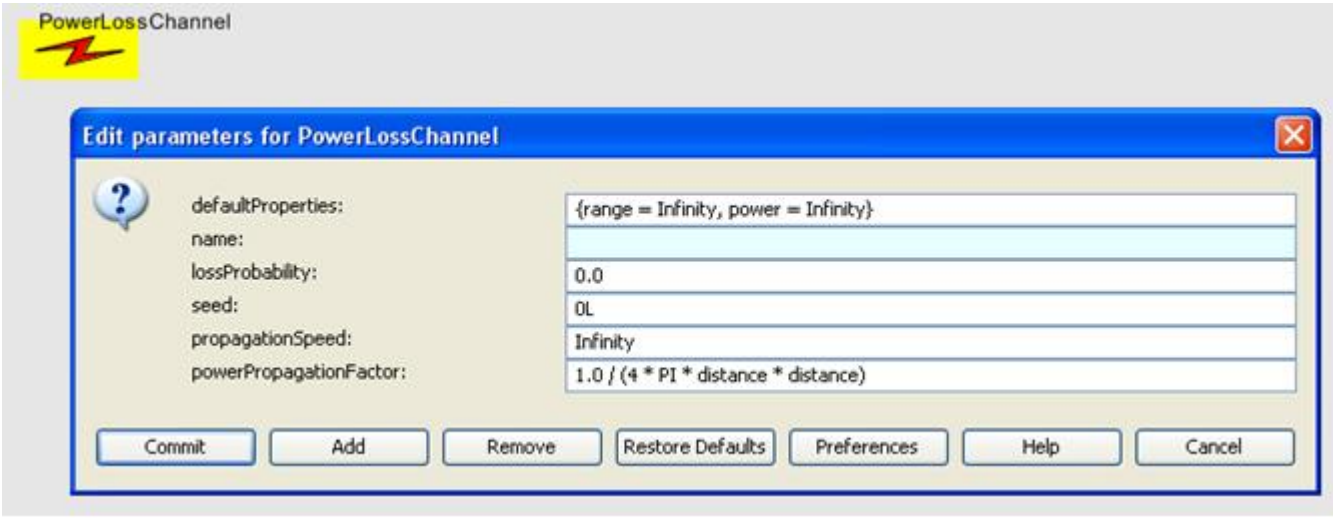

FIG. 3.4 – interface par défaut «*PowerLossChannel*»

3.6.2 WirelessComposite

Il s'agit d'un acteur composite pour une utilisation dans le domaine sans fil. On représentera chaque entité «*capteur, capteur-actionneur, répéteur, concentrateur, routeur et BoxaNova*» de notre système par un *WirelessComposite*. Pour cela on lui ajoute des ports d'entrées/sortie pour permettre la communication entre l'entité en question et les différents acteurs de systèmes via des mediums de communication. À l'intérieur de *WirelessComposite* on ajoute des *ModalModel* pour décrire le comportement de l'entité.

La *figure 3.5.a* représente l'interface extérieure initiale de *WirelessComposite* et la *figure 3.5.b* illustre son interface après l'ajout d'un port d'entrée et un port de sortie.

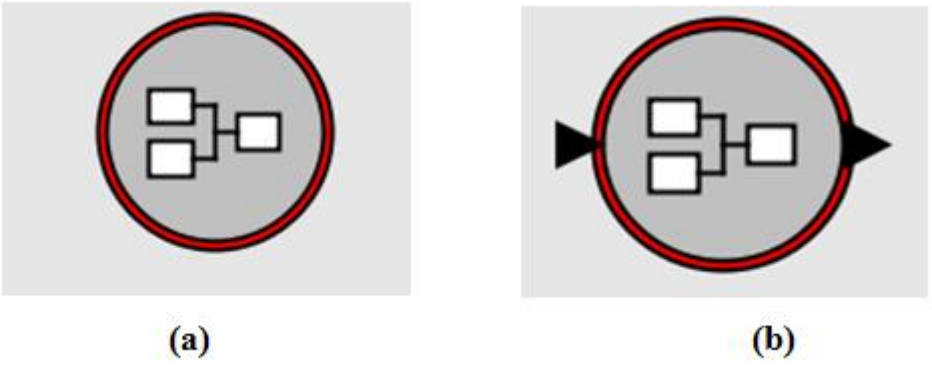

FIG. 3.5 – interfaces extérieurs *WirelessComposite*

La *figure 3.6* illustre l'intérieur d'un capteur, qui est représenté par un WirelessComposite pour lequel on a ajouté deux ModalModel, un port d'entrée et un port de sortie. On a aussi ajouté deux composants qu'on a reliés, avec le port «*power*» de ModalModel «*receptioncontrol*» afin de mesurer la puissance de signal qui rentre dans le capteur.

Les systèmes TNE sont en général installés dans des bâtiments. Leur placement doit tenir compte

des obstacles présents (*les murs*) car l'atténuation apportée par les murs varie suivant le type de communication (*WIFI, ZigBee, Bluetooth, etc.*).

Le composant Locator et TimeDelay sont combinés afin de déterminer la date d'arrivée d'informations dans le ModalModel «*sensing*» (*decrit ci-dessous*).

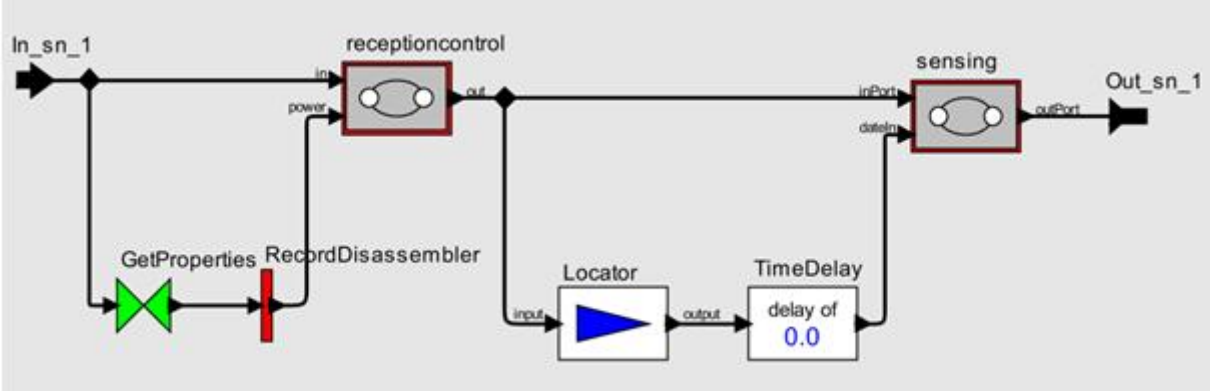

FIG. 3.6 – interface interne d'un capteur

Le comportement décrit par le *ModalModel* «*receptioncontrol*» est le même pour toutes les entités de systèmes qui sert à tester la puissance de signal qui lui arrive. Si cette dernière n'est pas inférieure à la puissance dont il a besoin alors il va transmettre la donnée. Le comportement de deuxième *ModalModel* change d'une entité à une autre selon la fonction à accomplir par l'entité.

3.6.3 ModalModel

Le ModalModel est un contrôleur de machines à états finis. On l'utilise pour décrire le comportement d'une entité de notre système à l'aide d'un automate à états finis.

La *figure 3.7* représente un exemple de comportement d'un capteur à l'aide d'un automate à états finis. Il comporte trois phases principales : d'abord une phase d'initialisation après la mise en marche, puis il attend l'arrivée des données par l'environnement. S'il reçoit les données alors il rentre dans la phase de transmission vers la BoxaNova. S'il tombe en panne avant la réception de données alors il rentre dans la phase d'attente de réparation. Une fois réparé, il revient au point d'attente de données.

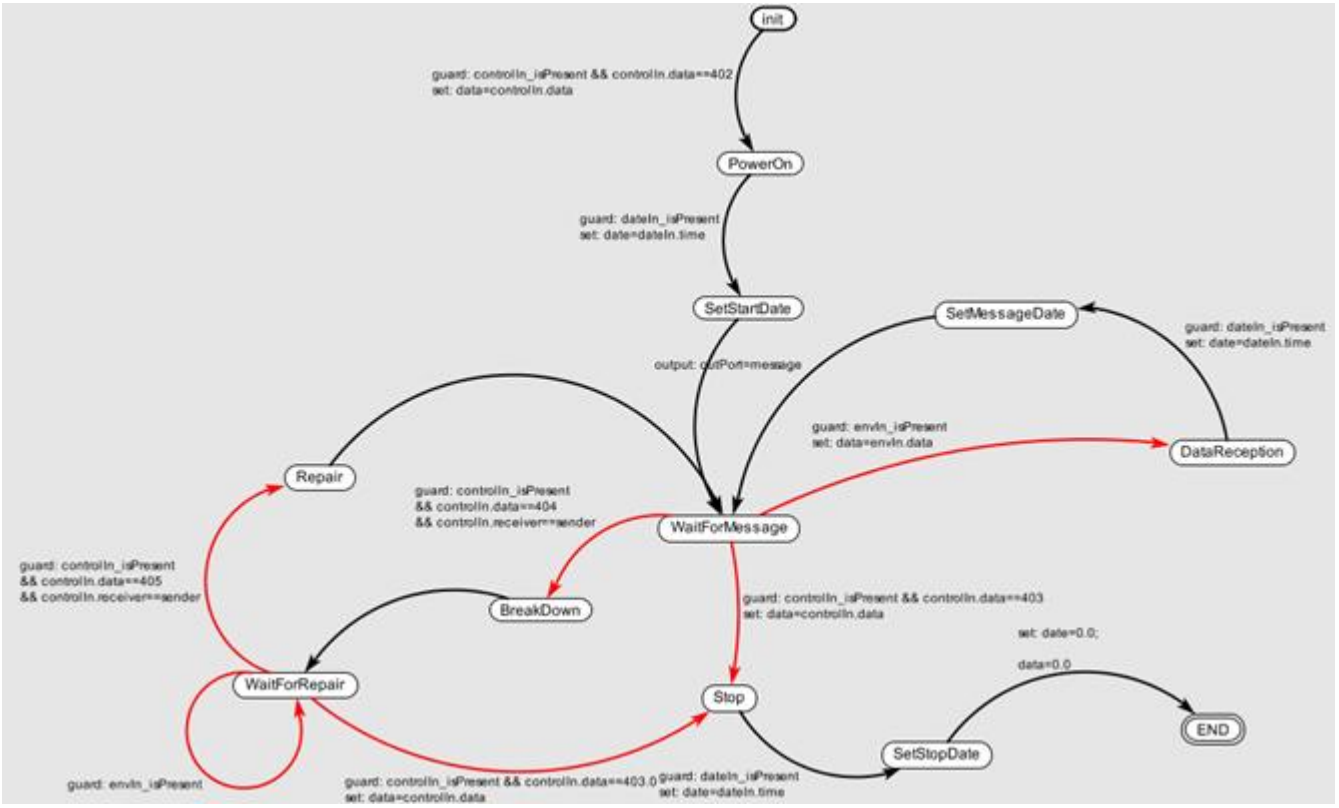

FIG. 3.7 – description du comportement d'un capteur

3.6.4 LinkVisualiser

Pour voir les liens entres les acteurs qui communiquent, on a choisi l'acteur *LinkVisualiser*. Il crée une ligne entre deux nœuds de communication qui sont à portée les uns des autres.

La *figure 3.8.a* represente l'interface de LinkVisualiser et la *figure 3.8.b* représente ses paramètres. Le paramètre ChannelName indique pour quel *WirelessChannel*, le *LinkVisualiser* fait l'animation. Le paramètre *sleepTime* exprime le temps entre le dessin d'une ligne et sa suppression.

FIG. 3.8 – LinkVisualiser

3.7 Scénario de simulation

Après avoir choisi les acteurs à utiliser (éléments et medium), on doit pouvoir simuler un système dans son environnement. Pour cela on a définis un acteur à part nommé «*scénario*». Il décrit les séquences d'exécution de comportement global de différentes entités. Il simule l'environnement en introduisant des valeurs entrées aux capteurs et aux capteurs-actionneurs. Le scénario lance le début et mis fin à la simulation, il injecte des pannes et il contrôle et gère les résultats, etc.

Pour la communication entre le scénario et le système on utilise des médiums supplémentaires qui sont parfaits (*sans atténuations, ni délais*). On ajoute des ports aux entités afin de communiquer avec les mediums qui sont ajoutés au système.

Les mediums à ajouter sont :

- ***controlMedium :*** c'est un medium qui permet d'envoyer un signal par le scenario pour mettre en marche, d'arrêter, de mettre en panne ou bien de réparer une entité du système.
- ***envMedium :*** c'est un medium qui sert à envoyer des données pour les capteurs et les capteurs-actionneur.
- ***checkMedium :*** c'est un medium spécial pour la BoxaNova afin que le scenario lui demande d'envoyer les dernières valeurs mesurées ou bien d'envoyer des commandes pour les capteurs et capteurs-actionneur.

Avec l'ajout de ces mediums et en ajoutant des ports aux entités, l'architecture interne des entités va changer. La *figure 3.9* illustre l'architecture interne de capteur de la *figure 3.6* après avoir effectué l'ajout. Le port *envin_ sn_ 1* communique avec *envMedium*, le port *control_ sn_ 1* communique avec *controlMedium*.

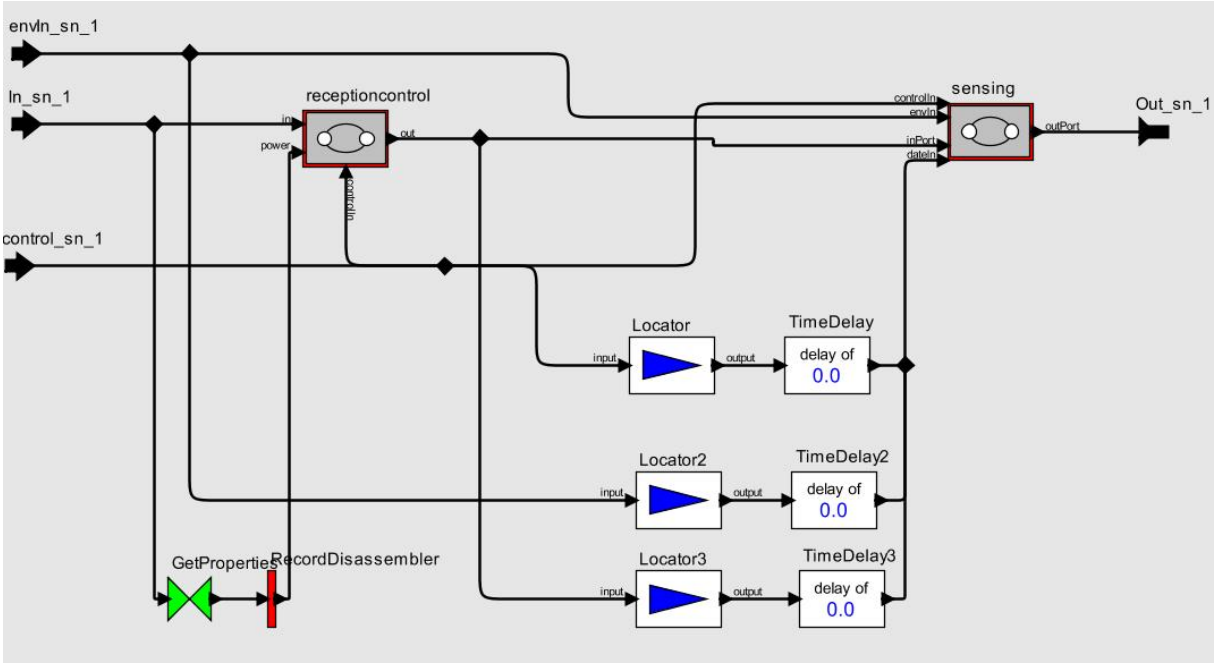

FIG. 3.9 – nouvelle interface interne d'un capteur

Les comportements des différents acteurs sont les suivants :

- **Scénario :** au départ, le scénario met en marche les différentes entités du système. Ensuite, il génère des mesures aléatoires et les envoie aux Capteurs et Capteurs-Actionneurs. Il peut également générer une panne pour une entité choisie et mettre fin à cette panne ensuite. Pour chaque intervalle de temps prédéfini, le scenario interroge la BoxaNova sur les dernières valeurs des différents messages reçus.
- **Capteur (*Sensor*)** : à la réception de la mise en marche, le capteur se met en attente des mesures provenant du scénario. Chaque valeur reçue engendrera un message qui comportera les champs suivants : Date, Data (la mesure à envoyer), Sender (*l'entité émettrice*), Receiver (*le destinataire*) et le group (*le réseau auquel appartienne l'entité*). Si une panne survient, le capteur se bloque et attend sa réparation. Après la réparation, le capteur reprend son fonctionnement normal jusqu'à son arrêt.
- **Repeater :** durant son fonctionnement normal, le répéteur transmet les messages reçus des capteurs/capteurs-actionneurs vers d'autres répéteurs si la portée est insuffisante pour atteindre la *BoxaNova*. Il transmet également les messages dans le sens inverse c-à-d. les commandes provenant de la *BoxaNova* vers les capteurs-actionneurs.
- **Concentrator :** il sauvegarde ses messages reçus dans sa mémoire et il fait la mise-à-jour à chaque réception du nouveau message. Le concentrateur peut envoyer les dernières valeurs obtenues suivant les demandes du scénario. Il peut également transmettre les messages de commande vers les capteurs-actionneurs.
- **BoxaNova :** elle se met en marche et s'arrête avec l'ensemble de ses composants. Lorsqu'elle est mise en panne, elle stoppe le fonctionnement de ses concentrateurs et de son routeur.

3.8 Démarche de modélisation et de transformation

Notre travail permet le passage d'un modèle des systèmes ubiquitaires présenté sous Kermeta vers le simulateur PtolemyII. Pour expliciter les étapes de notre solution, on les présente à l'aide d'un exemple. On choisi un système qui se compose de 7 capteurs, un capteur-actionneur, 3 répéteurs et une BoxaNova.

Etape 1

La *figure 3.10* illustre le système décrit ci-dessus (*présenté à la main*), seules les positions des composants sont indiquées. Les mediums de communication pour chaque port d'entrée et de port de sortie ne figurent pas, car on a utilisé qu'un seul medium de communication pour tous les composants de notre système. Dans le cas où on utilise plusieurs mediums, on doit préciser pour chaque port son medium, et pour chaque medium son nom ainsi que son type (*WIFI, ZigBee, Bluetooth, etc.*)

Le choix des équipements à utilisés et les endroits sur lesquels ils seront positionnés est le travail de l'ingénieur sur le terrain. Le choix varie d'un bâtiment à un autre selon la construction des murs et le besoin d'utilisateur.

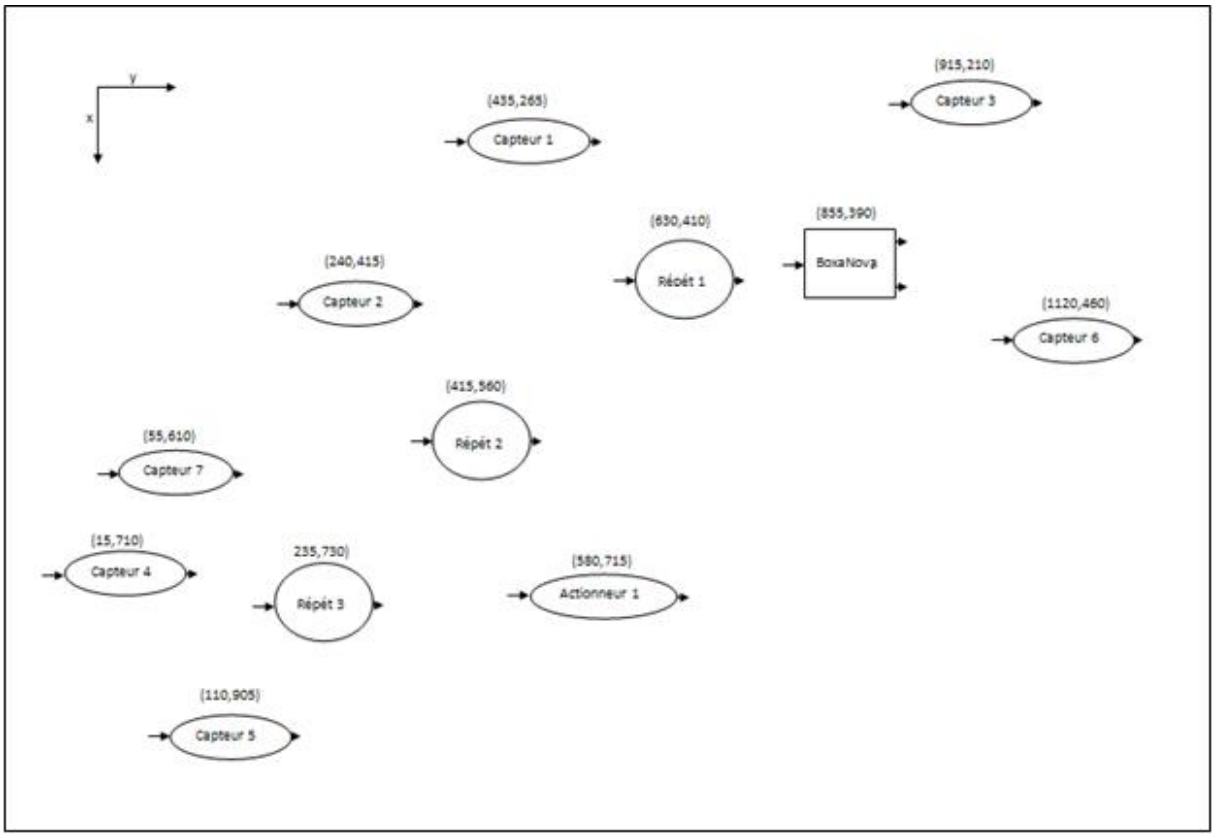

FIG. 3.10 – Représentation d'un exemple de Terra Nova Energy

Etape2

La *figure 3.11.a* représente le modèle qui traduit la description du système de la *figure 3.10*, ce modèle est réalisé avec le langage Kermeta sous l'extension XMI. Il est conforme au méta-modèle de *la figure 3.2*. On donne pour chaque entité ses coordonnées à l'aide de «*coordinates*», et on donne le radio-module qui précise les différentes interfaces de contact qui vont servir comme ports d'entrées ou de sorties. Pour chaque interface de contact on donne le medium avec lequel elle communique. Dans cet exemple on a le même medium pour toutes les interfaces.

Le passage de modèle (a) au modèle (b), est une transformation endogène qui se fait automatiquement à l'aide d'un programme. Cette transformation consiste à ajouter des ports et des mediums de communications qui seront utile pour le déroulement de scenario de simulation (*décrit ci-dessus*). Pour réaliser ce passage, on doit ajouter des interfaces de contact afin de permettre la communication avec les mediums ajouté au modèle. Pour faire ces ajouts, on parcoure le modèle (a) et on doit respecter les règles suivantes :

- Pour un capteur ou bien un capteur-actionneur de modele (a), on lui ajoute deux interfaces de contact. Une pour communiquer avec «*controlMedium*» et l'autre pour recevoir les données par le medium «*envMedium*».
- Pour un répéteur, on lui ajoute une interface de contact pour permettre sa communication avec «*controlMedium*».
- Pour la BoxaNova, on ajoute trois interfaces : deux pour communiquer avec «*checkMedium*», la troisième communique avec le «*controlMedium*». Pour les deux interfaces qui communiquent

avec «*checkMedium*», l'une pour recevoir la commande de scenario afin d'envoyer les dernières valeurs reçue, la deuxième interface permet d'envoyer une commande de la BoxaNova vers les capteurs et les capteurs-actionneurs.

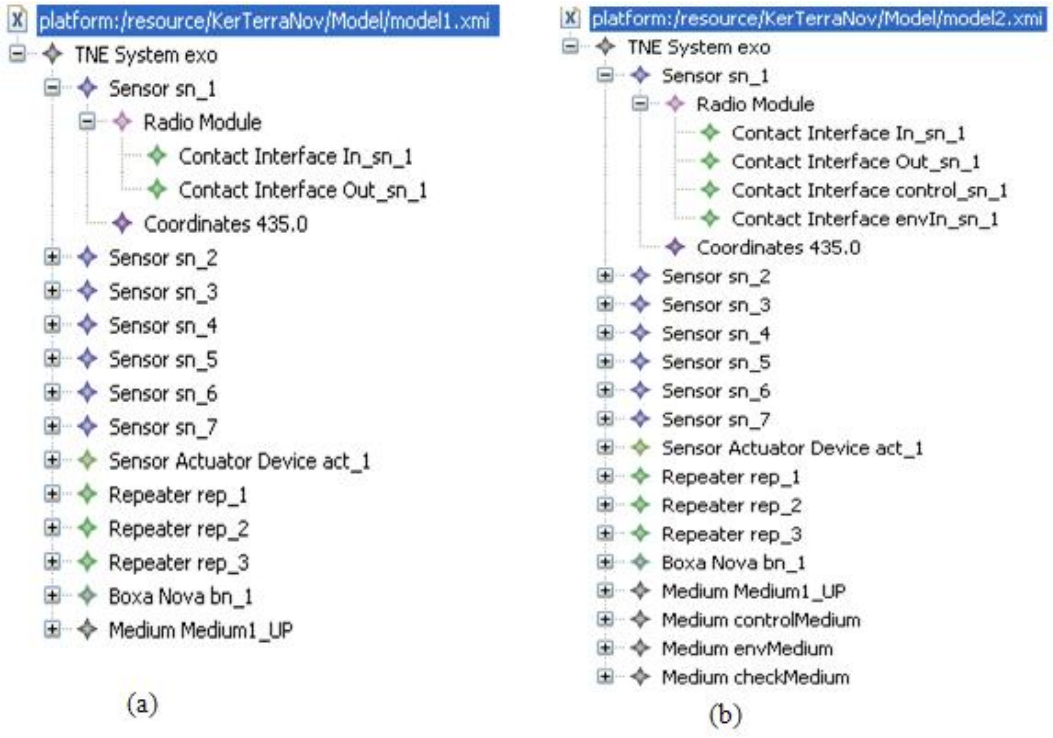

FIG. 3.11 – modèles de Terra Nova Energy sous Kermeta

Etape 3

Pour le passage de modèle (b) de la *figure 3.11* vers le simulateur PtolemyII on a utilisé une transformation exogène et verticale qui consiste à générer un texte en format XML. L'importation de ce dernier dans PtolemyII génère l'interface de la *figure 3.12*.

Pour les différents composants (*capteur, capteur-actionneur, routeur, répéteur, concentrateur, Boxa-Nova*) a été construit un programme. En parcourant le modèle (b) (*figure 3.11*), il suffit de faire appel au programme de composant en question pour le créer sur le simulateur automatiquement.

Afin de réaliser cette transformation on a remarqué quelques caractéristiques communes pour tous les acteurs de notre système :

- Le *programme 1* (*annexe*) a pour but de dessiner la forme de composant en récupérant ses coordonnées, sa hauteur et sa largeur ainsi que sa couleur et sa forme (*rectangle/ ellipse*).
- Le *programme 2* (*annexe*) permet de dessiner le medium de communication en donnant son nom, son type, sa vitesse de propagation, et ses coordonnés.
- Le *programme 3* décrit les ports des acteurs, en précisant avec quel medium ils communiquent et si ce sont des ports d'entrées ou de sorties
- Pour décrire le comportement d'un acteur on a besoin de décrire des états et des transitions, pour cela le *programme 4* a pour but de donner l'état et le *programme 5* pour donner la transition.

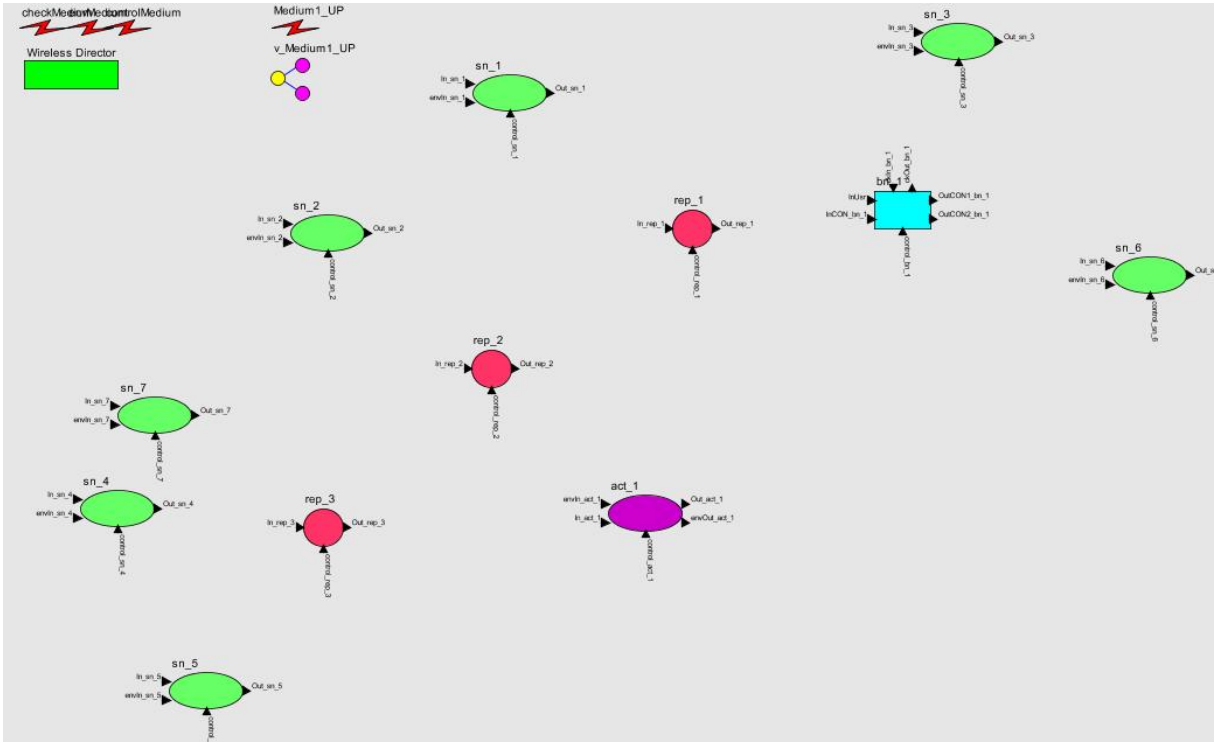

FIG. 3.12 – Interface de simulation d'un exemple de système Terra Nova Energy

En lançant la simulation pour l'interface de la (*figure 3.12*) on aura la (*figure 3.13*) qui montre des liens de communication entre les composants de système et les messages envoyés par les capteurs et les actionneurs.

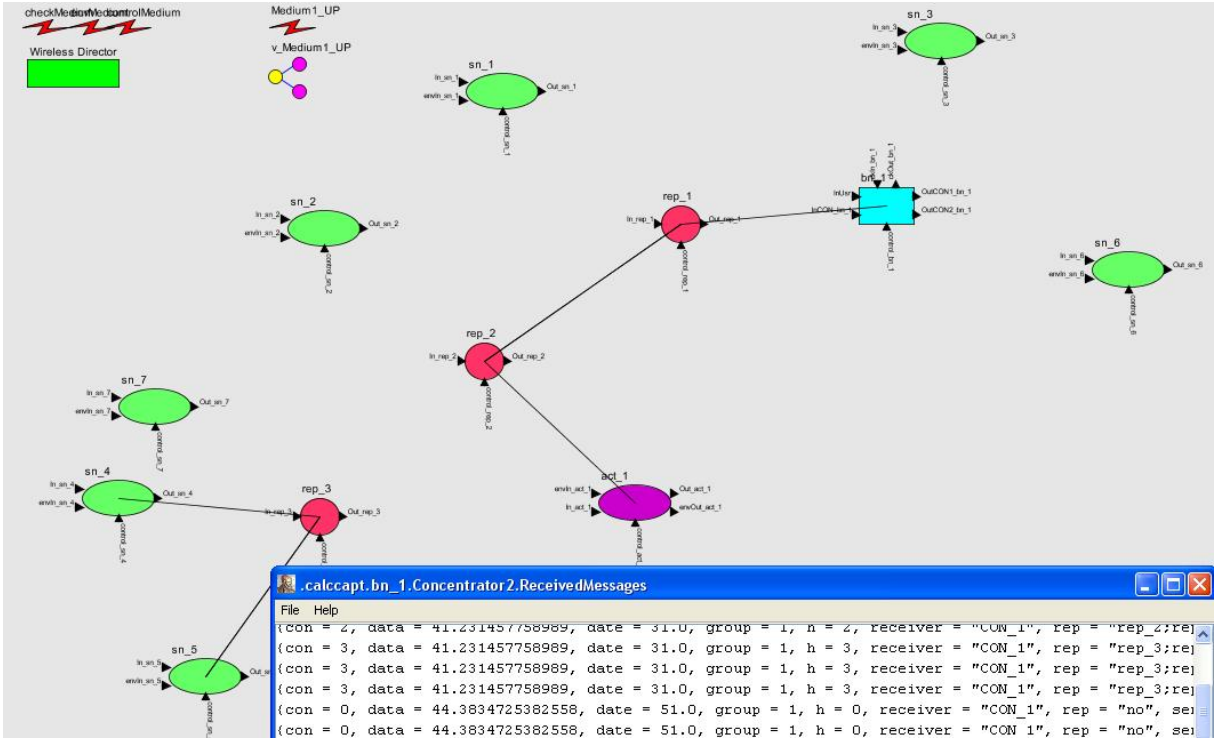

FIG. 3.13 – Lancement de la simulation

Les liens entres les acteurs de système indiquent la présence d'une communication au moment de la capture d'écran. Pour chaque capteur et capteur-actionneur, on affiche la donnée envoyée, la date d'envoi, les noms des répéteurs traversés par le message, le groupe de réseau auquel il appartient et le nombre de sauts effectués pour arriver à la BoxaNova.

3.9 Tests et résultats

Notre objectif de stage n'était pas de simuler mais de fournir l'outil de simulation pour les systèmes ubiquitaires. Pour voir l'utilité de l'interface à laquelle nous avons aboutis après la transformation de modèle vers le simulateur, on a effectué quelques testes :

- Lorsqu'un capteur ou un capteur-actionneur n'est pas à porté d'un répéteur ou bien de la BoxaNova, son message sera perdu.
- Lorsqu'un capteur ou un capteur-actionneur tombe en panne, il sera à "off" et son message sera absent sur la liste des messages reçue par la BoxaNova.
- Si un répéteur tombe en panne alors : si les capteurs et capteurs-actionneurs qui l'utilisent sont à porté d'un autre répéteur alors les messages trouvent un autre chemin. Sinon les capteurs et capteurs-actionneurs seront déconnectés du réseau.

3.10 Conclusion

Dans ce chapitre on a décrit les deux méta-modèles développés au sein de notre équipe, le méta-modèle pour les systèmes ubiquitaires et le méta-modèle dédié à *Terra Nova Energy*. Puis on a donné le contexte de notre travail ainsi que les trois étapes suivies pour réaliser le travail qui consiste au passage d'un modèle vers la simulation. Cette méthode a pour objectif majeur de faciliter la validation par simulation des systèmes ubiquitaires à grande échelle.

Conclusion et Perspectives

La complexité de conception des systèmes ubiquitaires augmente parallèlement, avec les besoins des utilisateurs qui ne cessent de s'accroître, et avec l'hétérogénéité des composants introduits sur le marché. Il est nécessaire pour les concepteurs de ces systèmes de trouver des méthodes pour les spécifier. Dans ce contexte, l'Ingénierie Dirigée par les Modèles, se présente comme une réponse pour relever ce défi. À partir d'un modèle exprimé à l'aide d'un langage spécifique au domaine, et en se basant sur les outils et les méthodes issus des travaux sur l'IDM, on peut générer, entre autres, d'autres modèles pour l'étude des systèmes par *simulation*.

Notre objectif dans ce mémoire était de faciliter la simulation d'un système ubiquitaire en passant, de manière automatisée, d'un modèle de ce dernier vers la simulation. Cette automatisation permet d'éviter la création manuelle des composants du système sur le simulateur, en utilisant les outils de l'IDM. Le but majeur de cette méthode est d'être capable, à terme, de simuler des systèmes ubiquitaires à grande échelle.

Pour atteindre cet objectif, on a commencé par donner un état de l'art qui comporte un aperçu de l'Ingénierie Dirigée par les Modèles, de la notion de système ubiquitaire et de des techniques de simulation. Les outils de simulation et de modélisation utilisés durant le stage ont été présentés, puis l'accent a été mis sur le méta-modèle des systèmes ubiquitaires et celui de Terra Nova Energy, enfin on a donné les étapes suivies pour la réalisation de travail demandé.

Notre travail a consisté à réfléchir sur la modélisation de systèmes ubiquitaires, conformément au méta-modèle disponible au laboratoire et à construire deux transformations permettant de passer d'un modèle décrivant un système donné à son modèle simulable

• La première transformation, de type *endogène* et *horizontale*, permet d'ajouter des éléments au modèle source. Les éléments ajoutés ont pour objectif de simuler l'environnement de notre système et de proposer également un scénario de simulation.

• La deuxième transformation consiste à générer un fichier en format XML, pour générer le modèle simulable, importable sur le simulateur PtolemyII. Cette transformation est de type *exogène* et *verticale* et est réalisée sur le modèle issu de la première transformation. Elle est automatisée à l'aide des programmes implémentés.

Le travail s'est achevé par la mise en œuvre de simulations de systèmes et une présentation des différents tests effectués sur les modèles simulables obtenus.

En perspectives, le travail pourrait être complété par l'ajout d'autres fonctionnalités comme :

✓Ajouter des obstacles et des murs sur l'interface de simulation avec de manière automatique, pousser la simulation le plus loin possible en augmentant le nombre de composants et de mediums de communication,

✓Améliorer le scenario de simulation afin qu'il soit mieux adapter pour le fonctionnement du système réel. Ajouter des fonctionnalités pour la simulation en consultants des experts dans le domaine,

✓Ajouter une palette de composante pour PtolemyII afin de permettre une utilisation ultérieure des composants réalisés.

Bibliographie

[1] Laurence Duchien, Cédric Dumoulin «IDM06, Actes des 2èmes journées sur l'Ingénierie Dirigée par les Modèles», Lille, juin 2006.

[2] Samba Diaw, Rédouane Lbath, Bernard Coulette, «DEtat de l'art sur le développement logiciel dirigé par les modèles», TSI-X,2008, Toulouse.

[3] Thomas Kühne, «What is a Model ?» In Jean Bezivin und Reiko Heckel (Hrsg.), Language ingineering for Model-Driven Software Development, Nr. 04101 der Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[4] Benoit Combemale, «Approche de méta-modélisation pour la simulation et la vérification de modèle, application à l'ingénierie des procédés»,Thèse, Toulouse, le 11 juillet 2008.

[5] Franck Fleurey, «Langage et méthode pour une ingénierie des modèles fiable», Thèse, Rennes 1, le 9 October 2006.

[6] Marjan Mernik, Jan Heering, Anthony M.Sloane, «When and how to develop domain specifie languages», ACM Computing Surveys,Vol. 37, No. 4, December 2005, pp. 316-344.

[7] Jesus Sanchez Cuadrado , Jesus Garcia Molina, «A Model-Based Approach to Families of Embedded Domain-Specific Languages», IEEE Computer Society 2009.

[8] Eric Piel, «Ordonnancement de systèmes parallèles temps-réel, de la modélisation à la mise en œuvre par l'ingénierie dirigée par les modèles», Thèse. Lille, le 14 décembre 2007.

[9] David Durand, «Gestion de la Qualité de Service dans les Applications Réparties sur Bus Middleware Orientés Objet Approche Dirigée par les Modèles», Thèse de Doctorat, université de Picardie Jules Verne, le 8 novembre 2006.

[10] Krzysztof Czarnecki, Simon Helsen, «Classification of Model Transformation Approches», OOPSLA'03 Morkshop on Generative Techniques in the Context of Model-Driven Architecture.

[11] Mark Weiser, «The computer for the 21st century», SIGMOBILE Mob. Comput. Commun. Rev., 3(3), 1999, S. 3-11.

[12] Amara Touil, Jean Vareille, Fred Lherminier, Philippe Le Parc, «Modeling and analysing ubiquitous systems using MDE approach». UBICOMM2010, Florence, Italie, du 25 au 30 octobre 2010.

[13]] Chantal Taconet, Zakia Kazi-Aoul, «Context-awareness and Model Driven Engineering» Illustration by an E-commerce application scenario.

[14] Xiaoli Yang, Qing Chen, Dorina C. Petri, Emil M. Petriu, «Internet-based Teleoperation of a Robot Manipulator for Education», Canada 2004.

[15] S. Otmane, N. Khezami, M. Mallem, «La Réalité Augmentée via Internet : Méthodes et Architecture Pour La Téléopération Collaborative», Université d'Evry Val d'Essonne.

[16] Weixuan Liu and E. J. Davison, «Servomechanism Controller Design of Web Handling Systems», IEEE Transactions on Control Systems Technology, vol. 11, no. 4, juillet 2003.

[17] Jean Vareille, Philippe Le Parc, Lionel Marcé, Stéphane Barré, ouu Abdeslam Mamoune, Joël Le Guen, Gilbert Floc'h, «Le contrôle réactif par Internet et son application aux machines de production».

[18] Jérémie Guiochet, «Maîtrise de la sécurité des systèmes de la robotique de service» TOULOUSE, 2003.

[19] JC Geoffry , G. Motet, «Maîtrise de la sécurité des systèmes de la robotique de service» , Paris, Inter Editions, pages 1 - 283, 2000.

[20] Jean-Louis Boimond, «Simulation, systèmes de production, réseaux de pétri, SIMAN-ARENA».

[21] Vincent Chapurlat, «vérification et validation des modèles de système complexes : application à la modélisation d'entreprise», Thèse, Université Montpellier II, Mars 2007.

[22] http ://www.ni.com/labview/f/

[23] http ://www.mathworks.com/products/simulink/

[24] http ://www-rocq.inria.fr/scicos/

[25] Philip Baldwin, Sanjeev Kohli, Edward A. Lee, Xiaojun Liu, Yang Zhao, «Modeling of Sensor Nets in PtolemyII», IPSN'04, April 26-27, 2004, Berkeley, California, USA.

[26] http ://www.limsi.fr/ jps/enseignement/tutoriels/archi/ARCHI.10.Transputer/ch10transputer.html

[27] Christopher Hylands, Edward Lee, Jie Liu, Xiaojun Liu, Stephen Neuendorffer, Yuhong Xiong, Yang Zhao, Haiyang Zheng, «Overview of the Ptolemy project», Université de Californie, 2 juillet 2003.

[28] Philip Baldwin, Sanjeev Kohli, Edward A. Lee, Xiaojun Liu, Yang Zhao, «Visualsense : visual modeling for wireless and sensor network systems», Version 5.0, July 15, 2005 UCB ERL Memorandum UCB/ERL M05/25.

[29] http ://www.eclipse.org/modeling/emf/ ?project=emf

[30] Zoé Drey, Cyril Faucher, Franck Fleurey, Vincent Mahé, Didier, Vojtisek, «Kermeta language» 3 November 2010.

ANNEXE

Programme1

```
operation desmodel(x:EDouble,y:EDouble,w:String,h:String,f:String,c:String):String is
do
var    st:String init String.new
      st.append("<property name="_location" class="ptolemy.kernel.util.Location"
value="["+x.toString+","+y.toString+"]">")
      st.append("\t\n</property>")
      st.append("\t\n<property name=\"_icon\" class=\"ptolemy.vergil.icon.EditorIcon\">")
      if f=="rectangle"
      then
      st.append("\t\n<property name=\"rectangle\"
class=\"ptolemy.vergil.kernel.attributes.RectangleAttribute\">")
      else
      st.append("\t\n<property name=\"ellipse\"
class=\"ptolemy.vergil.kernel.attributes.EllipseAttribute\">")
      end
      st.append("\t\n<property name=\"_location\" class=\"ptolemy.kernel.util.Location\" value=\"[-
2.0, 0.0]\">")
      st.append("\t\n</property>")
      st.append("\t\n<property name=\"width\" class=\"ptolemy.data.expr.Parameter\"
value=\""+w+"\">")
      st.append("\t\n</property>")
      st.append("\t\n   <property name=\"height\" class=\"ptolemy.data.expr.Parameter\"
value=\""+h+"\">")
      st.append("\t\n   </property>")
      st.append("\t\n   <property name=\"centered\" class=\"ptolemy.data.expr.Parameter\"
value=\"true\">")
      st.append("\t\n   </property>")
      st.append("\t\n   <property name=\"fillColor\" class=\"ptolemy.actor.gui.ColorAttribute\"
value=\""+c+"\">")
      st.append("\t\n   </property>")
      st.append("\t\n</property>")
      st.append("\t\n</property>")
    result:=st
end
```

Programme 2

```
operation medum (nom:String,val:String,propagation:String, factor:String,
location:String,Type:String):String is
do
var    st:String init String.new
      st.append("\t\n<entity name=\""+nom+"\"
class=\"ptolemy.domains.wireless.lib.PowerLossChannel\">")
      st.append("\t\n <property name=\"defaultProperties\" class=\"ptolemy.data.expr.Parameter\"
value=\"{"+val+"}\">")
      st.append("\t\n</property>")
      st.append("\t\n<property name=\"propagationSpeed\" class=\"ptolemy.data.expr.Parameter\"
value=\""+propagation+"\">")
      st.append("\t\n</property>")
      st.append("\t\n<property name=\"powerPropagationFactor\"
class=\"ptolemy.data.expr.Parameter\" value=\""+factor+"\">")
      st.append("\t\n</property>")
      st.append("\t\n<property name=\"_location\" class=\"ptolemy.kernel.util.Location\"
value=\"["+location+"]\">")
      st.append("\t\n</property>")
      st.append("\t\n<property name=\"Type\" class=\"ptolemy.data.expr.Parameter\"
value=\"&quot;"+Type+"&quot;\">")
      st.append("\t\n</property>")
      st.append("\t\n</entity>")
result:=st
end
```

Programme 4

```
operation etat(nom:String,val:String):String is
do
var    st:String init String.new
      st.append("\t\n   <entity name=\""+nom+"\" class=\"ptolemy.domains.modal.kernel.State\">")
      if nom=="init"
      then
      st.append("\t\n    <property name=\"isInitialState\" class=\"ptolemy.data.expr.Parameter\"
value=\"true\">")
      st.append("\t\n    </property>")
      end
      if nom=="END"
      then
      st.append("\t\n<property name=\"isFinalState\" class=\"ptolemy.data.expr.Parameter\"
value=\"true\">")
      st.append("\t\n</property>")
      end
      st.append("\t\n    <property name=\"_hideName\"
class=\"ptolemy.data.expr.SingletonParameter\" value=\"true\">")
      st.append("\t\n    </property>")
      st.append("\t\n    <property name=\"_controllerFactory\"
class=\"ptolemy.vergil.modal.modal.HierarchicalStateControllerFactory\">")
      st.append("\t\n    </property>")
      st.append("\t\n    <property name=\"_location\" class=\"ptolemy.kernel.util.Location\"
value=\"{"+val+"}\">")
      st.append("\t\n    </property>")
      st.append("\t\n  </entity>")
result:=st
end
```

Programme3

```
Operation
pmodel(q:String,name:String,locat:String,input:String,out:String,in:String,outran
:String,card:String,type:String):String is
do
var     st:String init String.new
        if q=="m" then
        st.append("\t\n<port name=\""+name+"\"
class=\"ptolemy.domains.modal.modal.ModalPort\">")
        else
        st.append("\t\n<port name=\""+name+"\"
class=\"ptolemy.domains.wireless.kernel.WirelessIOPort\">")
        end
        if input=="" then
        st.append("\t\n     <property name=\"input\"/>")
        else
        st.append("\t\n     <property name=\"output\"/>")
        end
        st.append("\t\n     <property name=\"outsideChannel\"
class=\"ptolemy.data.expr.StringParameter\" value=\""+out+"\">")
        st.append("\t\n     </property>")
        st.append("\t\n     <property name=\"insideChannel\"
class=\"ptolemy.data.expr.StringParameter\" value=\""+in+"\">")
        st.append("\t\n     </property>")
        if outtrans!="" then
        st.append("\t\n <property name=\"outsideTransmitProperties\"
class=\"ptolemy.data.expr.Parameter\" value=\""+outtrans+"\">")
        st.append("\t\n </property>")
        end
        if card!="" then
        st.append("\t\n     <property name=\"_cardinal\"
class=\"ptolemy.kernel.util.StringAttribute\" value=\""+card+"\">")
        st.append("\t\n     </property>")
        end
        if type!="" then
        st.append("\t\n<property name=\"_type\"
class=\"ptolemy.actor.TypeAttribute\" value=\""+type+"\">")
        st.append("\t\n</property>")
        end
        if locat!="" then
        st.append("\t\n     <property name=\"_location\"
class=\"ptolemy.kernel.util.Location\" value=\"["+locat+"]\">")
        st.append("\t\n     </property>")
        end
        st.append("\t\n   </port>")
result:=st
end
```

Programme 5

```
operation
transition(nom:String,garde:String,setaction:String,prem:String,non:String,out:Stri
ng):String is
   do
   var
    st:String init String_new
   st.append("\t\n    <relation name=\""+nom+"\"
class=\"ptolemy.domains.modal.kernel.Transition\">")
   if garde!=""
   then
    st.append("\t\n  <property name=\"guardExpression\"
class=\"ptolemy.kernel.util.StringAttribute\" value=\""+garde+"\">")
    st.append("\t\n  </property>")
   end
   if setaction!=""
   then
    st.append("\t\n    <property name=\"setActions\"
class=\"ptolemy.domains.modal.kernel.CommitActionsAttribute\"
value=\""+setaction+"\">")
    st.append("\t\n    </property>")
   end
   if prem!=""
   then
    st.append("\t\n    <property name=\"preemptive\"
class=\"ptolemy.data.expr.Parameter\" value=\"true\">")
    st.append("\t\n    </property>")
   end
   if non!=""
   then
   st.append("\t\n    <property name=\"nondeterministic\"
class=\"ptolemy.data.expr.Parameter\" value=\"true\">")
   st.append("\t\n    </property>")
   end
   if out!=""
   then
   st.append("\t\n       <property name=\"outputActions\"
class=\"ptolemy.domains.modal.kernel.OutputActionsAttribute\" value=\""+out+"\">")
   st.append("\t\n       </property>")
   end
   st.append("\t\n    </relation>")
    result:=st
end
```