Utilisation de SysML pour la modélisation des réseaux de capteurs

Nicolas Belloir*, Jean-Michel Bruel*, Natacha Hoang*, Congduc Pham*

*Université de Pau et des pays de l'Adour
LIUPPA, BP 1155, F-64013 Pau Cedex
{belloir,bruel,nhoang,cpham}@univ-pau.fr
http://liuppa.univ-pau.fr

Résumé. SysML est le nouveau langage de modélisation défini par l'OMG. Il peut être vu comme une extension d'UML destinée à la modélisation d'un large spectre de systèmes complexes. Son champ d'application est en ce sens plus large que celui d'UML mais sa filiation le rend tout particulièrement intéressant pour la modélisation de systèmes embarqués majoritairement composés de logiciel. Les logiciels déployés sur les réseaux de capteurs sans fil (WSN) sont un bon exemple de ce type d'application puisque la prise en compte de l'interaction forte entre le matériel et le logiciel inhérente à ce type de système est une condition importante pour une modélisation efficace. Dans cet article nous décrivons notre retour sur expérience concernant la modélisation d'un système utilisant des capteurs mobiles sans fil afin de mesurer les flux de personnes dans une ville. Dans cette étude, nous avons utilisé à la fois SysML pour la modélisation du système et UML pour la modélisation des parties logicielles. Nous présentons les points de recouvrements des deux langages d'une part, et nous en comparons les diagrammes statiques d'autre part.

1 Introduction

De nos jours, même la plus simple des applications informatiques est relativement complexe, principalement de par son caractère distribué, mobile et communiquant. La généralisation des architectures client/serveur, l'importance des notions de services, la coopération entre entités et les besoins de réactivité temps réel participent à imposer un environnement de développement rigoureux. Dans cet article, nous nous intéressons aux systèmes informatiques fortement contraints que constituent les réseaux de capteurs sans fil (*Wireless Sensors Networks* – WSN), technologie dont on peut consulter notamment l'état de l'art de Khemapech et al. (2005). Ces systèmes sont caractérisés par une forte interaction entre le matériel et le logiciel. Les composants internes des capteurs sont souvent propriétaires et constituent des composants sur étagère (*Commercial Off The Shelf* – COTS) dont la prise en compte nécessite une approche particulière. Les logiciels déployés sur les WSN doivent pouvoir prendre en charge l'interaction forte entre le matériel et le logiciel inhérente à ce type de système.

Spécialisée dans le développement d'application basée composants (*Component-Based Software Engineering* – CBSE), technologie présentée dans l'ouvrage de Szyperski et al.

(2002), et dans les réseaux, notre équipe cherche à déterminer une approche rigoureuse de développement pour les WSN. Pour cela, nous nous basons sur notre expérience de l'écriture de profils et de métamodèles, que nous appliquons aux dernières avancées en terme de langages de modélisation de systèmes. En effet, même si le rapprochement des langages de modélisation (comme UML) et des langages de description d'architecture (ADL) ont vu l'intégration dans la version 2.0 d'UML, spécifiée dans OMG (2005), de concepts clés comme ceux de ports ou de composites, force est de constater les manques de tels langages généralistes pour modéliser les systèmes complexes du type WSN, pour lesquels jusqu'à maintenant, et à notre connaissance, ont été utilisées des approches plutôt formelles telles que dans Sachdeva et al. (2005).

SysML est le nouveau langage de modélisation défini par l'OMG (2007b). Il peut être vu comme une extension d'UML destiné à la modélisation d'un large spectre de systèmes complexes. Après en avoir étudié les apports, nous avons été confrontés à un certain nombre d'interrogations. Dans cet article nous décrivons notre retour sur expérience concernant la modélisation d'un cas réel (un système utilisant des capteurs mobiles sans fil afin de mesurer les flots de personnes dans une ville, financé par la communauté d'agglomération de la ville de Pau). Dans cette étude, nous avons utilisé à la fois SysML pour la modélisation du système et UML pour la modélisation des parties logicielles. Nous discutons des points de recouvrement des deux langages d'une part, et nous en comparons les aspects statiques d'autre part.

2 Modélisation des systèmes embarqués communiquants

Nous nous intéressons ici à un type de système embarqué communiquant particulier : les réseaux de capteurs sans fil. Nous les présentons dans un premier temps puis nous expliquons pourquoi UML ne suffit pas pour modéliser ce type de système.

2.1 Les réseaux de capteurs sans fils : un système très complexe

Un capteur est un petit appareil autonome capable d'effectuer des mesures simples sur son environnement immédiat. L'utilisation de ces capteurs n'a rien d'une nouveauté, ceux-ci sont utilisés depuis longtemps dans des domaines comme l'aéronautique ou l'automobile. Ce qui est novateur, c'est la possibilité pour ces capteurs de communiquer de manière radio (réseaux sans fils de type WiFi) avec d'autres capteurs proches (quelques mètres) et pour certains d'embarquer de la capacité de traitement (processeur) et de la mémoire. On peut ainsi constituer un réseau de capteurs qui collaborent sur une étendue assez vaste. Ces réseaux de capteurs soulèvent un intérêt grandissant de la part des industriels ou d'organisations civiles où la surveillance et la reconnaissance de phénomène physique sont une priorité. Par exemple, ces capteurs mis en réseau peuvent surveiller une zone délimitée pour détecter soit l'apparition d'un phénomène donné (apparition de vibrations, déplacement linéaire...) soit mesurer un état physique comme la température (détection des incendies en forêts) ou la pression. Dans beaucoup de scénarios de gestion de crise (séismes, inondations,...) ces réseaux de capteurs peuvent permettre une meilleure connaissance du terrain afin d'optimiser l'organisation des secours, ou bien même renseigner précisément les scientifiques sur les causes d'un phénomène physique particulier. Il est aussi envisageable d'intégrer les aspects multimédia et nomadisme dans certaines applications mobiles mélangeant données et images. On le voit, les applications

possibles sont extrêmement nombreuses avec un impact fort sur un grand nombre d'applications civiles et des nouveaux effets sociétaux certains.

Ces réseaux de capteurs posent un certain nombre de défis scientifiques intéressants pour la communauté de recherche. Par exemple, l'organisation de ces capteurs en réseaux soulève des problèmes bien connus, mais qui restent extrêmement ardus, de routage (détermination du chemin optimal entre 2 points du réseau), de communication et d'architecture logicielle. Les scénarios usuels d'utilisation envisagent plusieurs milliers de capteurs que l'on pourra disperser pour surveiller des zones sensibles. Le facteur de résistance à l'échelle sera donc crucial. S'ajoute à ces difficultés le fait que les capteurs possèdent des ressources très limitées en terme de puissance de calcul, mais aussi en terme d'autonomie de fonctionnement puisque dans la plupart des scénarios de déploiement, les capteurs fonctionneront avec de petites batteries. Cette limitation des ressources rend nécessaire une certaine forme de coopération à grande échelle où les interactions entre capteurs peuvent être extrêmement complexes. En effet, outre les problèmes de dissémination ou de récupération des données, la réalisation d'un service complexe dans un réseau de capteurs doit pouvoir être effectuée grâce à une composition de services plus simples et donc à une forme de collaboration "intelligente" des capteurs. La reconfiguration et l'administration à distance, c'est-à-dire la gestion de ces capteurs, seront également des propriétés souhaitables qui seront indispensables dans un proche avenir pour optimiser les ressources et permettre la réutilisation de l'infrastructure déployée. Toutefois les architectures logicielles actuelles ne savent pas, ou mal, intégrer des éléments autonomes et mobiles comme le sont les capteurs.

2.2 Modélisation de nouvelles fonctionnalités et limitation d'UML

Les travaux dans le domaine du génie logiciel ont prouvé le besoin de développer les applications informatiques de manière modulable et faiblement couplée. L'ingénierie logicielle basée composant a ainsi apporté d'importantes contributions, offrant des méthodes, des concepts et des supports technologiques qui permettent ce type de développement. Dans ce contexte, de nombreuses nouvelles fonctionnalités peuvent alors être envisagées. Par exemple, la reconfiguration dynamique qui est une composante primordiale de la modularité est alors envisageable. En effet, cette opération permet de remplacer un composant par un autre dans une application en cours d'exécution. Cette action peut-être causée par la nécessité de substituer un composant mal implémenté c'est-à-dire ne réalisant pas les fonctionnalités prévues ou les réalisant de manière incorrecte, ou encore d'ajouter à ce composant de nouvelles fonctionnalités. Dans le contexte des réseaux de capteurs, il peut s'avérer pertinent, par exemple, de changer les données qu'un capteur devait initialement collecter sur son environnement ou tout simplement de reconfigurer le réseau lorsqu'un capteur n'a plus assez de batterie pour fonctionner. La reconfiguration dynamique devient alors une caractéristique importante des applications déployées sur ce type de réseau.

Normalisé par l'OMG, UML est le langage graphique le plus utilisé pour modéliser les divers aspects d'un système d'information. Par contre, dans le contexte de l'ingénierie système, son pouvoir d'expression est plus limité. En effet, certains concepts spécifiques à ce domaine ne peuvent être spécifiés simplement avec UML. Par exemple, le fait qu'il existe des paramètres dont un changement de valeur entraînerait un fonctionnement différent du système ne peut être modélisé avec UML. De plus, le lien fort entre le logiciel et le matériel ne trouve pas en UML de moyen satisfaisant d'expression. Dans les systèmes qui nous intéressent, il faut être capable

d'exprimer les contraintes portant sur la batterie, ou bien encore la faible quantité de ressources disponibles. Par exemple, si le niveau de la batterie devient limite, il faut pouvoir spécifier une action comme envoyer un signal aux autres capteurs leur demandant de se reconfigurer pour palier cette défection. Ces limitations ont fait l'objet de nombreux travaux de recherche. Notons actuellement les travaux portant sur le profil UML Marte (OMG (2007a)) qui est dédié aux systèmes temps réel et embarqués ainsi que les travaux portant sur le langage de description d'architecture AADL (Feiler et al. (2006)).

Pour remédier aux limitations d'UML identifiées précédemment, l'INCOSE[1], aidé par l'OMG, a donc réfléchi sur des améliorations d'UML 2 afin de proposer un langage de modélisation plus approprié à l'ingénierie système. Nous présentons SysML plus en détail dans la section suivante.

3 Présentation de SysML

SysML est le nouveau langage de modélisation spécifié par l'OMG. Il s'agit en fait d'un profil UML et il hérite donc des caractéristiques d'UML, ce qui est à la fois un atout et une faiblesse comme nous le verrons dans la section 5. Il s'agit d'un langage de modélisation graphique dont la sémantique semi-formelle est définie dans OMG (2007b) - il est entendu que par "sémantique", nous nous référons à Harel et Rumpe (2004) qui notent que malgré le manque de sémantique formelle et les critiques bien connues sur le sujet, les langages diagrammatiques ont prouvé leur importance dans le développement système et logiciel. SysML est défini comme un langage de modélisation pour l'ingénierie système capable d'offrir un support pour la modélisation de multiples processus et méthodes. Néanmoins, comme explicité dans le document de spécification, chaque méthodologie peut imposer des contraintes additionnelles sur la manière dont un élément de construction ou un type de diagramme donné peut être utilisé. Cela sous-entend qu'à cause du nombre élevé de champs couverts par l'ingénierie système, une approche interdisciplinaire est difficile à obtenir. De plus, les processus d'ingénierie, tant pour l'ingénierie logicielle que système, ont évolué indépendamment chacun de leur coté comme l'explique Boehm (2006). Dans ce contexte, SysML semble être en mesure de devenir un support permettant de rapprocher ces deux familles d'ingénierie. Dans cet esprit, nous avons à travers notre étude de cas, essayé d'évaluer ce rapprochement.

Les concepteurs de SysML ont travaillé d'une part, à limiter ou adapter les concepts trop proches de l'ingénierie logicielle, et d'autre part à simplifier la notation UML originale en réduisant notamment le nombre de types de diagramme disponible, afin d'en simplifier l'utilisation. Ainsi, la figure 1 tirée de Friedenthal et al. (2007) montre les diagrammes fournis par SysML. On note l'apparition de deux nouveaux diagrammes. Le *Requirement Diagram* (diagramme des exigences) a pour rôle de spécifier les besoins de l'application. Un des points intéressants de ce type de diagramme est qu'il permet de relier les exigences avec les diagrammes répondant à ces exigences comme nous le verrons sur l'exemple de la figure 2 présenté à la section 4.2. Le *Parametric Diagram* (diagramme paramétrique) est le second type de diagramme entièrement nouveau. Il permet de spécifier des expressions mathématiques entre les éléments des modèles. Parmi les diagrammes conservés d'UML, l'*Activity Diagram* (diagramme d'activité) a été légèrement modifié. Par contre, les *Class Diagram* (diagramme de

[1]INCOSE : International Council on Systems Engineering : `http://www.incose.org/`

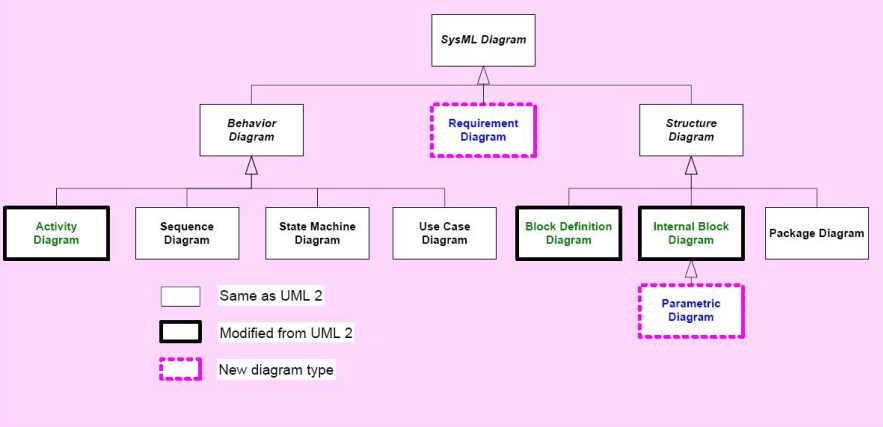

FIG. 1 – *Taxonomie des diagrammes SysML*

classe) et *Composite Diagram* (diagramme composite) d'UML ont été renommés et modifiés plus en profondeur à travers le concept de *Block*, permettant d'exprimer tout élément structurel d'un système. Nous nous intéresserons d'ailleurs plus en détail aux diagrammes structurels d'UML et de SysML dans la section 5.

Enfin, SysML se veut, à la manière d'UML, une norme évolutive. La version 1.0 a été adoptée le 19 septembre 2007 et déjà le comité travaillant sur SysML à l'OMG propose des pistes d'amélioration pour passer à la version 1.1. Cette approche incrémentale, même si elle implique une réactivité forte à la fois des éditeurs d'environnement de modélisation et des utilisateurs, a montré son succès avec la famille des langages UML et l'avènement de UML 2.

4 Etude de cas

Dans cette section, nous présentons d'une part le projet Sicop dans le cadre duquel nous avons mené notre étude de cas, et d'autre part un ensemble d'éléments de modélisation mettant en œuvre le langage SysML.

4.1 Le projet Sicop

L'objectif du projet Sicop[2], financé par la communauté d'agglomération de Pau Pyrénées, est de déterminer si la technologie des WSN est adéquate pour l'étude des mouvements urbains sur une période donnée. Dans ce contexte, les WSN permettent d'automatiser les études statistiques mais également d'atteindre un haut degré de pertinence. Dans ce scénario, des capteurs individuels sont distribués à une population cible et volontaire, qui les portera lors de ses déplacements urbains. Ainsi les déplacements sont observés et enregistrés par le système. Nous appelons ces capteurs des capteurs *mobiles*. Les déplacements seront enregistrés grâce

[2]Sicop : Sensor in the City of Pau (`http://liuppa.univ-pau.fr/ALCOOL/article.php3?id_article=13`)

au déploiement au préalable de capteurs en des endroits (extérieurs ou intérieurs) prédéfinis de la ville. Ce sont les capteurs dits *fixes*. Un capteur mobile porté par une personne pourra communiquer avec un capteur fixe lorsque cette personne passera à proximité de celui-ci. Les informations collectées par les capteurs fixes pourront alors être agrégées sur un réseau de type Internet (tel que l'infrastructure Pau Broadband Country[3]) par le biais de ponts WiFi pour être enregistrées dans une base de données. Comme il est difficile de mettre un point d'agrégation (les ponts WiFi par exemple) pour chaque capteur fixe déployé, les informations d'un capteur mobile concernant les différents capteurs fixes rencontrés pourront être également sauvegardées puis envoyées en différé lorsque ce capteur mobile passera à proximité d'un capteur fixe proche d'un point d'agrégation. Les différents déplacements d'un capteur mobile, représentés par la suite des capteurs fixes qu'il a rencontrés dans le temps, seront ainsi enregistrés, permettant aux scientifiques concernés d'étudier les différents déplacements urbains d'une population test. Il est aussi envisageable que les capteurs mobiles puissent communiquer entre eux afin de remplir, si besoin est, le rôle de station relais pour l'échange d'informations ou bien pour augmenter l'étude avec des données concernant les interactions/corrélations entre les déplacements. Les possibilités de reconfiguration des capteurs mobiles et fixes permettront de pouvoir réutiliser l'infrastructure déployée pour d'autres types d'application.

4.2 Eléments de modélisation

Dans un projet système classique des ingénieurs spécialistes de divers domaines technologiques (électronique, hydraulique, logiciel ...) et de divers domaines métiers (aéronautique, automobile ...) sont amenés à interagir. C'est là la cible de SysML. Cependant, SysML nous semble particulièrement intéressant également dans le cas où un ingénieur logiciel est amené à travailler dans des systèmes principalement composés de logiciel comme par exemple en informatique pervasive, dont le domaine est présenté par Satyanarayanan (2001). C'est cette approche que nous avons choisie pour mener notre investigation.

Ce cas d'étude est très orienté logiciel et peut paraître peu complexe pour un ingénieur système. Néanmoins, il nous semble particulièrement adapté pour explorer la frontière entre UML et SysML d'une part, et d'autre part il est révélateur des problèmes à venir avec l'avènement de l'informatique pervasive dans laquelle les systèmes d'informations vont devoir de plus en plus fonctionner sur, et/ou utiliser, des environnements contraints. Dans notre étude, il est nécessaire de prendre en compte la sévérité des contraintes inhérentes au support matériel que sont les capteurs. Par exemple, de par leur faible autonomie (par rapport à d'autres systèmes mobiles), il est impératif de limiter au maximum les traitements impliquant une forte dépense d'énergie telles que les communications.

Le traitement de ces contraintes peut s'envisager à plusieurs niveaux : bien évidemment elles doivent être considérées et spécifiées au niveau de la modélisation du système dans son ensemble. Mais elles doivent également être traitées au niveau de la conception logicielle puisqu'elles ont un impact sur la manière dont le logiciel doit fonctionner. Ce type d'entrelacement entre le logiciel et son support matériel est naturel pour un ingénieur système alors qu'il l'est habituellement moins pour un ingénieur logiciel. Ce dernier devra donc avec SysML savoir précisément jusqu'à quel niveau de raffinement il doit modéliser son système. Par exemple,

[3]Le Pau Broadband Country `http://paubc.info` est un projet visant à tester la viabilité d'un réseau à très haut débit dans une agglomération.

SysML permet de modéliser une architecture matérielle, mais on peut se demander s'il est pertinent de le faire dans le cas où le projet à modéliser réutilise du matériel existant.

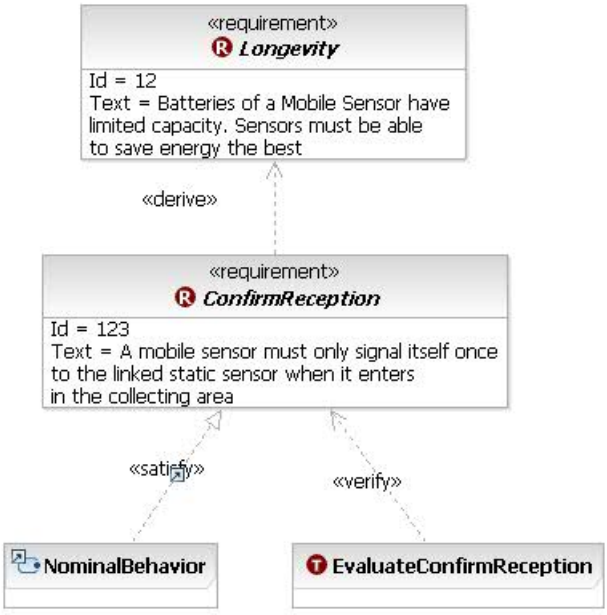

FIG. 2 – *Exemple de diagramme SysML d'expression des besoins*

Grâce aux bases UML de SysML les ingénieurs logiciels peuvent se familiariser facilement avec la plupart des diagrammes SysML. Seuls les diagrammes paramétriques ou les diagrammes des besoins sont réellement nouveaux (OMG, 2007b, p.11). En fait, si l'on regarde la pratique, en ingénierie logicielle les besoins sont exprimés la plupart du temps dans d'autres langages qu'UML. L'utilisation des diagrammes de cas d'utilisation, de classe, ou encore de séquence tôt dans la phase d'analyse permet de vérifier la bonne compréhension des besoins plutôt que de les spécifier. De plus, en UML les besoins ne sont pas reliés aux diagrammes correspondants. Cela implique donc pour l'ingénieur logiciel un effort d'attention élevé pour s'assurer que tous les besoins se retrouvent bien dans la spécification finale UML puisqu'il n'a aucune traçabilité à sa disposition. Cependant, les diagrammes des besoins en SysML ne sont pas non plus exempts de reproche. S'ils permettent effectivement d'introduire une traçabilité entre un besoin et le (ou les) diagramme(s) réalisant ce besoin, ils consistent en fait simplement en un élément structurel contenant une description textuelle du besoin. On peut également faire des dérivations de besoins ou bien créer une hiérarchie de besoins, mais cela semble léger pour répondre à la demande forte d'expression bien formalisée - et automatisée - des besoins dans les langages de modélisation. La figure 2 montre un diagramme d'expression des besoins. Celui-ci spécifie des propriétés liées au besoin d'économie d'énergie. Le bloc *Longevity* est un besoin de haut niveau. Le bloc *ConfirmReception* est beaucoup plus précis. La relation <<derive>> montre que le second besoin découle directement du premier. Dans

la partie basse de la figure, on trouve le diagramme d'état *NominalBehavior* qui satisfait le besoin *ConfirmReception* (lien de type <<satisfy>>) ainsi que le cas de test *EvaluateConfirmReception* qui vérifiera la bonne implémentation de ce besoin.

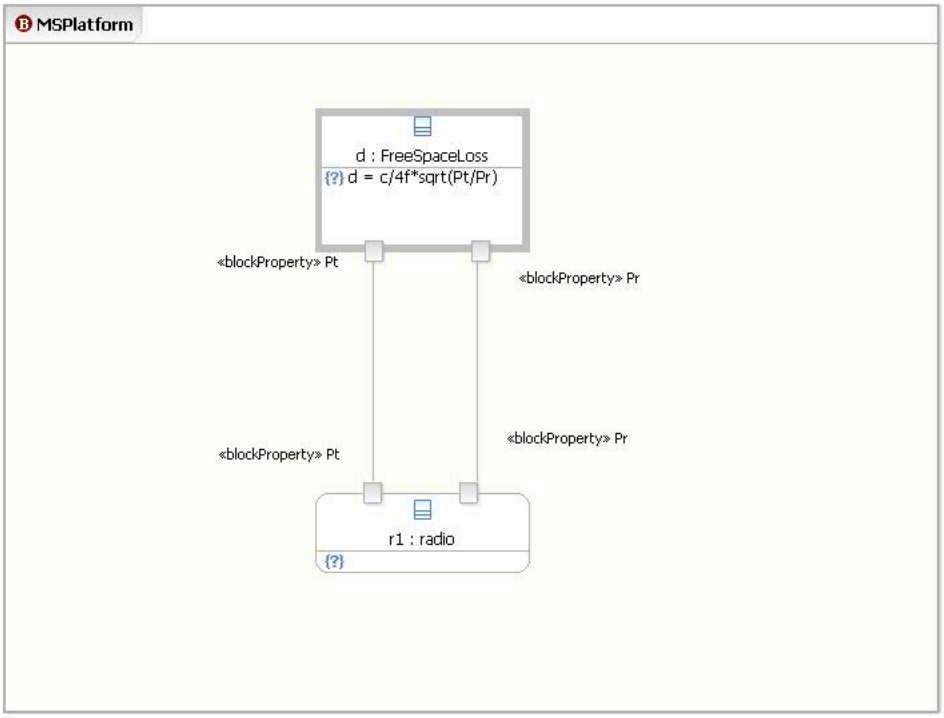

FIG. 3 – *Exemple de diagramme paramétrique*

Les diagrammes paramétriques représentent la seconde nouveauté de SysML. Ils offrent la possibilité d'exprimer des propriétés mathématiques entre différents éléments d'un modèle. En cela ils offrent un apport considérable par rapport à UML où ce type de notation n'était possible qu'à travers des contraintes textuelles ou exprimées dans le langage formel OCL. Par exemple, dans le cadre de notre étude, ces diagrammes nous ont permis de spécifier des contraintes portant sur la portée d'une émission WiFi. En effet, nous spécifions le fait qu'un capteur mobile ne rentre en communication avec un capteur fixe que si celui-ci est à une portée d'au moins 20% de sa puissance d'émission. Pour cela, il nous faut calculer l'affaiblissement en espace libre, chose que nous spécifions sur la figure 3 : l'équation, tirée de Stallings (2005), est spécifiée à l'aide du bloc de contrainte *FreeSpaceLoss*. Les paramètres *Pt* et *Pr* sont transmis par le bloc radio.

Pour l'expression des contraintes, la possibilité d'utiliser OCL est préservée bien que non explicitement exprimée dans la spécification (à part dans l'utilisation d'un bloc particulier appelé *constraint block* et utilisable uniquement sur les diagrammes de définition de bloc ou les diagrammes de package, ce qui est limitatif).

De la même manière qu'UML, SysML n'est qu'un langage de modélisation. Cela signifie qu'il n'y a pas de méthode associée spécifiquement au langage. On peut toutefois réutiliser des règles de bonne utilisation naturelles. Ainsi, une fois les besoins établis, il semble opportun de se consacrer à l'établissement des cas d'utilisation. En SysML, dans ce type de diagramme, même si la sémantique est identique à celle d'UML, on note dans la pratique que les acteurs du système seront la plupart du temps des systèmes externes plutôt que des acteurs humains. Dans notre étude de cas, les acteurs ne sont pas seulement les personnes portant les capteurs mobiles ou encore l'opérateur récoltant les données ; les différents types de capteurs (mobiles, fixes reliés au réseau ou fixes non reliés au réseau) sont également des acteurs. Une fois les cas d'utilisation réalisés, il est possible de spécifier les diagrammes de bloc et d'interaction afin d'avoir une première vue du système.

Le diagramme de définition de bloc et le diagramme de bloc interne qui correspondent, respectivement au diagramme de classe et au diagramme composite en UML, peuvent étrangement se révéler difficile à créer pour un ingénieur logiciel. En effet, le point difficile à notre avis concerne le niveau de profondeur de la modélisation matérielle qui doit être mise en œuvre. Par exemple, un même bloc peut être considéré comme un composant électronique ou comme un composant logiciel. Une interaction entre deux blocs peut être vue comme une liaison électrique, comme un lien logiciel portant des messages binaires ou comme un échange de messages entre deux composants logiciels. Pour preuve, dans notre étude, la modélisation des échanges entre deux capteurs peut être traitée à bas niveau (trame réseaux) ou encore d'un point de vue haut niveau (échange de messages). On peut encore vouloir exprimer des contraintes concernant la puissance ou le rythme des émissions radio pour économiser la batterie. Ainsi, il convient de bien cibler ce que l'on veut précisément modéliser.

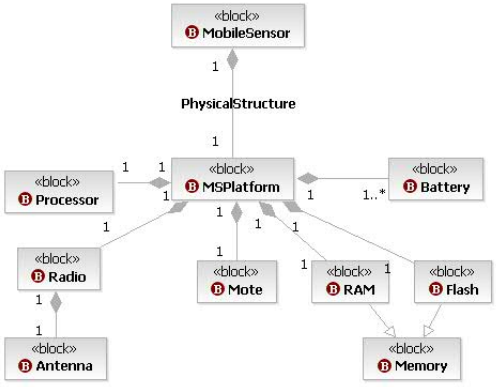

FIG. 4 – *Diagramme de définition de bloc*

La figure 4 présente le diagramme de définition de bloc spécifiant une vue haut niveau d'un capteur mobile. On peut remarquer facilement une imprécision du langage. En effet, le bloc *mote*, qui représente la carte mère de la partie calculatoire du capteur mobile est lié au bloc *platform* via une composition de même type que celle qui lie le bloc *battery* à *platform*. Or, la composition entre *platform* et *battery* vient du lien physique entre les deux, comme pour *mote*,

mais alors qu'on peut imaginer changer la batterie, il est impossible de changer la carte mère. Il y a donc un manque au niveau de la spécification. Faut-il alors supprimer *mote* et l'inclure dans *platform* ? Cela aurait pour conséquence de rendre cet élément moins visible. Se pose également le problème de la représentation interne de *platform* (voir la figure 5) car dans ce cas le *mote* devient central, ce qu'il n'était pas dans le diagramme de définition de bloc. Une heuristique pourrait alors en découler : si un bloc est central, alors il faut l'inclure dans le bloc principal. L'autre solution serait d'ajouter des propriétés sur les relations de composition, telles que la non séparabilité, comme nous l'avons proposé dans Belloir (2004).

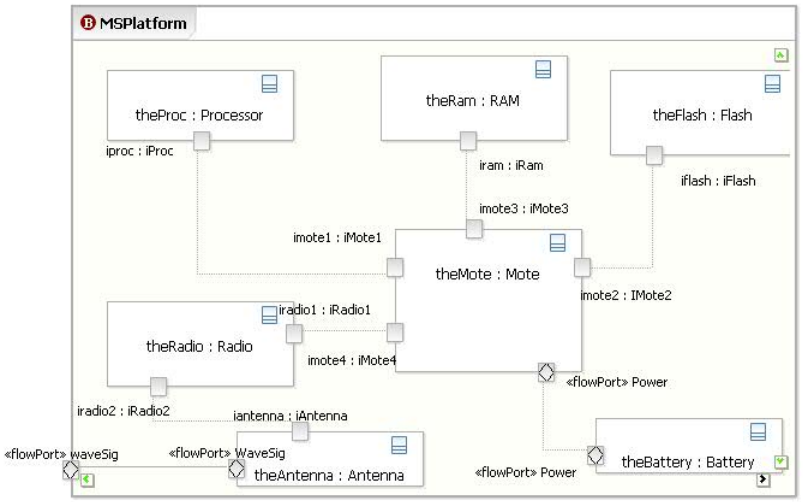

FIG. 5 – *diagramme de représentation interne de bloc*

5 Discussion

Dans cette section, nous allons tenter d'évoquer les avantages et les inconvénients de l'utilisation de SysML pour le développement des systèmes du type WSN. Dans le cadre de cet article, nous limiterons essentiellement nos réflexions aux diagrammes structuraux.

5.1 De la question de la sémantique

Si l'on regarde l'évolution des différentes versions d'UML, on constate qu'un effort récurrent a été porté sur l'amélioration du pouvoir d'expression du langage parallèlement à une meilleure définition de la sémantique associée aux éléments du langage. Cependant ces deux axes sont parfois antinomiques. Pour s'en convaincre, prenons le cas de la représentation d'un élément contenant un autre élément. En UML 1.x, le concepteur devait exprimer cette contenance à l'aide des relations d'agrégation et de composition, relations dont Barbier et al. (2003) ont montré l'imprécision sémantique. UML 2 a révolutionné les moyens de représenter cette

contenance grâce au concept de classe structurée, bien plus visuel. Cependant, leur sémantique, au lieu d'améliorer et de simplifier les choses, ne fait qu'accentuer le doute puisque, non seulement les relations de composition et d'agrégation sont toujours disponibles, mais en plus leur sémantique reste imprécise comme montré par Belloir (2004). Après expérimentation de SysML, il s'avère que le nouveau langage n'apporte pas d'éclaircissement sur ce point. En effet, si l'on considère par exemple un bloc, ce qui est à l'intérieur peut être envisagé suivant de nombreux points de vue. Par exemple, l'intérieur d'un capteur mobile peut être modélisé suivant sa structure électronique, électrique ou encore informatique suivant que le concepteur s'intéresse à l'un ou à l'autre. La difficulté vient lors des phases transitoires au cours desquelles le type - ou la technologie - concerné peut varier. Cela ne fait qu'ajouter un point de variation sémantique à cause notamment du large champ sémantique couvert par le concept de bloc. On trouve là la limite de la généralisation de ce concept et il semble inévitable lors d'un projet de rapidement devoir typer les différents blocs selon le domaine à modéliser.

En SysML, les diagrammes de bloc (BDD) et les diagrammes internes de bloc (IBD) sont directement hérités d'UML (*Diagramme de Classe* et *Diagramme de Structure Composite*) en y apportant des restrictions et des extensions (OMG, 2007b, p.44). Les liens entres les deux notations sont donc parfois assez évidents, même si du coup l'utilisation d'outils SysML, pour la plupart assez récents, provoque des situations paradoxales (cf. figure 6 où l'outil permet d'insérer une classe au lieu d'un bloc dans un IBD). Notons que cela n'est pas possible avec Enterprise Architect.

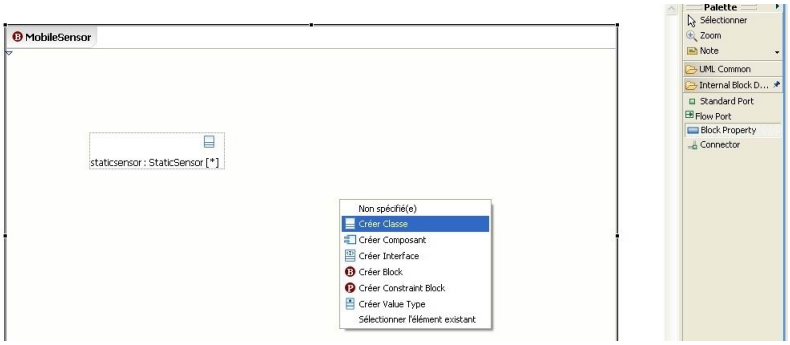

FIG. 6 – *Héritage de menus contextuels avec Embedded Plus*

5.2 Utilisation de SysML : jusqu'où ?

Lors de notre expérimentation de SysML nous avons été amené à nous poser la question du passage à l'échelle. en effet, au travers d'un cas concret relativement peu complexe en terme de taille ou de nombre de composants en jeu, nous avons déjà été confrontés à un certain nombre d'interrogations.

Parmi ces dernières, la question de la profondeur de modélisation nous a semblé particulièrement importante à déterminer. En effet, même si à ce jour aucune méthode d'ingénierie système ne s'appuie sur SysML, il est clair que ce langage, à l'instar d'UML, est plutôt orienté

vers une approche de raffinements successifs. Cela étant posé, à un certain stade de la modélisation, on se trouve confronté à des diagrammes sur lesquels on a identifié les éléments matériels et logiciels. L'étape suivante consiste donc à modéliser ces mêmes éléments d'un point de vue technique. Par exemple, lorsqu'un bloc est identifié comme logiciel, il est alors nécessaire de le modéliser d'un point de vue logiciel et non plus système, avec des éléments spécifiques au logiciel (classe, objet ...). De même lorsqu'on a identifié un bloc électronique, il va falloir produire des schémas sur lesquels on spécifiera des éléments spécifiques à ce domaine (diodes, transistors, micro-processeurs ...). On peut parler de conception avancée. Le problème qui se pose alors est que SysML ne permet pas de spécifier cela. Il convient alors de se tourner vers des langages de modélisation adaptés à ces domaines. Ainsi, SysML peut être vu comme un langage de haut niveau permettant l'analyse et la conception de systèmes complexes jusqu'à une certaine granularité, mais qui n'est pas suffisant pour le développement complet d'un système. A un certain stade, il est nécessaire de passer à un langage plus spécifique au sous-système modélisé, comme UML pour le cas de sous-systèmes logiciels.

Le passage à un langage de modélisation plus spécifique lors de la conception avancé pose plusieurs problème. D'une part, comment passe-t-on d'un élément SysML à un élément représentant la même entité mais dans un langage différent. Si l'on considère le cas d'UML, le bloc représentant un élément logiciel va se retrouver sous la forme d'une classe ou d'un composant. Le cas d'UML est d'ailleurs particulier puisque SysML est un profil UML. D'autre part, on est amené à se demander comment préserver (et/ou transférer) les propriétés ou les contraintes spécifiées avec SysML en phase amont. Si l'on prend une des caractéristiques nouvelles de SysML, comment lier par exemple un <<requirement>> et les éléments de traçabilité afférents avec des éléments de modélisation de diagrammes UML (comme par exemple une classe) ? La spécification SysML ne dit rien à ce sujet. Notons ici un paradoxe de l'approche. SysML est défini comme un nouveau langage mais a été créé sous la forme d'un profil UML. Cela permet donc en théorie de mélanger à la fois des concepts UML et des concepts SysML. Si cette approche permet d'étendre le pouvoir d'expression d'UML, en revanche on est en droit de se poser la question de la sémantique qui en résulterait. Que dire par exemple d'un diagramme d'expression des besoins indiquant qu'un besoin est satisfait par une classe alors que le concept de classe n'est pas défini dans SysML ? Et même si cette représentation était satisfaisante, ce que nous ne croyons pas, qu'en est-il du passage de SysML à un langage autre que UML ? Enfin, si l'on pose comme règle de passer à UML dès qu'un élément du modèle est identifié comme logiciel, on est confronté au problème de la cohérence du modèle UML dans sa globalité. En effet, si le passage à un diagramme de classe à partir d'un bloc semble aisé, il n'en est pas forcément de même pour les autres diagrammes. Par exemple, est-il nécessaire de répéter en UML un diagramme de séquence SysML mettant en scène l'interaction entre deux blocs logiciels ? Dans l'affirmative, on se retrouve avec un risque de duplication de l'information et donc un risque de désynchronisation entre le diagramme SysML et le diagramme UML. Dans la négative, le modèle UML ne sera pas complet. Et dans le cas d'un bloc logiciel auquel serait attaché un diagramme paramétrique, comment traduire ce dernier en UML ? On le voit la réponse à ces questions nécessite un effort de clarification de la part de la communauté.

L'autre grand absent dans la mouvance SysML est l'aspect méthodologique. En effet, si UML vient du besoin d'unifier les méthodologies objets, ce à quoi il a par ailleurs échoué, SysML répond lui à une tentative de création d'un nouveau standard pour la conception des systèmes. Ses bases méthodologiques liées au domaine de l'ingénierie système sont donc en-

core moins fournies. sachant que SysML, comme UML, est une notation et non une méthode, on peut se demander quelle approche particulière est la plus adaptée, et en particulier s'il faut se tourner vers les méthodes issues de l'ingénierie des systèmes ou bien adapter les méthodes logicielles. L'autre challenge auquel SysML devra répondre est celui de la normalisation. En effet les divers domaines de l'ingénierie système sont souvent fortement contraint par des normes sévères (aéronautique, automobile ...). La normalisation du langage ainsi que l'insertion du langage au sein de processus normalisé prendra sans doute du temps.

5.3 Retour sur expérience

Cette étude de cas nous a permis de mettre en œuvre une modélisation en langage SysML. Nous donnons ici notre retour sur expérience concernant d'une part SysML et d'autre part l'apport de SysML à la modélisation des réseaux de capteurs.

Cette étude a levé quelques interrogations qui ne semblent pouvoir être résolues qu'après un certain nombre d'expériences pratiques, notamment industrielles. Pour ce qui est de notre expérience, nous pouvons simplement avancer quelques pistes de réflexion et quelques commentaires sur l'utilisation conjointe des deux notations :

- La documentation SysML permet de mettre en œuvre sans trop de problème les nouveaux diagrammes.
- Pour des développeurs habitués à manipuler les concepts UML 2 de port ou de connecteur, le passage au concept de bloc ne pose aucun problème, si ce n'est la perpétuelle interrogation sur le niveau de détail entre logiciel et matériel.
- Les outils SysML commencent à être utilisables. Pour notre part nous avons essayé IBM Rational Software Modeler avec le plug-in SysML d'EmbeddedPlus Engineering[4] et Topcased[5]. Tous deux possèdent l'inconvénient d'être principalement issus d'outils UML 2 existants. L'avantage est l'intégration dans un même outil des deux langages à utiliser. L'inconvénient est l'amalgame fait parfois entre les deux notations, comme nous l'avons illustré en figure 6.
- Une des avancées qui nous semble significative dans l'utilisation de SysML concerne le concept de *requirement*. En effet, non seulement il permet d'exprimer directement les besoins là où UML permettait uniquement de les représenter avec des cas d'utilisation et des notes par exemple, c'est-à-dire à un stade de conception un peu plus avancé. Les relations de traçabilité de SysML sont plus précises que par exemple les relations de type <<trace>> ou <<dépendance>> d'UML. Même si les outils actuels ne permettent pas de rendre toute l'utilité d'un tel concept (trace arrière quand on détecte la violation d'une contrainte par exemple), il nous semble fort intéressant, surtout dans un contexte où le matériel et le logiciel sont très liés.

Peut-être que le développement de SysML verra les ingénieurs système se rapprocher des notations des ingénieurs logiciels. SysML a par exemple réduit le nombre de diagrammes UML pour simplifier le nombre de concepts utilisés. Mais si on extrapole l'expérience de l'arrivée d'UML 2 pour la communauté des ADLs, comme décrit par Bruel (2007), il semble que les changements de réflexes soient trop difficiles à mettre en place dans des communautés très ancrées dans leurs habitudes (comme dans l'aéronautique par exemple).

[4] Cf. l'URL : `http://www.embeddedplus.com`
[5] Cf. l'URL : `http://www.topcased.org`

D'autre part, l'utilisation de SysML pour la modélisation des réseaux de capteurs a globalement été concluante. En effet, la prise en compte des aspects systèmes permet de bien identifier ce qui est du ressort de l'ingénierie système et ce qui du ressort de l'ingénierie logicielle. L'autre aspect intéressant concerne l'ajout des diagrammes paramétrique qui vont permettre de spécifier notamment sur un même modèle des équations mathématiques importantes dans le cadre de systèmes mobiles (localisation d'un capteur, puissance d'émission ...). Nous n'avons pas traité en profondeur les aspects liés au capteurs physiques, considérant que nous réutilisions ceux fournis avec les capteurs mobiles. Il est certain que SysML représenterait un grand intérêt dans le cadre d'un système dans lequel le développement de matériel serait une des actions à réaliser.

6 Conclusion

Dans le domaine de l'ingénierie logicielle, UML a apporté une indéniable révolution, devenant une norme de fait. L'ingénierie système tente de franchir ce même cap en proposant aux ingénieurs système le nouveau langage SysML. L'apparition de nouvelles technologies comme les réseaux de capteurs sans fil dans lesquelles le logiciel et le matériel sur lequel celui-ci est déployé sont fortement entrelacés, nécessitent de nouveaux supports de modélisation. En effet, l'entrelacement de ces deux mondes précédemment souvent disjoints, impose de tirer le meilleur parti possible des avancées des deux domaines. Dans ce cadre, nous avons mené une étude de cas utilisant conjointement UML et SysML afin de modéliser une application basée sur un réseau de capteurs sans fil pour faire du recensement urbain.

Cette étude a mis en relief des points très positifs. L'utilisation de SysML apporte un réel bénéfice par rapport à UML pour la modélisation de contraintes liées au matériel, la définition et la traçabilité des besoins ou encore l'expression de contraintes mathématiques. Ce langage s'avère également un bon moyen de communication avec des ingénieurs métiers grâce notamment au concept générique de bloc. Il y a fort à parier que certaines de ces avancées vont rapidement se répercuter dans UML.

Cependant, nous avons eu à nous confronter à plusieurs aspects contraignants. Parmi ceux-ci, le manque de précision pour trouver la charnière entre l'utilisation de SysML et d'UML (ou d'autres langages et/ou méthodes de conceptions liés à des domaines spécifiques tels que la mécanique, l'électronique ou l'hydraulique) nous a souvent gêné. D'autre part, le fait que les outils mettant en œuvre SysML sont issus d'outils UML rend cette frontière encore plus ténue et nous paraît être source de malentendus. De plus, la généricité du concept de bloc est source également d'incompréhensions et nécessite d'être rapidement précisée lors de la modélisation. Enfin, malgré la filiation avec UML, il y a un manque dans la littérature d'études de cas mettant en œuvre SysML afin d'avoir une vision plus exhaustive des possibilités du langage. Gageons que cela aidera à adapter les normes et les méthodes de développement système souvent fort contraignantes et supportant mal ce type de langage graphique. Cette étape est indispensable pour permettre à SysML de passer du coup d'essai au coup de maître.

Pour nous, cette étude a mis en évidence l'intérêt et les manques d'un tel langage et la nécessité d'un effort de recherche soutenu pour le développer. Ainsi, nous avons pour objectif d'investir ce champ de recherche. Pour ce faire, nous comptons travailler à répondre aux questions identifiées dans le paragraphe précédent. Plus particulièrement, nous nous intéressons à améliorer l'interaction entre UML et SysML. Cela revient d'une part à déterminer précisément

le niveau à partir duquel le langage orienté système qu'est SysML devient trop spécifique au domaine du système pour permettre une modélisation de qualité du logiciel devant fonctionner sur ce système. D'autre part, nous comptons nous appuyer sur les techniques d'ingénierie des modèles et de transformations de modèle afin à la fois d'automatiser au maximum le passage d'un modèle à un autre et de garantir également le respect des contraintes et les liens de traçabilité exprimés avec SysML.

Remerciements

Ce travail est financé à la fois par la Communauté d'Agglomération de Pau Pyrénées[6] à travers le financement d'une bourse de thèse et par la région Aquitaine[7] via le financement d'une plate-forme de test de réseaux de capteurs. Nous les remercions pour leur soutien.

Références

Barbier, F., B. Henderson-Sellers, A. L. Parc-Lacayrelle, et J.-M. Bruel (2003). Formalization of the Whole-Part Relationship in the Unified Modeling Language. *IEEE Transactions on Software Engineering 29*(5), 459–470.

Belloir, N. (2004). *Composition logicielle basée sur la relation Tout-Partie*. Ph. D. thesis, Université de Pau et des Pays de l'Adour.

Boehm, B. (2006). Some Future Trends and Implications for Systems and Software Engineering Processes. *Systems Enginnering 9*(1), 1–19.

Bruel, J.-M. (2007). Composition logicielle à l'aide de machine à états. *Génie Logiciel* (80), 2–6. ISSN : 0295-6322.

Feiler, P. H., D. P. Gluch, et J. J. Hudak (2006). The Architecture Analysis and Design Language (AADL) : An Introduction. Technical Note CMU/SEI-2006-TN-011, Software Engineering Institute.

Friedenthal, S., A. Moore, et R. Steiner (2007). OMG Systems Modeling Language Tutorial. `http://www.omgsysml.org/INCOSE-2007-OMG-SysML-Tutorial.pdf`.

Harel, D. et B. Rumpe (2004). Meaningful Modeling : What's the Semantics of "Semantics" ? *IEEE Computer 37*(10), 64–72.

Khemapech, I., I. Duncan, et A. Miller (2005). A survey of wireless sensor networks technology. In *PGNET, Proceedings of the 6th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking & Broadcasting*.

OMG (2005). UML 2.0 Superstructure Final Adopted specification formal-05-07-04. OMG document, Object Management Group.

OMG (2007a). A UML Profile for MARTE. OMG document ptc/07-08-04, Object Management Group.

OMG (2007b). OMG Systems Modeling Language Specification. Technical Report formal/07-09-01, Object Management Group.

[6] `http://www.agglo-pau.fr`

[7] `http://aquitaine.fr/`, numéro de projet : 20061104047

Sachdeva, G., R. Dömer, et P. Chou (2005). System Modeling : A Case Study on a Wireless Sensor Network. Technical Report TR-05-12, Center for Embedded Computer Systems, Irvine, California.

Satyanarayanan, M. (2001). Pervasive computing : vision and challenges. *IEEE Personnal Communications 8*(4), 10–17.

Stallings, W. (2005). *Réseaux et communication sans fil*. Pearson education.

Szyperski, C., D. Gruntz, et S. Murer (2002). *Component Software – Beyond Object-Oriented Programming – Second Edition*. ACM Press. New York, NY : Addison-Wesley.

Summary

SysML is the new system modeling language promoted by the OMG. It can be seen as an extension of UML that targets the modeling of a broad range of complex systems. Its application field is then wider and its UML roots are useful to design embedded systems mostly composed by software. Applications deployed on wireless sensor networks (WSN) are a good example of such applications as the interactions between hardware and software components are of prime importance in the design of efficient sensor networks. In this paper we describe our feed-back, based on a case study that models a WSN system. In this case study, we have used both SysML (to design the complete system) and UML (to model the software parts). We present the overlapping of the two languages and we compare the both structural diagrams of theses languages.