École Centrale de Nantes **École des Mines de Nantes** **Université de Nantes**

MASTER AUTOMATIQUE, ROBOTIQUE ET INFORMATIQUE APPLIQUÉE

Spécialité : Temps réel, Conduite et Supervision

Année 2012–2013

Mémoire bibliographique

Présenté et soutenu par :

Maxime Alay-Eddine

le Mardi 12 Février 2013

à l'Institut de Recherche en Communications et Cybernétique de Nantes

TITRE

Analyse sémantique de langages de haut niveau

Conception et vérification des systèmes embarqués

En partenariat avec
SAGEM Défense Sécurité - Groupe Safran

JURY

Président :	**Jean-François Lafay**	**Professeur des universités, École Centrale de Nantes**
Directeur :	**Olivier-Henri Roux**	**Professeur des universités, École Centrale de Nantes**
Laboratoire :	**Institut de Recherche en Communications et Cybernétique de Nantes – UMR CNRS 6597**	

Résumé

Analyse sémantique de langages de haut niveau

Les moyens de communication font partie des technologies les plus anciennes, celles dont l'histoire est intégralement liée à celle de l'humanité. Cette épopée a commencé avec l'invention de l'écriture, puis de l'imprimerie et du téléphone, pour finalement atteindre notre époque du tout numérique et de la dématérialisation.
Mais peut-on dire que nous disposons d'un moyen de communication efficace, et ce après plus de 12000 ans d'aventure humaine ?
Dispose t-on de langages efficaces pour décrire l'information et existe t-il des systèmes de vérification de ces modèles ?

Ce document présente les résultats d'un séminaire bibliographique réalisé autour des langages de haut niveau, destiné à répondre à ces questions. Nous étudierons en particulier Unified Modeling Language, Systems Modeling Language, et Architecture Analysis and Design Language, mettant en avant leurs points forts, et leurs points faibles. Nous ferons également l'étude de leurs interactions, et terminerons par une partie dédiée à la vérification de modèles écrits avec ces langages.

Remerciements

Je souhaite remercier Messieurs Olivier-Henri Roux, Jean-Yves Martin, Morgan Magnin, Didier Lime et Pierre Molinaro pour leur soutien dans ce Master, et pour m'avoir permis de l'effectuer en parallèle de ma troisième année à l'Ecole Centrale de Nantes.

Acronymes

AADL Architecture Analysis and Design Language

DSL Domain Specific Language

Fiacre Format Intermédiaire pour les Architectures de Composants Répartis Embarqués

INCOSE International Concil on Systems Engineering

LTL Linear Temporal Logic

MBE Model-Based Software Systems Engineering

OMG Object Modeling Group

S/E-LTL State/Event Linear Temporal Logic

SAE Society of Aumotive Engineers

SysML Systems Modeling Language

Tina Time Petri Net Analyzer

Topcased The Open-Source Toolkit for Critical Systems

UML Unified Modeling Language

Table des matières

Chapitre 1

Introduction

Concevoir un système est une opération complexe faisant intervenir de nombreux acteurs. Choisir et employer les outils adaptés à la capitalisation des informations issues de cette étape est un important facteur de réussite, encore faut-il pouvoir ensuite diffuser ces connaissances aux différents membres de son équipe.
En 1989, de nombreux industriels se sont rassemblés pour travailler sur ce problème, et définir un système standard de modélisation et de communication orienté objet. Ce consortium, formé de Hewlett-Packard, IBM, Sun Microsystems, Apple Computer, American Airlines et Data General, créa ainsi en 1989 l'Object Modeling Group, organisme qui regroupe maintenant plus de 2000 entreprises [1].

En parallèle, dès la fin des années 50 émergeait la théorie du "Model Checking" (ou vérification de modèles) de E. Clarke et A. Emerson [2], récompensés en 2007 par le prix Turing. Le nom en lui-même est évocateur : le Model Checking s'applique à des modèles déjà établis, et nous comprenons dès lors qu'une nouvelle problématique se pose : peut-on définir des modèles à la fois compréhensibles par l'homme et pour les machines ? Est-il possible d'effectuer une analyse formelle sur des langages type UML [1], SysML [2] ou AADL [3] ?

Travaillant en tant qu'apprenti ingénieur dans le groupe Safran, je me concentrerai sur cette problématique dans le cadre du Master ARIA, et l'appliquerai à un projet industriel : étude et réalisation d'un enregistreur de données pour centrales inertielles.

1. Unified Modeling Language
2. Systems Modeling Language
3. Architecture Analysis and Design Language

Chapitre 2

Analyse des langages

2.1 Démarche

La première partie de mon travail de Master sera de concevoir et modéliser un système embarqué à partir d'un ensemble de contraintes définies dans un cahier des charges. Le but étant de pouvoir partager les informations issues de cette phase de conception et de respecter la politique du groupe Safran, j'ai d'abord effectué une analyse des langages employés par l'entreprise, à savoir UML et SysML. Des recherches sur des langages dérivés m'ont ensuite poussé à m'intéresser à AADL, que je présenterai ainsi à mon employeur. Ce chapitre présente les résultats de ces études.

2.2 UML

2.2.1 Présentation

Créé en 1995 par Booch, Rumbauch et Jacobson, UML est un langage orienté objet de haut niveau, dont le but est de fournir des outils d'analyse, de conception et d'implémentation pour les systèmes à dominante logicielle [3]. Basé sur des composantes visuelles, UML est un langage extrêmement graphique et générique, pouvant couvrir un large champ d'applications pour lesquels tous les modules ne sont pas forcément utiles.

Le processus d'évolution d'UML repose sur l'analyse des différents modèles utilisés par les professionnels, qui proposent ensuite des éléments d'amélioration pour construire une solution unifiée [4] ; ce système permet de réduire le temps de déploiement des mises à jour au sein des entreprises utilisant déjà ce standard, tout en assurant une démarche pragmatique, basée sur l'expérience de la communauté UML.

2.2.2 Description

UML est constitué de quatorze diagrammes répartis en deux catégories [3] :

Diagrammes structurels Ils décrivent l'architecture du système modélisé.

Diagramme de classe Décrit le système à partir de ses classes et de leurs attributs. On y retrouve donc l'essence même des architectures orientées objet.

Diagramme de composants Décrit les composants du système et leurs dépendances.

Diagramme d'objets Décrit l'architecture orientée objet du système à un instant donné.

Diagramme de profil Agit comme un meta-modèle permettant de simplifier la représentation des structures semblables.

Diagramme de structure composite Décrit la structure interne d'une classe et les interactions possibles.

Diagramme de déploiement Décrit l'architecture matérielle et les environnements logiciels présents sur ces derniers.

Diagramme de paquetage Décrit le système en tant que groupes logiques interdépendants.

Diagrammes de comportement Ils décrivent les interactions et comportements du système modélisé.

Diagramme de séquence Décrit les interactions entre les sous-parties du système étudié, en tant que séquences de messages.

Diagramme de temps Décrit le comportement du système avec ses contraintes temporelles.

Diagramme d'activité Décrit les flux de contrôles du système et décompose les flux de travail de façon atomique.

Diagramme global d'interaction Semblable au diagramme d'activité, mais indique également l'organisation spatiale des acteurs d'une interaction.

Diagramme de cas d'utilisation Décrit les fonctionnalités du système, les acteurs, ainsi que les interactions et buts des échanges entre ces interlocuteurs.

Diagramme d'états-transitions Décrit le système comme un ensemble à états-transitions type automate fini.

Diagramme de communication Permet de synthétiser les informations contenues dans les diagrammes de cas d'utilisation, de séquence, et de classe.

Faire le modèle UML d'un système, c'est donc choisir le(s) diagramme(s) le(s) plus pertinent(s) pour en décrire le fonctionnement et l'architecture.

2.2.3 Analyse

Bien qu'UML soit désormais un standard flexible utilisé *de facto* dans la plupart des entreprises, ce langage a de nombreux défauts [5][6] :

- Peu de personnes sont capables d'écrire un diagramme UML. Cela est dû à la richesse de ce langage et à sa généricité. La norme indiquée en référence [3] fait par exemple 748 pages, ce qui entraîne une courbe d'apprentissage très élevée. Nous noterons cependant qu'une personne ne connaissant pas UML sera capable de déchiffrer la plupart des informations contenues dans

un diagramme respectant cette norme. La représentation de l'information est donc ici plus complexe que sa diffusion.
- UML ne dispose pas d'un vrai formalisme. Ce manque provient de la conception même du langage, volontairement générique, mais rend difficile les analyses formelles et les transformations type modèle-code.
- Une modélisation UML implique la réalisation de nombreux diagrammes, et noie facilement les interlocuteurs dans l'information.

Pour résumer, le langage UML fournit des bases solides pour modéliser un système, mais sera adapté aux besoins de chaque entreprise, qui reprendra uniquement les diagrammes les plus intéressants et les complètera avec ses propres éléments sémantiques.

2.3 SysML

2.3.1 Présentation

SysML est un langage très jeune. Créé en 2001 par l'OMG [1] et l'INCOSE [2], son objectif est d'ajouter une extension standardisée à UML destinée aux ingénieurs systèmes [7]. Cette extension permet d'aborder des systèmes plus éloignés du génie logiciel et de modéliser les contraintes de conception non applicables à celui-ci.

2.3.2 Description

SysML reprend de nombreux diagrammes du langage UML, et y ajoute de nouveaux éléments [8] :

Diagramme de cas d'utilisation Identique en UML et en SysML, il modélise les fonctionnalités que le système doit fournir. Le cas d'utilisation est une unité fonctionnelle utilisée pour la description et la recette du système.
Equivalent UML : Diagramme de cas d'utilisation

Diagramme de séquence Identique en UML et en SysML le diagramme de séquence modélise la chronologie des interactions entre les éléments du système ou entre le système et l'extérieur.
Equivalent UML : Diagramme de séquence

Diagramme d'activité Même utilisation en UML et en SysML. Le diagramme d'activité modélise les flux d'informations et les flux d'activité du système.
Equivalent UML : Diagramme d'activité

Diagramme d'états-transitions Identique en UML et en SysML, il représente les différents états que peut prendre un élément ou une opération ainsi que ses réactions aux évènements extérieurs.
Equivalent UML : Diagramme d'états-transitions

Diagramme de définition de bloc Le diagramme de définition de bloc en SysML est semblable au diagramme de classe en UML. Il donne une représentation statique des entités du système, de leurs propriétés, de leurs

1. Object Modeling Group
2. International Concil on Systems Engineering

opérations et de leurs opérations.
Equivalent UML : Diagramme de classe

Diagramme de bloc interne Le diagramme de bloc interne SysML et le diagramme de structure composite UML donnent une représentation « Boîte blanche » qui matérialise les imbrications des parties et leurs interconnections par les ports.
Equivalent UML : Diagramme de structure composite

Diagramme de paquetage Le diagramme de paquetage montre l'organisation générale du modèle en UML comme en SysML. En SysML il sert en plus à donner différentes vues du système.
Equivalent UML : Diagramme de paquetage

Diagramme paramétrique Ce diagramme modélise les paramètres physiques du système. Il sert à tester les performances physiques et quantitatives du système.
Introduit avec SysML

Diagramme d'exigence Le diagramme de spécification permet de collecter et d'organiser toutes les exigences textuelles du système.
Introduit avec SysML

Table d'allocation Cette table introduit dans la modélisation des systèmes de notation tabulaire absents d'UML.
Introduit avec SysML

Les autres diagrammes UML ne sont pas utilisés dans SysML.

SysML se présente donc comme un dérivé d'UML venant proposer une solution centrée sur l'ingénierie système. Là où nous avions un langage de modélisation générique, nous obtenons ici un langage plus proche d'un DSL [3] [9].

2.3.3 Analyse

SysML réduit le nombre de diagrammes disponibles et utilise une sémantique propre à l'ingénierie système [9]. En se recentrant sur les besoins de ce domaine, il se positionne comme un outil plus facile à assimiler et à utiliser. Il fournit des moyens de modélisation de systèmes physiques, mais conserve les défauts d'UML en particulier concernant sa sémantique, rendant difficile les opérations de vérification formelle [9].

2.4 AADL

2.4.1 Présentation

Commencé par Peter Feiler en 1999, le langage AADL fut finalement validé et publié en 2004 auprès du SAE [4]. Alors qu'UML était utilisé pour décrire une architecture logicielle, couplé à SysML pour l'architecture matérielle, Feiler souhaitait créer un système de modélisation des interactions entre ces différentes parties pour représenter les systèmes embarqués. Ce domaine fait en effet intervenir trois parties très différentes : mécanique, électronique et informatique.

3. Domain Specific Language
4. Society of Aumotive Engineers

Il décrit ainsi AADL de cette manière [10] : *"Les langages d'architecture ont introduit le concept de composant pour structurer l'architecture d'un système. Les blocs Simulink modélisent la partie commande, les blocs SysML modélisent la partie physique, UML modélise l'architure conceptuelle, et AADL modélise l'architecture des systèmes embarqués"*. Défini sur le modèle MBE [5], AADL apporte un formalisme à la modélisation des systèmes embarqués, ouvrant la voie au Model Checking.

2.4.2 Description

Un modèle AADL est réalisé à partir de composants, répartis en quatre catégories [10] :

Partie logicielle ou software Décrit la partie logicielle du système.

- **Donnée** Donnée à insérer dans le code source et leurs types.
  Aussi noté data.
- **Tâche ou fil** Tâche permettant de gérer les exécutions concurrentes.
  Aussi noté thread.
- **Groupe de tâches** Groupe d'abstraction destiné à organiser les fils de façon logique dans un processus.
  Aussi noté thread group.
- **Processus** Ensemble exécuté.
  Aussi noté process.
- **Sous-programme** Code définissant des méthodes supplémentaires à appeler dans le système.
  Aussi noté subprogram.
- **Groupe de sous-programmes** Permet d'organiser les sous-programmes en bibliothèques.
  Aussi noté subprograms.

Partie matérielle ou hardware Décrit la partie matérielle du système.

- **Processeur** Permet d'ordonnancer et d'exécuter les fils et processeurs virtuels.
  Aussi noté processor.
- **Processeur virtuel** Ressource logique capable d'ordonnancer les threads à des processeurs physiques.
  Aussi noté virtual processor.
- **Mémoire** Stocke le code et les données.
  Aussi noté memory.
- **Bus** Gère les connexions entre processors, memory et devices.
- **Bus virtuel** Représente une communication abstraite comme un protocole d'échange ou un canal virtuel.
  Aussi noté virtual bus.
- **Périphérique** Représente les senseurs et autres composants qui créent les interfaces avec l'environnement extérieur du système.
  Aussi noté device.

5. Model-Based Software Systems Engineering

Partie composite Décrit l'assemblage des composants du système.

Système Intègre le software, le hardware et les autres composants système dans des unités distinctes avec une architecture définie.
Aussi noté system.

Partie générique Décrit des éléments génériques de modélisation.

Abstraction Définit un environnement d'exécution générique et conceptuel pouvant être réduit par la suite à des composants plus précis.
Aussi noté abstract.

2.4.3 Analyse

Un modèle AADL se décrit grâce à un langage textuel, permettant de générer une représentation graphique du modèle, plus facile à lire. Nous définissons d'abord les composants (moules de base du modèle), puis leurs implémentations (briques formées à partir des moules), pour enfin les assembler et former le système. Ce langage possède une sémantique forte garantissant l'équivalence entre les modèles textuels et graphiques, comblant ainsi les lacunes d'UML ou de SysML.

2.5 Synthèse

La problématique de modélisation d'un système ne dispose pas de solution parfaite et unique. Plusieurs langages proposent des éléments de réponse, mais ils ne trouvent leur pleine efficacité qu'une fois employés ensemble sur des domaines spécifiques. L'ingénieur système utilisera UML pour la couche logiciel du système et son environnement opérationnel, SysML pour l'architecture physique, et verra AADL comme un moyen d'unir ces langages et de préparer une analyse formelle de son modèle.

Chapitre 3

Vérification formelle

3.1 Démarche

La vérification formelle consiste à appliquer des méthodes mathématiques à des systèmes pour en vérifier des propriétés. Ces méthodes sont particulièrement importantes dans les domaines comme l'aéronautique, où la moindre erreur peut coûter la vie de centaines de personnes. Une analyse formelle permet de démontrer de façon rigoureuse qu'un système possède certains critères. Une telle analyse sur des sémaphores permet par exemple de démontrer que l'accès concurrent à une ressource dite critique ne se produira pas. Dans le cadre de mon Master, je devrai utiliser de tels outils pour vérifier des modèles réalisés avec UML, SysML, et AADL. L'étude des moyens disponibles pour mener à bien cette tâche m'a mené à la chaîne de compilation *(ou toolchain)* AADL-Fiacre [1]-Tina [2], analysée dans cette partie.

3.2 Fiacre

3.2.1 Présentation

Fiacre [11] est un modèle formel intermédiaire destiné à la représentation des comportements et des aspects temporels des systèmes. Son objectif est de permettre aux outils de vérification formelle *(model-checkers)* tels que Tina d'analyser les propriétés de modèles définis avec des langages haut niveau, comme UML ou AADL. Cet intermédiaire est nécessaire pour conserver la simplicité et l'efficacité des model-checkers, et agit comme un langage pivot décomposant le modèle initial en briques analysables [12].

3.2.2 Description

Fiacre est inspiré des réseaux de Petri temporels, de deux travaux nommés V-Cotre et Ntif, et des dernières avancées sur l'ordonnancement des systèmes temps-réel [11]. Il se base sur deux notions [12] :

1. Format Intermédiaire pour les Architectures de Composants Répartis Embarqués
2. Time Petri Net Analyzer

Les processus Ils décrivent le comportement des composants séquentiels.
Aussi notés process.

Les composants Ils décrivent un système comme une composition de processus.
Aussi notés components.

Fiacre est un langage fortement typé, qui ajoute ainsi des contraintes de sûreté quant aux types manipulés par chaque processus modélisé.

L'opération de transformation de modèle utilisée par Fiacre se base sur la sémantique d'AADL, notamment sur les fils d'exécution *(threads)*. L'idée est d'associer un processus Fiacre à chaque fil AADL, et de les faire communiquer ensemble via un processus annexe appelé *glue*. C'est ce processus glue qui relaiera les informations dans le système avec l'ordonnancement adéquat. Nous obtenons ainsi un nouveau modèle, écrit dans un langage analysable avec Tina. La conversion se fera à l'aide d'un outil comme Topcased [3], un module de l'environnement de développement Eclipse capable de lire et de transformer les modèles AADL.

3.3 Tina

3.3.1 Présentation

Tina[13] est une boîte à outils fournissant des moyens d'édition et d'analyse pour les réseaux de Petri et réseaux de Petri temporels.

3.3.2 Description

Tina est constitué de [13] :
- un éditeur graphique de réseaux ou d'automates
- un outil de construction de comportements
- un outil d'analyse et de vérification formelle.

Nous nous intéresserons en particulier à la partie analyse/vérification avec *selt*, outil développé en 2006 [14] et utilisant une extension de la logique LTL [4] sur les propriétés des états et transitions, nommée S/E-LTL [5][15].

Pour des formules logiques de Tina notées p et q, nous utiliserons les notations suivantes [12] :
- $-p$ correspond à la négation de p
- $p/\backslash q$ correspond à la conjonction de p et q
- $()p$ sera vraie si p est vérifiée à l'état suivant (opérateur *Next*)
- $[]p$ sera vraie si p est vérifiée dans tout état du chemin étudié (opérateur *Always*)
- $<>$ p sera vraie si p est vérifiée dans un état futur du chemin étudié (opérateur *Eventually*)
- pUq sera vraie si p est vérifiée jusqu'à ce que q soit vérifiée (opérateur *Until*)

3. The Open-Source Toolkit for Critical Systems
4. Linear Temporal Logic
5. State/Event Linear Temporal Logic

Nous combinons maintenant avec cette notation des processus Fiacre $process1$, $process2$, possédant les états $w\_process1$, $w\_process2$ (processus en attente), $cs\_process1$, $cs\_process2$ (processus en zone critique) [12] :

- $[](cs\_process1 + cs\_process2 <= 1)$ correspond à l'exclusion mutuelle : les processus 1 et 2 ne peuvent jamais être en même temps dans leur zone critique
- $[]((w\_process1 =><> cs\_process1)/\backslash(w\_process2 =><> cs\_process2))$ correspond à la condition de non-famine : si le processus 1 ou 2 entre en état d'attente, il entrera forcément en zone critique par la suite

Une analyse Tina consiste donc à vérifier une propriété correspondant aux exigences du système modélisé (non-famine, exclusion mutuelle, mais aussi délais bornés...).

3.4 Synthèse

La vérification formelle de modèles UML/SysML/AADL passe donc par l'outil Tina, qui nécessite une traduction intermédiaire du système réalisée avec Fiacre. L'utilisateur souhaitant modéliser son système pourra cependant se munir d'outils appropriés comme Topcased pour ne pas avoir à se plonger dans la syntaxe de Fiacre. Nous pouvons donc distinguer une partie "Front-End", visible pour l'utilisateur, constituée des modèles AADL, Tina/*selt*, et une partie "Back-End", masquée, constituée de Fiacre et des outils annexes lancés par *selt* pour vérifier les propriétés étudiées.

Nous noterons qu'une grande partie des outils de modélisation cités dans cette partie sont utilisables avec l'environnement de développement Eclipse et les modules appropriés, permettant à un nouvel utilisateur de disposer de tous les outils nécessaires à ses travaux sous un seul logiciel. Les modules particulièrement intéressants sont OSATE et Topcased pour UML/SysML/AADL, Topcased assurant également la conversion AADL vers Fiacre. Il suffira alors d'utiliser Tina pour importer les modèles générés et les vérifier.

Ces outils sont néanmoins encore en phase d'amélioration [11][12]. Dans la version actuelle, le nombre de composants doit être défini à l'avance, mais des travaux sont en cours pour proposer de modéliser des systèmes paramétriques [11]. Cela pourrait par exemple être utilisé pour les problèmes de scalabilité, comme pour les nouvelles architectures web où de nouvelles instances de serveur sont lancées afin de gérer le flux des utilisateurs. De plus, il n'est pas encore possible de convertir les architectures multiprocesseurs AADL vers Fiacre ou de gérer les préemptions lors de cette conversion [12]. L'utilisateur prendra donc ces informations en considération lors de son analyse.

Chapitre 4

Conclusion

Nous avons ainsi fait un état de l'art de la modélisation/vérification de modèles UML/SysML/AADL.

Nous avons vu les problèmes liés aux langages de haut-niveau et à leur manque de formalisme, qui impliquent l'utilisation de couches supplémentaires pour faire du Model-Checking. Ils ont cependant une utilité dans la représentation de l'information : il est bien plus simple de lire un diagramme UML que de lire un diagramme AADL, encore plus simple de comprendre un tel diagramme que de lancer une vérification avec Tina. Les pré-requis ne sont également pas les mêmes : la lecture d'un diagramme UML ne demande aucune connaissance particulière, tandis que Tina demande une certaine maîtrise de LTL, ou qu'AADL demande des connaissances poussées dans l'architecture système. La chaîne de modélisation complète fait ainsi intervenir différents langages, tirant profit des qualités de chacun pour masquer les défauts des autres.

La chaîne de conception croît donc en complexité à mesure que l'on s'approche du Model-Checking, point sur lequel des travaux sont en cours pour rendre abordable la vérification sans connaissances préliminaires.

L'étude de ces modèles m'aura en particulier fait relever les problèmes de modélisation de la scalabilité, domaine stratégique pour de nombreux secteurs qu'il s'agisse de l'aéronautique ou du web. Je mènerai donc au cours de mon Master une veille technologique approfondie sur ce sujet, prenant note des difficultés rencontrées en entreprise, et contactant les différents acteurs susceptibles de m'aider à les résoudre. A l'heure où les architectures se délocalisent avec le Cloud-Computing, trouver un équilibre entre le nombre d'instances matérielles utilisées et la charge reçue reste encore le résultat d'une démarche expérimentale : le Model-Checking y a donc une place à prendre.

Bibliographie

[1] Allison June Barlow Chaney. Object management group. `http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Object_Managemen_Group.html`. [En ligne ; dernier accès 04 Février 2013].

[2] Armin Biere. Tutorial on model checking modelling and verification in computer science. `http://fmv.jku.at/papers/Biere-AB08.pdf`, 2008.

[3] Omg unified modeling languagetm (omg uml), superstructure version 2.4.1, 2011.

[4] P.A. Muller and N. Gaertner. *Modélisation objet avec UML*, volume 514. Eyrolles, 2000.

[5] Lispy. Why uml fails to add value to the design and development process. `http://lispy.wordpress.com/2008/10/29/why-uml-fails-to-add-value-to-the-design-and-development-process/`. [En ligne ; dernier accès 04 Février 2013].

[6] Yohann Poiron. Les raisons d'utiliser ou non l'uml dans vos phases de conception pour un développeur. `http://www.blog-nouvelles-technologies.fr/archives/1038/les-raisons-dutiliser-ou-non-luml-dans-vos-phases-de-conception-pour-un-developpeur/`. [En ligne ; dernier accès 04 Février 2013].

[7] T. Weilkiens. *Systems engineering with SysML/UML : modeling, analysis, design*. Morgan Kaufmann, 2008.

[8] Pascal Roques. Comparatif des diagrammes uml et sysml. `http://www.uml-sysml.org/sysml`. [En ligne ; dernier accès 09 Décembre 2012].

[9] Hélène Mollère Youssef Srour Raphaël Sudre Karen Lavignasse, Nathalie Lépine. Génération de fichiers de paramétrage. 2007.

[10] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL - An Introduction to the SAE Architecture Analysis and Design Language*. SEI series in software engineering. Addison-Wesley, 2012.

[11] Bernard Berthomieu, Jean-Paul Bodeveix, Patrick Farail, Mamoun Filali, Hubert Garavel, Pierre Gaufillet, Frederic Lang, and François Vernadat. Fiacre : an Intermediate Language for Model Verification in the Topcased Environment. In *ERTS 2008*, Toulouse, France, 2008.

[12] Bernard Berthomieu, Jean-Paul Bodeveix, Christelle Chaudet, Silvano Zilio, Mamoun Filali, and François Vernadat. Formal verification of aadl specifications in the topcased environment. In *Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies*, Ada-Europe '09, pages 207–221, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool tina – construction of abstract state spaces for petri nets and time petri nets. *International Journal of Production Research*, 42(14) :2741–2756, 2004.

[14] Bernard Berthomieu. The tina toolbox home page. `http://projects.laas.fr/tina//news.php`. [En ligne ; dernier accès 09 Décembre 2012].

[15] S. Chaki, E. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/event-based software model checking. In *Integrated Formal Methods*, pages 128–147. Springer, 2004.