Des buts à la modélisation système : une approche de modélisation des exigences centrée utilisateur

Fernando Wanderley, Nicolas Belloir, Jean-Michel Bruel, Nabil Hameurlain, João Araújo

Des buts à la modélisation système : une approche de modélisation des exigences centrée utilisateur

Fernando Wanderley* — **Nicolas Belloir**** — **Jean-Michel Bruel***** — **Nabil Hameurlain**** — **JoãoAraújo***

* *CITI, FCT, Universidade Nova de Lisboa, Caparica, Portugal*
(f.wanderley, p191)@fct.unl.pt
** *LIUPPA, Université de Pau et des Pays de l'Adour*
BP 1155, 64013 Pau Cedex, France
(nicolas.belloir,nabil.hameurlain)@univ-pau.fr
*** *CNRS, IRIT, Université de Toulouse, Cedex, France*
bruel@irit.fr

RÉSUMÉ. *Un des problèmes en ingénierie des exigences consiste à capter les besoins des utilisateurs le mieux possible. Or force est de constater que les supports d'ingénierie tels que les modèles orientés but ou les diagrammes d'exigences orientés système tels que ceux de* SysML *sont parfois trop complexes pour les utilisateurs finaux. Dans cet article, nous proposons un processus systématique permettant aux utilisateurs d'exprimer les exigences à l'aide de modèles cognitifs plus simples tels que les Mind Maps. Ces derniers sont alors transformés en modèles* Kaos *puis en modèles* SysML *en appliquant des techniques de transformations de modèles. Nous avons appliqué cette approche à un cas d'application industriel.*

ABSTRACT. *One of the existing problems in requirement engineering consists in arriving to capture final user requirements as well as possible. But it is clear that engineering methods such as goal-oriented models or system-oriented requirement diagrams such as* SysML *are sometimes too hard for end-users. In this paper we define a systematic process allowing ennd-users to specify requirements using cognitive models such as mind maps. Then, those models are transformed into* Kaos *models, then in* SysML *models, using model transformation rules. This approach is applied to an industrial case application.*

MOTS-CLÉS : *Modèles de recueil des exigences graphiques, ingénierie des exigences orientée utilisateurs, ingénierie des modèles, Mind Maps,* Kaos, SysML

KEYWORDS: *Modèles graphiques des exigences, Recueil des besoins centré-utilisateur, Ingénierie des modèles, Mind Maps,* Kaos, SysML

1. Introduction

Selon une étude récente menée par le Standish Group[1], environ 66 % des logiciels développés ne répondent pas aux attentes des utilisateurs que ce soit au niveau de leurs fonctionnalités (niveau système) ou au niveau de leurs comportements (niveau buts). Ce rapport indique que la plus grande part des échecs relevés était liée à des malentendus et un manque de communication entre les ingénieurs des exigences et les utilisateurs. Ces données confirment qu'aucune autre partie du travail de création d'un logiciel n'est aussi difficile que d'établir une communication fiable entre les financiers, les experts domaines, les ingénieurs des exigences et les ingénieurs logiciels (Pressman, 2006 ; Sommerville, 2010).

D'autre part, les modèles graphiques des exigences tels que les modèles orientés but (comme KAOS (Dardenne *et al.*, 1993)) ou les modèles SYSML[2] (Object Management Group, 2012) pour l'ingénierie système sont une des manières les plus efficaces pour identifier les exigences d'un système et les besoins utilisateurs.

Dans ce contexte, des études telles que (Sommerville, 2010) ou (Wanderley *et al.*, 2012) ou (Wanderley et Araujo, 2013) ont montré que des modèles cognitifs tels que les *Mind Maps*[3] pouvaient être utilisés en ingénierie des exigences afin de faciliter la communication entre les différentes parties prenantes. Ces travaux ont également montré que les techniques d'ingénierie des modèles pouvaient aider à la génération de modèles d'exigences traditionnels (tels que des modèles conceptuels exprimés en UML ou encore en KAOS). Cette approche permet de réunir les propriétés bien connues des *Mind Maps* et les techniques de transformations de modèles afin de transformer automatiquement les exigences exprimées sous la forme de *Mind Maps* en modèles logiciels tels que diagrammes de classes UML, modèles fonctionnels et modèles KAOS. Enfin, dans une étude récente (Wanderley et Araujo, 2013), nous avons montré comment l'utilisation de *Mind Maps* et des techniques agiles de recueil des exigences, encourageait la construction de modèles de buts KAOS plus efficacement et plus simplement, en impliquant des utilisateurs non formés à KAOS.

Dans ce papier, nous étendons ces techniques afin de définir des transformations permettant de passer des modèles KAOS générés aux modèles SYSML suivants : diagrammes des exigences et diagrammes de définition de blocs. Une étude de cas industrielle complète (expression des exigences en *Mind Maps*, transformation de ces modèles en KAOS, puis transformation en modèles SYSML) a été réalisée.

La suite de ce papier est organisée comme suit. La section 2 présente les concepts nécessaires à notre approche, c'est-à-dire les *Mind Maps*, les modèles orientés but (et plus particulièrement KAOS), SYSML et les techniques d'ingénierie des modèles. La section 2.5 décrit les métamodèles de chacun, et les transformations entre eux.

---

1. Standish Group -Accessed May 2012 - `http://blog.standishgroup.com/`
2. SYSML - System Modeling Language
3. Nous utilisons volontairement le terme *Mind Maps* au lieu de traductions françaises telles que carte heuristique ou carte cognitive ou carte mentale ou carte des idées afin de rester sur un terme générique reconnu et supporté par des logiciels tels que *xMaps* ou autres.

La section 3 décrit l'application de notre approche dans un contexte industriel et plus particulièrement les transformations entre les modèles *Mind Maps* et KAOS, puis entre KAOS et SYSML. La section 4 présente quelques approches similaires tandis que la section 5 apporte une conclusion et présente des directions pour de futurs travaux.

2. Fondements

2.1. *Les modèles Mind Maps en tant que modèles centrés utilisateur*

Une *Mind Map* selon (Buzan et Buzan, 2003) est un diagramme utilisé pour voir, classifier et organiser des concepts, ainsi que pour générer de nouvelles idées. Elle est utilisée pour connecter des mots, des idées et des concepts à une idée ou un concept central. Elle est similaire à un réseau sémantique ou à une carte cognitive, mais sans les restrictions sur les types de connections utilisées. Une *Mind Map* est un diagramme radial qui, à l'aide d'un vocabulaire (c'est à dire d'un ensemble de mots-clés), peut représenter et modéliser de manière cognitive un concept ou un domaine spécifique. Les principaux bénéfices de l'utilisation de ce type de représentation sont : organisation des idées et des concepts, mise en accent de mots significatifs, association entre éléments d'une branche, regroupement d'idées, support à la mémoire visuelle et à la créativité, déclenchement d'innovations, le tout de manière simple. Certaines études, résumées dans (Wanderley et Araujo, 2013), soulignent que ce type de représentation rend plus facile pour l'esprit humain le traitement de l'information, tout en réduisant la charge cognitive nécessaire pour absorber les concepts du domaine et leurs objectifs.

Lors de travaux précédent, nous avons mis en évidence que plusieurs techniques et outils basés sur les *Mind Maps* sont considérés, tant par le monde académique que par le monde industriel, comme de puissants outils pour l'élicitation des exigences, notamment en développement agile (Wanderley et Araujo, 2013) ; nous renvoyons à la lecture de cet article pour les détails de cette étude bibliographique.

L'ingénierie des exigences est reconnue comme un processus social qui se caractérise par des prises de décisions permanentes entre de nombreux participants (gestionnaires, utilisateurs finaux, et analystes du système). Lors de la phase de recueil des exigences, (Moody *et al.*, 2003) affirme que le développement logiciel est plus un travail artisanal qu'une réelle discipline d'ingénierie. À cet égard, la cognitive humaine joue un rôle essentiel pour la compréhension des problèmes humains et organisationnels dans l'ingénierie des exigences, et sert à identifier les moyens d'amélioration de la qualité de la perception visuelle du modèle de spécification produit. Dans cet article, nous adoptons une représentation visuelle centrée sur l'utilisateur basée sur les modèles mentaux (Davidson *et al.*, 1999).

Dans ce contexte, ce papier propose l'adoption des *Mind Maps* afin de faciliter la communication (entre les utilisateurs finaux et les ingénieurs des exigences) et offrir ainsi un support simplifié pour la modélisation des buts.

2.2. *Ingénierie des exigences basée sur les buts*

L'ingénierie des exigences basée sur les buts (GORE[4]) considère l'organisation et les objectifs des acteurs comme la source des exigences (fonctionnelles et non-fonctionnelles). Avant de travailler à l'élicitaton des exigences, il convient de se concentrer sur l'élicitation des buts (van Lamsweerde, 2001). Les GOREs modélisent donc les buts. Ces derniers peuvent être vus comme les propriétés désirées du système qui ont été exprimées par les utilisateurs. En outre, ils peuvent être spécifiés à différents niveaux d'abstraction, couvrant les préoccupations stratégiques à haut-niveau et à un niveau inférieur les problèmes techniques. Selon la classification de (van Lamsweerde, 2001), plusieurs méthodes peuvent être considérées comme appartenant aux GOREs : le framework i* (Yu, 1995), GBRAM (Antón *et al.*, 1995), NFR (Lawrence Chung, 2000), KAOS (van Lamsweerde, 2001) et (Dardenne *et al.*, 1993), TROPOS (Castro *et al.*, 2002), la carte des buts/stratégies(Rolland et Salinesi, 2005), ou encore GRL (University of Toronto,). Parmi ces méthodes, KAOS est l'une des plus citées. C'est pour cette raison que nous avons focalisé notre travail dessus.

Un *but* en KAOS peut être défini comme une déclaration d'intention sur un système dont la satisfaction, en général, exige la coopération de certains *agents* qui configurent le système. Les buts sont satisfaits par les *exigences* qui sont prises en compte dans les spécifications des opérations du logiciel ou par des hypothèses qui expriment un comportement effectué par des *agents extérieurs*. Les *agents logiciels* sont des composants actifs qui réalisent des opérations satisfaisant les exigences pour lesquelles elles ont été définies. KAOS propose quatre visions d'un problème à travers les modèles suivants : le *modèle des buts*, le *modèle objet*, le *modèle de responsabilité* et le *modèle opérationnel*. Ces modèles sont basés sur les *buts (Goal)*, les *exigences (Requirement)*, les *entités (Entity)*, les *évènements (Event)*, et les *relations (Relationship)* entre ces concepts. Des *objets* peuvent être spécifiés pour décrire le modèle structurel du projet. Ils peuvent être passifs (*entity*, *event* ou *relationship*) ou actifs (*agents*). Des obstacles (*contraintes (constraint)*) et des relations entre buts (*conflits entre buts (conflict goals)*) sont utilisés pour analyser les scenarii dans lesquels des buts pourraient être non satisfaits, et contribuent ainsi à l'identification des vulnérabilités du système (Lamsweerde et Letier, 2003).

2.3. *Le langage* SYSML

SYSML est un langage de modélisation graphique pour l'ingénierie système, et plus particulièrement pour les systèmes complexes. Il est utilisé pour spécifier, analyser, concevoir et vérifier les systèmes complexes. Ce langage fournit des représentations graphiques basées sur une sémantique semi-formelle pour modéliser les exigences, la structure, le comportement et certains aspects mathématiques d'un système. Il peut être intégré avec d'autres méthodes et modèles d'ingénierie.

---

4. GORE - Goal-Oriented Requirements Engineering

Ce langage a été défini comme une extension de UML 2[5]. Il est construit à partir d'un sous-ensemble de ce dernier auquel on a ajouté un certain nombre de concepts manquant afin de satisfaire l'ingénierie système. Son utilisation par le monde industriel connaît une croissance significative ; il a été utilisé dans la réalisation de cas industriels en production. Par exemple, Airbus l'a utilisé pour modéliser une partie des spécifications de l'A350 (Riviére et Benac, 2010). Dans cette étude, nous nous limitons à deux de ses neuf diagrammes.

Le *diagramme des exigences* (*Requirements Diagram*) permet de représenter les exigences et les relations entre-elles de deux manières. Soit à l'aide d'un tableau, soit à l'aide d'une représentation graphique basée sur le diagramme de classes d'UML. Dans les deux cas, les exigences sont représentées selon un point de vue décompositionnel, permettant d'exprimer la traçabilité, la vérification ou la satisfaction par ou vers des éléments autres du système (tels que des blocs structurels (*blocks*) ou des activités (*activity*) par exemple).

Le *diagramme de définition de blocs* (*Block Definition Diagram*) étend également le diagramme de classe d'UML. Il représente la vue structurelle du système d'un point de vue externe. Le système est donc vu comme un ensemble de systèmes et de sous-systèmes. Il aide à capturer les composants constituant le système et les relations entre eux. La vue interne structurelle du système est donnée par un autre diagramme, le diagramme de blocs internes (*Internal Block Diagram*), représentant la vue interne d'un bloc (spécifié dans un diagramme de définition de blocs) à l'aide de parties et de connexions entre ces parties.

D'autres diagrammes sont définis par SysML, notamment des diagrammes comportementaux, mais nous ne les considérons pas dans cette étude. La traçabilité entre éléments du système est une des avancées importantes de SysML. Elle permet notamment de relier les exigences aux éléments de modèles les satisfaisant. Cela permet d'assurer une meilleure couverture des exigences et de leur prise en compte.

2.4. *Ingénierie basée sur les modèles*

L'ingénierie basée sur les modèles (MDE[6]) se concentre sur les modèles comme principal intérêt (Schmidt, 2006). L'utilisation des modèles pour la construction de systèmes complexes est un des moyens permettant de résoudre les capacités cognitives limitées des humains à comprendre un système complexe dans sa globalité (Kleppe *et al.*, 2003). La MDE propose de réduire les besoins qu'a l'ingénieur logiciel d'interagir manuellement avec le code source, lui permettant de se concentrer sur des modèles de plus haut-niveau, dans lesquels il peut notamment s'abstraire des problèmes de développement des différentes plateformes supports. Cette stratégie permet d'améliorer certains points clés de la création d'un système tels que la productivité, l'interopérabilité et la communication (Mernik *et al.*, 2005).

---

5. UML - Unified Modeling Language
6. MDE : Model Based Engineering

Le développement par les modèles requiert une définition rigoureuse de ces derniers. Cela est réalisé par la définition de *métamodèles* et par la spécification de *transformations* entre ces métamodèles. Ces dernières permettent de transformer un modèle exprimé grâce à un métamodèle donné en un autre modèle, conforme à l'initial (mais exprimé différemment), voire en code source.

Dans notre travail, nous avons utilisé pour décrire nos transformations le langage ATL[7] et son moteur d'exécution. La section suivante présente les concepts essentiels des métamodèles *Mind Map*, KAOS et SYSML. Leurs descriptions complètes peuvent-être trouvées dans nos travaux précédents (Wanderley et Araujo, 2013). La description complète du métamodel SYSMLpeut être trouvée dans (Object Management Group, 2012). Un résumé des règles de transformations entre ces modèles suivent.

2.5. *Métamodèles et transformations de modèles*

2.5.1. *Le métamodèle Mind Map*

Le métamodèle présenté par (Wanderley et Araujo, 2013) décrit qu'une *Mind Map* est associée à une structure (*Structure*) et à un contenu ou un concept (*Content*). Cette structure est définie par la composition de nœuds (*Node*) et de connexions (*Edge*) formant la structure d'un graphe. Ce type de graphe peut être défini par une paire $G = (V, E)$ où $V \neq \varnothing$ est défini comme un ensemble fini d'éléments appelés sommets (ou nœuds), et *E* est une relation binaire sur *V*. L'élément *E* forme un ensemble de paires, de la forme $(vi, vj), (i, j)$ qui sont appelés bords.

Les nœuds peuvent être classifiés en groupes de nœuds (*Group*) ou en feuilles (*Leaf*), établissant ainsi une relation hiérarchique entre eux. La notation (*Notation*) est le graphe de ressources utilisé pour spécifier et différencier un nœud d'un autre dans la carte. Elle peut être représentée textuellement ou via un format XML (*Textual*), par une icône (*Icon*), cette dernière pouvant être utilisée pour tout autre élément ou par une image (*Image*). La structure d'une *Mind Map* peut également être représentée graphiquement (*GraphicalConector*), étiquetée (*Tag*) en tant que *NodeLink* ou comme une flèche *Arrow*.

2.5.2. *Le métamodèle* KAOS

La Figure 1 (Matulevicius et Heymans, 2005) montre les éléments du métamodèle KAOS essentiels à la transformation de modèles. La métaclasse *KAOS* en est la racine. Cette métaclasse est composée de nœuds et de liens (*Nodes* et *Links*). Dans le métamodèle, les nœuds peuvent être : un obstacle *Obstacle* ; un nœud opérationnel *OperationNode* ; un objet *Object* (représenté par une entitée *Entity*, dans les considérations objets, et par un agent *Agent* dans les considérations agents), et peut être un but *Goals* (traduit par un *Requirement*), une propriété de domaine (*DomainProperty*),

7. ATL - ATL Transformation Language - `http://www.eclipse.org/atl/`

une attente (*Expectation*) ou un but léger (*SoftGoal*). Un but léger est un nœud qui ne peut jamais être racine, ni feuille de type *Refinement*, d'un autre but.

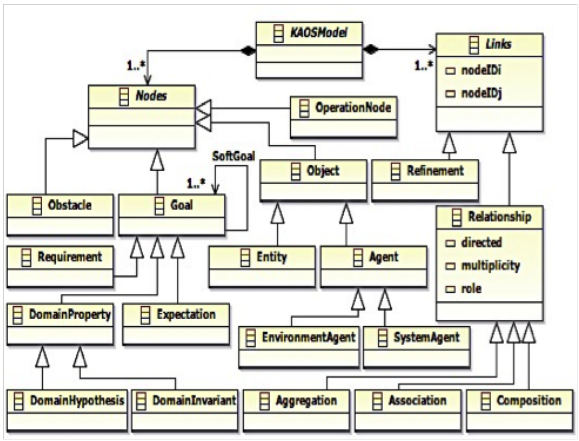

Figure 1. *Métamodèle* KAOS

Bien qu'il y ait de nombreux liens, nous nous focalisons dans cet article uniquement sur les liens de raffinement (*Refinement*) et les relations (*Relationships - Aggregation*, *Association* et *Composition*) dans le contexte du modèle objet.

2.5.3. *Le métamodèle* SysML

SysML n'a pas à proprement parlé de métamodèle. Il a été défini par un profil UML, lui-même basé sur un sous-ensemble UML appelé *UML4SysML*. Cependant, il est possible de définir un métamodèle basé sur sa définition officielle.

La Figure 2 illustre les éléments essentiels du métamodèle traitant de l'expression des exigences. Celle-ci est basée principalement sur une extension des métaclasses *Class* et *Trace* du métamodèle UML. Une exigence (*Requirement*) est une classe (*Class*) à laquelle on a adjoint deux propriétés textuelles permettant de spécifier une valeur d'identifiant (*id*) et un texte (*texte*) la décrivant. Les exigences peuvent être liées entre elles en utilisant des extensions de la métaclasse UML *Trace* : *Derive* et *Refine* permettent respectivement d'étendre et de raffiner une exigence ; *Satisfy* est utilisée afin de lier une exigence à un élément de modèle la satisfaisant ; *Verify* permet de lier un cas de test à une exigence afin de définir comment vérifier la réalisation de l'exigence dans le système. Les cas de test peuvent être spécifiés en utilisant à la fois une opération réalisant le test (structurel) et sa description comportementale.

Notre approche nécessite également un certain nombre de concepts transverses SysML, définis à partir du package *ModelElements*, tels que les points de vue («`viewpoint`») (servent à modéliser les préoccupations des utilisateurs), le lien de contenance («`containment`») (relation organisationnelle), ou encore la dépendance entre

un point de vue et une vue qui est matérialisée par la relation de contrôle («control»). Elle nécessite enfin la matérialisation de la notion d'acteur à travers la représentation «actor» définie dans le pacakge *Uses Cases*. Nous ne rentrons pas dans le détail de ces éléments de métamodèle et renvoyons à (Ahmad, 2013) pour plus de détail.

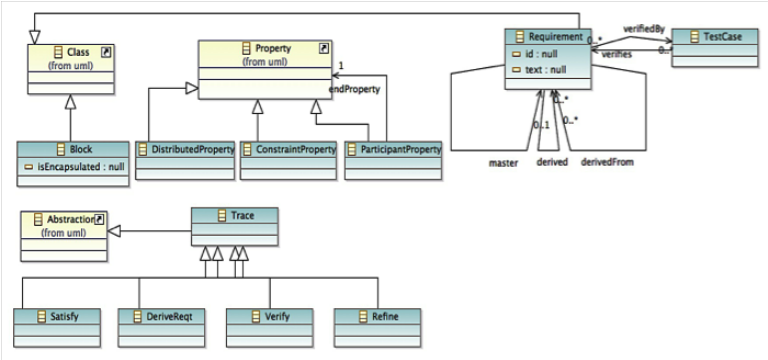

Figure 2. *Partie du métamodèle* SysML *concernant les exigences*

2.6. *Règles de transformations*

Nous décrivons ici un ensemble de règles permettant de transformer un modèle source décrit dans un métamodèle à un modèle cible décrit dans un autre métamodèle, en restant conforme au premier. Nous décrivons en premier les règles de transformations entre *Mind Map* et KAOS, puis celles entre KAOS et SysML.

2.6.1. *Transformations de Mind Map vers* KAOS

Le processus complet de transformations de Min Map vers KAOS a été décrit dans (Wanderley et Araujo, 2013) : nous y avons décrit une approche de recueil des exigences orientée modèle, permettant de générer systématiquement des modèles KAOS à partir de modèles *Mind Maps*, en utilisant un métamodèle exprimé en *Ecore*, et des transformations de modèles. Trois types de *Mind Maps* sont traitées, chacune spécifiant des préoccupations KAOS (agents, buts et objets). Après leurs transformations vers KAOS, l'ingénieur des exigences compose les modèles partiels générés afin de fournir un modèle KAOS complet. La nouveauté apportée par ce papier est de fournir un cadre permettant de passer des modèles KAOS exprimés à des modèles SysML et ainsi à fournir une série de transformations de modèles permettant de fournir un modèle des exigences SysML à partir de *Mind Maps* définies par les utilisateurs.

2.6.2. *Transformations de* KAOS *à* SysML

Nous avons identifié dans (Ahmad, 2013) un ensemble de correspondances entre des concepts KAOS et des concepts SysML. Elles sont résumées dans le tableau 1.

Concepts	KAOS	SYSML
Requirement Description	Abstract Goal, Elementary Goal	Textual Requirement
Monitoring	Contribution Goal	«satisfy»
Relationship	AND/OR, Contribution Nature (Positive, Negative), Contribution Type (Direct (Explicit), Indirect (Implicit))	«Verify», «Refine»
Dependency, Impact	Contribution Nature (Positive, Negative)	«Derive», «Contain»

Tableau 1. *Correspondance entre* KAOS *et* SYSML

Cela nous a permis de définir des règles de transformations que nous résumons ci-dessous, en les organisant en règles de transformations des exigences et règles de transformations architecturales.

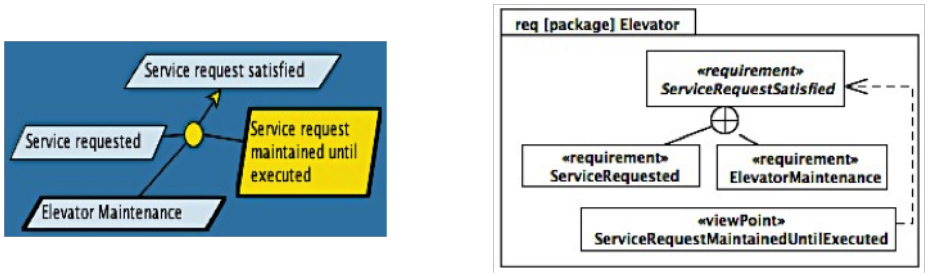

Figure 3. *Transformation d'un modèle* KAOS *vers un diagramme d'exigences* SYSML

Transformations des exigences : chaque but (*goal*) KAOS est transformé en une exigence («requirement») SYSML. Le but racine KAOS est transformé un une exigence abstraite SYSML (*abstract* «requirement»), et ses fils sont transformés en exigences SYSML à l'aide de la relation de décomposition $\otimes$. Les exigences KAOS sont vues comme des exigences systèmes SYSML et les attentes KAOS comme des exigences utilisateurs SYSML (matérialisées par les points de vue «viewPoint»). La Figure 3 illustre cette transformation.

Transformations architecturales : chaque entité KAOS (*entity*) est traduite en un bloc SYSML. Chaque opération KAOS (*operation*) est traduite en une activité SYSML ou en une opération d'un bloc. Chaque agent environnemental (*environment agent*) KAOS correspond à un acteur (*actor*) SYSML et chaque agent

système (*system agent*) KAOS à un bloc SYSML. Les Figures 4 et Figures 5 illustrent ces transformations.

Figure 4. *Objets et agents* KAOS *sources de la transformation vers* SYSML

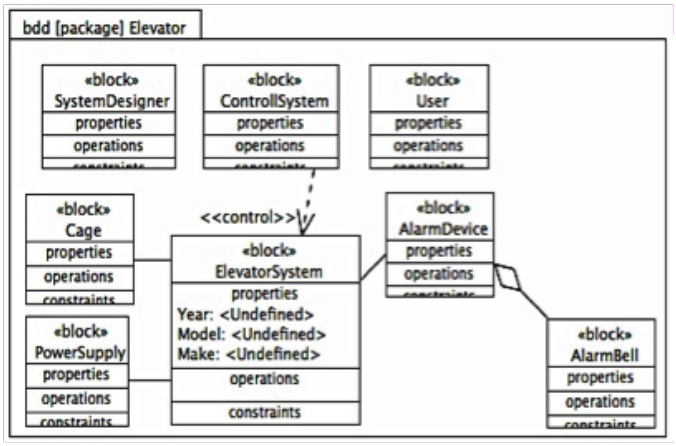

Figure 5. *Modèle de blocs* SYSML *généré*

3. Cas d'étude industriel

Cette section décrit l'application de notre approche à un cas d'étude industriel. Une partie a été présentée dans (Wanderley et Araujo, 2013) et étendue à la transformation des modèles générés vers SYSML.

Mobiciti[8] est une start-up brésilienne qui vend des produits innovants aux contenus informatifs, culturels ou publicitaires diffusables aux usagers des transports publics. *Audiobus* est un système intelligent qui utilise un système d'information géographique et diffuse les informations sous la forme d'un son numérique localisé. Des messages personnalisés et des contenus développés spécifiquement sont diffusés dans une zone donnée propre à chaque véhicule.

8. Mobciti site - `http://www.mobciti.com/`

3.1. *Génération de modèles* KAOS *pour Audiobus*

L'activité d'élicitation permettant la capture des agents, des buts et des objets a été réalisée par le client à distance via la plateforme *Hangout*. Cet environnement a été choisi pour sa facilité d'utilisation et pour ses aptitude de partage de bureau qui nous a permis de réaliser la modélisation de la *Mind Map* en ligne simultanément avec à la fois le client et l'ingénieur des exigences. Ce dernier peut poser des questions au client afin d'identifier les préoccupations principales de KAOS (i.e. agents, objets, buts). Les Figures 6 et 7 montrent le type des résultats obtenus. Notre approche définit un processus systématique permettant de générer chaque préoccupation KAOS séparément à travers une notation définie sous *Mind Map*. L'utilisation de *Mind Map* améliore la séparation des préoccupations de KAOS et aide à obtenir le modèle des buts.

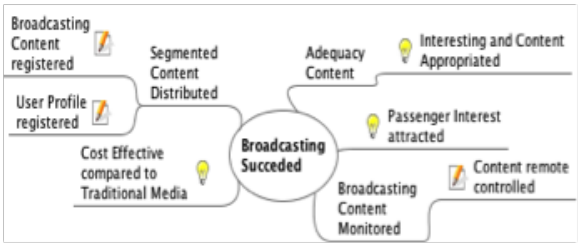

Figure 6. *Mind Map de Audiobus*

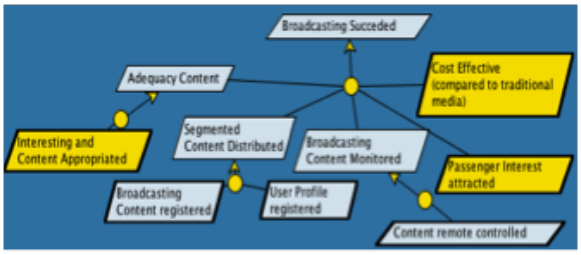

Figure 7. *Mind Map enrichie avec des annotations aidant à la transformation*

3.1.1. *Recomposition du modèle* KAOS

Après la génération des modèles KAOS partiels (i.e. modèles des buts, modèles d'agents et modèles objets), l'ingénieur des exigences se contente de les recomposer. Sa tâche consiste à assigner les responsabilités de chaque agent avec ses buts, ses exigences ou ses attentes respectifs. Il doit également raffiner le modèle objet et finalement spécifier les relations de contrôle entre un agent (composant logiciel) et les objets qu'il contrôle, ce qui donne au final le modèle KAOS décrit dans la Figure 8.

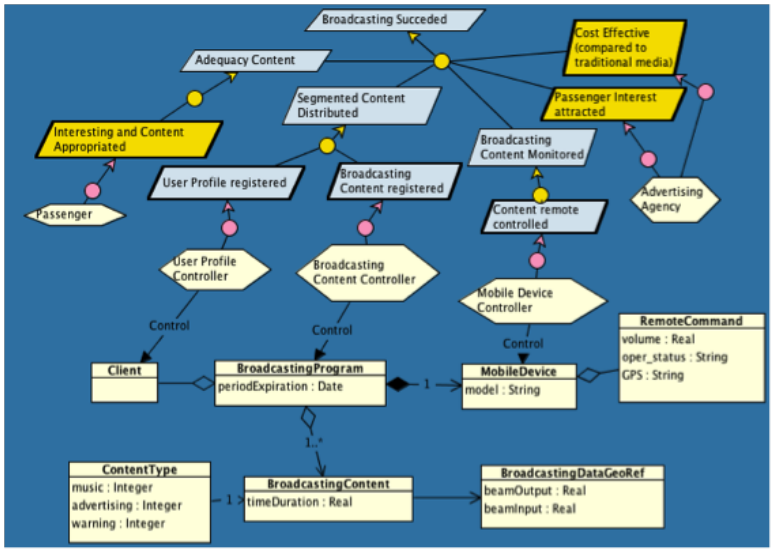

Figure 8. KAOS *du système Audiobus*

3.2. *Génération du modèle* SYSML *d'Audiobus à partir des modèles* KAOS

3.2.1. *Des buts aux exigences*

En se basant sur le modèle partiel de KAOS et sur les préoccupations générées par *Mind Map*, les buts (*Goal*), attentes (*Expectation*) et exigences (*Requirements*) d'Audiobus ont été transformés de la manière suivante (voir la Figure 9) : (i) le but racine (*BroadcastingSucceded*) a été transformé en «`requirement`» avec la propriété *abstract* positionnée à *vrai* ; (ii) les buts enfants de *BroadcastingSucceded* non définis en tant qu'attente ou qu'exigence ont été transformés en «`requirement`» et liés à leur parent (i.e. *Adequacy-Content* à l'aide de la relation de décomposition : (iii) les buts attentes (i.e. *CostEffective*) ont été transformés en point de vue («`viewPoint`») et ont été liés à leur parent à l'aide de la relation «`deriveBy`» : (iv) le but exigence (i.e. *UserProfileRegistred*) a été transformé en «`requirement`» et relié via «`satify`».

3.2.2. *Des agents et objets aux blocs*

En se basant sur les modèles partiels KAOS agents et objets générés par *Mind Map* (Figure 10), la seconde étape de notre méthode a produit un modèle en deux étapes distinctes (montrées par la Figure 11) : (i) tous les agents KAOS ont été transformés en «`actor`», après quoi l'ingénieur des exigences a dû spécifier l'acteur comme un utilisateur ou un agent système : (ii) toutes les objets (incluant les attributs et les relations) ont été transformées en «`block`» avec des propriétés (*attributes*) et les mêmes abstractions concernant les relations.

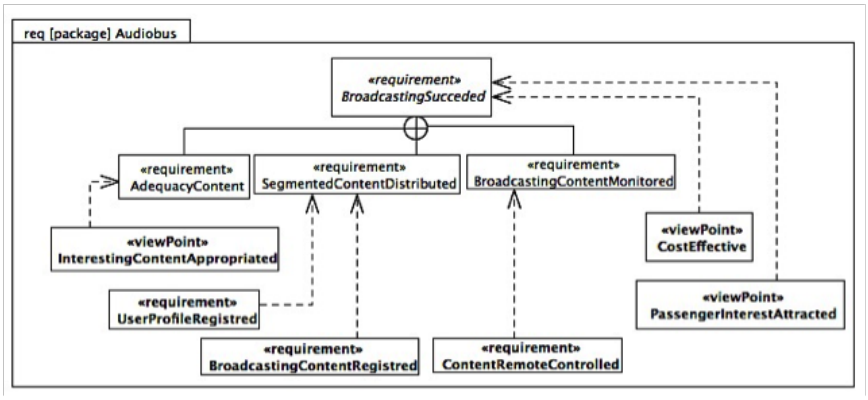

Figure 9. *Diagramme des exigences* SysML *d'Audiobus*

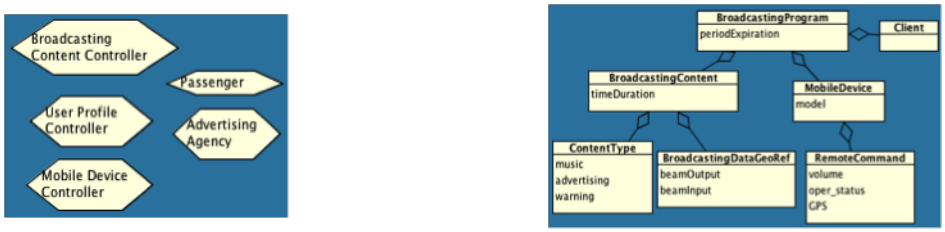

Figure 10. *Modèles* KAOS *partiels (Agents et Objets) d'Audiobus*

3.2.3. *Recomposition des modèles* SysML *d'Audiobus*

Après la transformation de chaque préoccupation de KAOS en SysML, la dernière activité de l'ingénieur des exigences a été d'annoter (manuellement) ces deux modèles SysML en les enrichissant avec des liens de traçabilité permettant d'améliorer la qualité sémantique du modèle. Dans l'exemple, chaque «`actor`» (utilisateur) est lié à son «`viewPoint`» correspondant ; chaque exigence à son «`block`» (*agent système*).

A la fin de l'étude de cas, le client s'est montré satisfait de la documentation générée (qui n'existait pas auparavant) pour son produit et a mentionné qu'elle avait été générée simplement et rapidement et avait produit un document d'exigences cohérent. Pour son prochain développement, l'entreprise envisage de mener une phase de recueil des besoins traditionnelle par une équipe et en parallèle la même étude appliquant cette approche par une autre équipe. L'objectif est d'établir une comparaison afin d'établir les apports de l'approche.

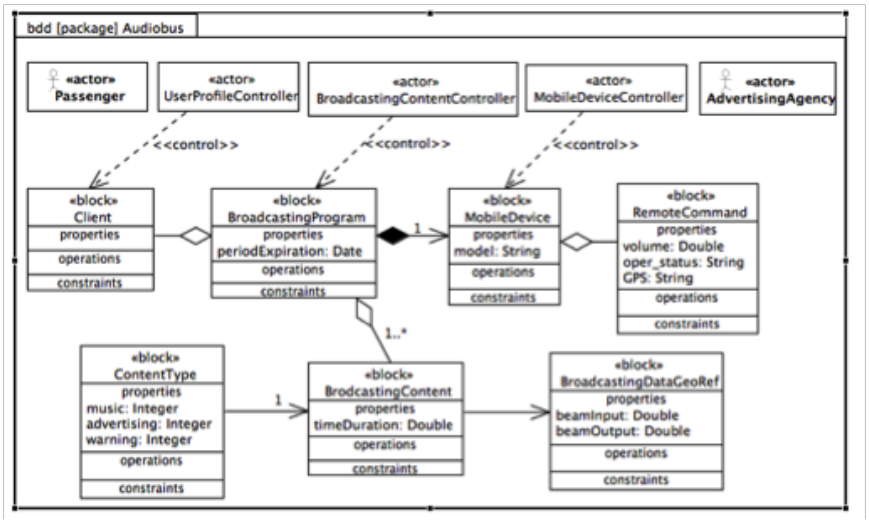

Figure 11. *Diagramme de bloc* SysML *d'Audiobus*

4. Related Works

Une étude récente (Monteiro *et al.*, 2012) a appliqué des techniques d'ingénierie des modèles aux GOREs, en réalisant une correspondance bi-directionnelle entre deux approches GORE (i* et KAOS), réalisant une plateforme interopérable qui permet de faire migrer un modèle de buts vers un autre modèle de buts via des transformations automatiques, en gardant la consistance et la traçabilité. Dans (Niu *et al.*, 2009), est proposé l'utilisation d'une plateforme permettant de tracer les aspects depuis les buts jusqu'à l'implémentation. La plateforme fournit un langage support pour modéliser les buts aspects et des mécanismes pour les transformer en programmes orientés aspects. Cette approche utilise des transformations de type modèle vers code, tandis que la notre utilise des transformations modèle vers modèle. Un état de l'art des approches d'ingénierie des modèles en rapport avec les exigences non fonctionnelles est traité par (Ameller *et al.*, 2010) : en général, elles n'y sont pas supportées. Un cadre intégrant les exigences non fonctionnelles dans le cœur du processus de transformations de modèle est décrit, mais aucune approche à part entière n'est proposée. Dans (Patricio *et al.*, 2011), un langage extensible permettant d'unifier et de représenter les langages GOREs est proposé. Le traitement unifié des concepts du modèle permet de définir plus de projections de chaque problème que les langages traditionnels. En outre des modèles hybrides peuvent être utilisés. Cependant, si l'adoption d'un langage unifié n'est pas possible, une approche basée modèle reste possible.

Pour résumer, notre approche diffère de celles présentées précédemment par le fait qu'elle utilise un modèle créatif (*Mind Map*), plus proche des experts domaine et

des utilisateurs finaux, avant de générer les modèles d'exigence, en rendant ainsi le processus de capture des exigences plus agile.

5. Conclusion

L'article définit une approche de recueil des exigences orientée modèle permettant de générer systématiquement des modèles SysML à partir de modèles KAOS, eux-même générés à partir de modèles *Mind Maps*, en utilisant les techniques de métamodélisation et de transformations de modèles. Un processus agile systématique permettant la participation active des clients a été défini et appliqué à un cas d'étude réel. La majeure contribution de ce travail est de fournir une plateforme pour faciliter la communication entre les développeurs et les experts du domaine en utilisant *Mind Map*, et de rendre plus précis, consistants et traçables les modèles utilisés, des exigences à l'analyse et la conception, par l'adoption des techniques et méthodes de l'ingénierie des modèles comme part entière de cette plateforme.

Comme travail futur, nous comptons terminer l'implémentation complète du processus de transformation en ATL, pour concevoir d'une part un protocole empirique permettant d'évaluer la compréhensibilité des *Mind Map*, et d'autre part pour inclure les *Mind Maps* comme outils permettant de capturer les opérations à inclure dans les modèles KAOS et automatiser la partie assemblage de ces modèles. Nous accompagnerons l'entreprise afin de comparer cette approche à une approche plus traditionnelle et évaluer ainsi son apport. Enfin, dans ce contexte, l'utilisation des méthodes formelles basées sur des techniques classiques de test et de preuve, devient incontournable pour certifier et valider les transformations de modèles présentées dans ce papier.

6. Bibliographie

Ahmad M., « Modeling and Verification of Functional and Non Functional Requirements of Ambient, Self-Adaptive Systems », PhD thesis, University of Toulouse, 2013.

Ameller D., Franch X., Cabot J., « Dealing with Non-Functional Requirements in Model-Driven Development », *IEEE Int. Requirements Engineering Conf.*, 2010, p. 189-198.

Antón A. I., McCracken W. M., Potts C., « Goal Decomposition and Scenario Analysis in Business Process Reengineering », *Proceedings of Advanced Information Systems Engineering*, CAiSE '94, London, UK, 1995, Springer-Verlag, p. 94–104.

Buzan T., Buzan B., *The Mind Map Book*, BBC Books, London, 2003.

Castro J., Kolp M., Mylopoulos J., « Towards Requirements-Driven Information Systems Engineering : the Tropos Project », *Information Systems*, vol. 27, n° 6, 2002, p. 365 - 389.

Dardenne A., van Lamsweerde A., Fickas S., « Goal-Directed Requirements Acquisition », *Science of Computer Programming*, vol. 20, n° 1-2, 1993, p. 3 - 50.

Davidson M. J., Dove L., Weltz J., « Mental Models and Usability », *Cognative Psychology 404*, Depaul University, novembre 1999.

Kleppe A. G., Warmer J., Bast W., *MDA Explained : The Model Driven Architecture : Practice and Promise*, Addison-Wesley, 2003.

Lamsweerde A. V., Letier E., « From Object Orientation to Goal Orientation : A Paradigm Shift for Requirements Engineering », *Workshop on Radical Innovations of Software and System Engineering*, LNCS, Springer-Verlag, 2003, p. 4–8.

Lawrence Chung Brian A. Nixon E. Y. J. M., *Non Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Massachusetts, USA, 2000.

Matulevicius R., Heymans P., « Analysis of KAOS Meta-model », rapport, 2005, University of Namur, Belgium.

Mernik M., Heering J., Sloane A. M., « When and How to Develop Domain-specific Languages », *ACM Comput. Surv.*, vol. 37, n° 4, 2005, p. 316–344, ACM.

Monteiro R., Araujo J., Amaral V., Goulao M., Patricio P., « Model-Driven Development for Requirements Engineering : The Case of Goal-Oriented Approaches », *8th Int. Conf. on the Quality of Information and Communications Technology*, IEEE Computer, 2012, p. 75–84.

Moody D., Sindre G., Brasethvik T., Solvberg A., « Evaluating the Quality of Information Models : Empirical Testing of a Conceptual Model Quality Framework », *25th Int. Conf. on Software Engineering*, 2003, p. 295-305.

Niu N., Yu Y., González-Baixauli B., Ernst N., Sampaio Do Prado Leite J. C., Mylopoulos J., « Transactions on Aspect-Oriented Software Development », chapitre Aspects Across Software Life Cycle : A Goal-Driven Approach, p. 83–110, Springer-Verlag, 2009.

Object Management Group, « OMG Systems Modeling Language V1.3 », rapport, 2012, Object Management Group.

Patricio P., Amaral V., Araujo J., Monteiro R., « Towards a Unified Goal-Oriented Language », *35th IEEE COMPSAC*, 2011, p. 596–601.

Pressman R. S., *Software Engineering : A Practitioner's Approach*, McGraw Hill, 2006.

Riviére A., Benac C., « MBSE using SysML :A350 XWB Experience », 2010.

Rolland C., Salinesi C., « Modeling Goals and Reasoning with Them », *Engineering and Managing Software Requirements*, p. 189-217, Springer, 2005.

Schmidt D., « Guest Editor's Introduction : Model-Driven Engineering », *Computer*, vol. 39, n° 2, 2006, p. 25-31.

Sommerville I., *Software Engineering*, Addison-Wesley, 9th édition, 2010.

University of Toronto, « GRL - Goal-oriented Requirement Language », Toronto, Canada.

van Lamsweerde A., « Goal-Oriented Requirements Engineering : a Guided Tour », *5th IEEE Int. Symp. on Requirements Engineering*, 2001, p. 249-262.

Wanderley F., Da Silveira D. S., Araujo J., Lencastre M., « Generating Feature Model from Creative Requirements Using Model Driven Design », *16th Int. Software Product Line Conference - Volume 2*, ACM, 2012, p. 18–25.

Wanderley F., Araujo J., « Generating Goal-Oriented Models from Creative Requirements using Model-Driven Engineering », *Int. Work. on Model-Driven Requirements Engineering*, 2013, p. 1-9.

Yu E., « Modelling Strategic Relationships for Process Reengineering », PhD thesis, Graduate Depart-ment of Computer Science, University of Toronto, 1995.