Une méthodologie de modélisation multi-modèles distribués par métier pour les systèmes embarqués

Theurer Wolfgang

THÈSE

présentée à l'École Nationale Supérieure des Ingénieurs des Études et Techniques d'Armement
En vue de l'obtention du titre de

DOCTEUR
de
l'École Nationale Supérieure de l'Aéronautique et de l'Espace

Spécialité : Informatique fondamentale et parallélisme

par

Wolfgang THEURER

Une méthodologie de modélisation multi-modèles distribuée par métier pour les systèmes embarqués

Soutenue publiquement le 13 décembre 2006

Jury :

Jean-Pierre ELLOY	Professeur, IRCYN	Rapporteur
Jean-Marc JEZÉQUEL	Professeur, IRISA	Rapporteur
Frédéric BONIOL	Professeur, INPT - ENSEEIHT	Co-directeur
Philippe DHAUSSY	Enseignant chercheur, ENSIETA	Co-directeur
Yamine AIT AMEUR	Professeur, LISI/ENSMA	Président
Joël CHAMPEAU	Enseignant chercheur, ENSIETA	Examinateur
Yvon KERMAREC	Enseignant chercheur, ENST - Bretagne	Examinateur
Claire PAGETTI	Ingénieur de recherche , ONERA - CERT	Examinatrice

Thèse réalisée au laboratoire DTN de l'ENSIETA (Brest) sous la direction de MM. Frédéric Boniol et Philippe Dhaussy.

Remerciements

Un travail de thèse comme celui-ci n'est pas l'œuvre d'un seul homme, de même qu'il ne débute pas à l'instant où l'Histoire[1] le fera commencer. Le cheminement et l'histoire individuelle des personnes qui y ont contribué sont aussi importants que la phase de rédaction. Je dois beaucoup à de nombreuses personnes et je vais tenter de leur exprimer ma gratitude en quelques lignes tout en tentant de restituer cette thèse dans son contexte.

Mais comment procéder ? Par ordre d'importance ? Impossible d'identifier un ordre (même partiel)... Par ordre alphabétique ? Pourquoi pas, mais le travail semble ardu...

Reste l'ordre chronologique... Allons-y donc pour un ordre inversement chronologique d'apparition dans ma courte vie.

J'adresse un grand merci aux membres de mon jury, qui se sont déplacés, m'ont écouté avec suffisamment d'attention pour me poser des questions parfois gênantes, mais toujours pertinentes. Bien entendu, j'adresse des remerciements tous particuliers à mes rapporteurs, Jean-Pierre Elloy et Jean-Marc Jézéquel, pour avoir accepté la lourde de tâche de rapporter mon mémoire -comme il est difficile de lire une thèse de façon assez profonde pour finalement pouvoir la juger !-. Merci, donc pour le temps passé, vos commentaires avisés et votre jugement sans complaisance qui donnent toute la valeur et la légitimité de mon travail. Merci à Yamin Ait Ameur, Claire PAGETTI, Yvon Kermarec et Joël Champeau pour leurs questions lors de ma soutenance.

Merci à Hugues BONNIN pour toutes nos discussions passionnantes. Merci pour toutes ces activités que tu acceptes de partager avec moi et ce n'est que le début... D'une manière générale, merci à tous mes collègues et amis de CS-SI qui m'ont accueilli malgré cette thèse encore en cours et les inévitables crises d'angoisse qui en découlaient...

Merci à toi Sophie, qui partage ma vie depuis 3 ans... Comment te remercier de la patience sans borne dont tu as fait preuve à l'égard d'un thésard cyclothymique et acariâtre. J'espère que la délivrance de la soutenance me révélera comme l'homme charmant que tu mérites, mais j'ai bien peur d'avoir toujours une excuse pour grogner... L'avenir nous le dira. Merci aussi à toi et à Wiosna d'avoir relu ce document en faisant une traque sans pitié aux fautes de syntaxes, et autres barbarismes... Merci à toutes les deux pour cela aussi...

A vous, mes camarades de labo, merci pour ces trois ans de franche rigolade. Merci aussi pour les aspects plus sérieux de notre collaboration. En particulier François et Joël avec qui j'ai guerroyé sur le front méta-méta... Roderic, je me souviendrai toute ma vie de notre collaboration dans l'enseignement, le sauvetage de mondes et notre étude du comportement social des otaries. Merci aux étudiants de l'ENSIETA à qui j'ai eu l'honneur d'enseigner, de m'avoir supporté (croyez-moi ce n'est pas simple).

[1] NDLR : ne faites pas attention, si ça lui fait plaisirs...

Dans mon voyage dans le temps, je crois que j'en arrive au moment où je me suis rendu à Toulouse pour rencontrer mes futurs encadrants de thèse, Frédéric BONIOL et Philippe DHAUSSY...

Dire que sans vous rien de tout cela n'aurait été possible est une banale évidence indigne de moi[2]... Merci du temps que vous m'avez consacré parfois pris sur votre vie familiale, merci de votre soutient, merci pour nos quelques oppositions qui furent autant de stimulations et de remise en question nécessaires, c'est un honneur d'avoir été votre élève.

J'aimerai aussi remercier le Pr André RASPAUD du LaBRI qui m'a réellement donné l'amour de la recherche durant ma maîtrise et mon DEA. Merci donc professeur, car bien qu'au final mes recherches m'aient amené très loin des colorations fortes, vous êtes pour beaucoup dans leur aboutissement.

Et puisque nous sommes arrivés à l'époque de mes études bordelaises, merci à Julien et Fabrice pour les mini GT et les apéros-math, merci à OB et Julien (encore) pour le temps passé sur Bonnie[3], et pour les bons moments passés ensemble.

Merci à vous, bien sûr, mes amis de toujours Philippe, Philippe, Sophie, Niels, Diane, Aymeri... vous savoir sur cette terre est un réconfort en soi. Merci donc d'exister, tout simplement. Philippe et Wiosna, merci pour l'honneur que vous me faites d'être le parrain de votre enfant, que je remercie d'avance de ne pas trop vous faire tourner bourrique (et paf, pas encore né et déjà cité dans une thèse, il ira loin cet enfant).

Merci à mes parents pour tout ce que vous m'avez offert, la vie, votre amour, votre soutient... merci pour tout, de tout coeur et à jamais. Merci aussi à Marie et Maurice qui les accompagnez maintenant et qui n'êtes pas en reste de soutient ni de conseils.

Pardon à tout ceux que j'ai pu omettre ici, je ne vous oublie pas pour autant.

Encore un mot avant de vous laissez savourer la lecture de cet incontournable document [4] : ne voyez pas dans le ton quelque peu badin du présent texte un quelconque manque de conviction dans mes remerciements. Bien au contraire, à l'instar de Frédéric DARD, je traite les affaires sérieuses avec légèreté et les affaires légères avec le plus grand sérieux...

Wolfgang Theurer
Toulouse, le 10 février 2007

[2] NDLR : ça ne s'arrange pas

[3] NDLR : ne voyez rien de scabreux là dedans ! L'auteur parle de son projet TER de maîtrise, le choix des mots étant bien malheureux je vous l'accorde

[4] NDLR : non mais c'est pas vrai, le voilà qui recommence !

DON’T PANIC ![5]

The hitchhicker’s guide to the galaxy, Megadodo Publications, Ursa Minor

[5] Warning : the selected font (“Huge-friendly-letters”) is not available on this planet

Table des matières

Première partie

Introduction

Introduction

"There are two ways of constructing a software design : one way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies."
Charles Antony Richard Hoare.

Les systèmes embarqués modernes, tels que le système avionique de l'Airbus A380, le Système de Navigation et d'attaque d'un avion d'armes ou d'un navire militaire, sont des systèmes de plus en plus complexes, critiques, désormais à logiciels prépondérants, et souvent plongés dans un milieu physique perturbateur voire hostile (cf. phénomène électromagnétique par exemple). La conception de tels systèmes nécessite de ce fait le concours de plusieurs équipes spécialisées dans des domaines différents : la sûreté de fonctionnement, la sécurité informatique, les performances temps réel, le comportement électromagnétique. Compte-tenu de la complexité globale du système, toutes ces équipes ne peuvent avoir chacune qu'un « point de vue » partiel du système qu'elles doivent pourtant concourir à définir, spécifier, paramétrer, réaliser, tester et enfin faire fonctionner. Cet éclatement des compétences conduit au fait qu'il n'existe plus de définition globale, complète et *détaillée* du système intégrant tous ces aspects. La conséquence évidente en est une plus grande difficulté du processus de conception du système, et un risque accru d'erreurs ou d'incompréhensions :
1. d'une part entre les équipes de conception elles-mêmes qui bien que croyant parler du même système peuvent élaborer des modèles par "point de vue" incohérents ou contradictoires
2. et d'autre part des risques d'erreurs ou d'incompréhensions entre le maître d'œuvre et le maître d'ouvrage (le client) qui exprime souvent son cahier des charges (les exigences du système) sans différencier les points de vue.

Deux points paraissent alors fondamentaux :
1. la maîtrise du raffinement d'un ensemble d'exigences systèmes en exigences élémentaires typées par point de vue
2. la maîtrise des relations entre points de vue, et notamment la définition d'une relation de cohérence entre ces points de vue.

Les objectifs de cette thèse sont, tout d'abord d'identifier les problèmes posés par la multiplication des intervenants dans la conception des systèmes embarqués puis de proposer un support

méthodologique basé sur l'utilisation de modèles semi-formels pour répondre à ces différentes problématiques. Ce travail se pose comme une étude en largeur ayant pour vocation de proposer un cadre sur lequel peuvent se greffer des techniques existantes ou à venir répondant à des problèmes précis survenant lors de la conception d'un système embarqué temps-réel.
Pour ce faire nous nous intéresserons au point de vue et aux besoins du maître d'œuvre (ou du chef de projet) qui seul possède une vue globale du système. Nous introduisons un modèle "pivot" du système qui lui est spécialement dédié ainsi que des modèles "d'interaction" entre le maître d'œuvre et les autres intervenants. Enfin, nous apportons un ensemble de modèles d'intégration permettant de gérer l'intégration continue des sous-parties du système développées par les intervenants garantissant ainsi la cohérence du tout au niveau le plus abstrait possible.

Chapitre 1

Problématique : Modélisation distribuée des grands systèmes

"Coming together is a beginning, staying together is progress, and working together is success."
Henry Ford

1.1 Problématique industrielle

La pratique industrielle, dans le cadre des grands projets et quelle que soit la souplesse du cahier des charges initial, peut se décrire de la manière suivante : dans un premier temps, la mise en oeuvre du projet, son démarrage effectif une fois le choix des intervenants effectué, se présente sous la forme d'un dialogue entre toutes les parties, directement ou par maître d'oeuvre interposé. Dans tous les cas, l'issue est la même : les réunions s'enchaînent afin de décider de la distribution des responsabilités, en terme de spécifications, à chacune des parties prenantes. Ce processus consiste à évaluer la cohérence entre les différents éléments de solution (généralement relativement abstraits) proposés par les parties en présence. Le résultat de ces dialogues est un modèle de "blocs" en interaction, dont les interfaces doivent être définies. Une fois cela fait, chaque intervenant réalise sa partie du contrat souvent sans plus d'interaction avec les autres parties, en dehors du maître d'oeuvre. Surviennent ensuite les phases d'intégration et de test, qui peuvent révéler des problèmes non prévus lors de la conception, dont certains peuvent amener à remettre en cause toute l'architecture du système. Les points faibles de cette démarche industrielle sont liés à sa nature empirique : la bonne marche du projet repose principalement non seulement sur l'expérience technique de ceux qui le mènent, mais aussi sur la capacité qu'ils ont à dialoguer. Si la démarche première de mise en avant des compétences n'est sûrement pas à remettre en cause, chacun étant certainement un spécialiste dans son domaine, la capacité supposée des intervenants à se communiquer des informations

pertinentes sur leurs éléments de solution respectifs est encore à démontrer. La cohérence du système final dépend, de manière parfois critique, de la cohérence des décisions prises dans cette phase liminaire de conception, qui reste pourtant relativement informelle. Cependant, le manque de politique de conservation des relations vérifiées à cet instant, entraînera nécessairement une impossibilité de les vérifier ultérieurement, du moins pas de manière automatique. Or c'est bien le moins si l'on souhaite garantir la cohérence. Aucune traçabilité n'est donc possible dans ce cadre, en dehors du recours à la mémoire collective. Il est toujours possible de revenir à une version antérieure si cela a été prévu, mais pas de connaître les choix qui y ont mené. Afin de pouvoir vérifier à chaque instant, et donc garantir la cohérence du système tout au long de son cycle de vie, il est nécessaire d'intégrer la distribution par équipe-métier dans le processus de développement et de formaliser le stockage et l'échange d'informations entre ces équipes, aussi bien sur le fond que sur la forme. En effet, que ce soit dans un cadre de sous-traitance interne ou externe, la réalisation d'un (sous)système passera par le travail commun de collaborateurs regroupés en équipes selon leur spécialité. Les interactions décrites plus haut se référaient donc à un dialogue entre différentes équipes de spécialistes, que nous dénommerons dorénavant des "équipe-métier" ou "facettes".

1.2 Éléments de solutions : État de l'art

Pour des besoins d'abstraction et de maîtrise de la complexité, l'ingénierie logicielle se tourne vers l'utilisation de modèles. En effet, les systèmes embarqués se sont énormément complexifiés et le logiciel a pris une part de plus en plus importante dans ces systèmes. L'industrie automobile en est un exemple significatif puisque aujourd'hui des réseaux embarqués parcourent les véhicules et inter-connectent des dizaines calculateurs (ABS, ESP, contrôle de l'injection, etc). Le domaine de l'avionique civile qui jusqu'à lors limitait au maximum la complexité et la taille de ses composantes logiciel pour des raisons de criticité vit actuellement une révolution technologique avec l'adoption des IMA[1] (Avioniques Modulaires Intégrées) qui autorisent, entre autres, des calculateurs multi-tâches et des bus multiplexés, introduisent par conséquent des ordonnanceurs et des protocoles de communication temps-réel complexes.
Pour maintenir une qualité en terme de performance, de sûreté ou de validité, de nombreux formalismes de modélisation se sont développés pour permettre une analyse *a priori* ou faciliter les tâches de mise au point *a posteriori*. Au sein d'une même application logicielle, nous devons intégrer de nombreuses caractéristiques qui se répartissent entre les entités liées aux besoins de l'utilisateur, celles liées aux choix architecturaux et celles venant de la plate-forme finale. Pour ces différentes catégories de caractéristiques et d'entités, nous pouvons avoir différents modèles ou plusieurs composantes dans un même modèle. On obtient alors une répartition par domaine ou par aspect qui est une tendance forte des modèles de logiciels. Dans ce cas, chaque domaine nécessite une définition non ambiguë qui s'appuie sur une approche semi-formelle ou formelle. Bien sûr, si de nombreux domaines existent avec des définitions différentes, nous avons besoin d'unifier les différentes démarches et proposer des approches permettant le passage entre domaines. Nous présentons maintenant un rapide tour d'horizon des différentes techniques basées sur l'utilisation de modèles dont nous nous sommes inspirés dans le cadre de cette thèse.

1.2.1 Approches multi-modèles

[1] Integrated Modular Avionics

Un cadre commun : MDE

L'ingénierie dirigée par les modèles (ou MDE (Model Driven Engineering)) introduite par Stuart Kent [Ken02] répond au problème du développement de systèmes en proposant l'utilisation de modèles comme la partie fondamentale du développement. Idéalement le concepteur ne construit que des modèles du système, et par transformations successives automatiques ou semi-automatiques aboutit au système concret final. Cette approche offre de nombreux avantages : d'une part les modèles étant une abstraction du système, ils sont plus aisés à manipuler et à comprendre que le code. D'autre part, elle permet d'exprimer toutes les parties du système dans un langage commun et standardisé connu de tous les intervenants (UML (Unified Modeling Language) [BF00], HRT-HOOD [CW96]...), donc de masquer aux non-spécialistes d'un domaine les obscures arcanes des langages dédiés à d'autres domaines. Enfin l'utilisation de modèles, en masquant les détails d'implantation, simplifie la modularité et la réutilisation de code.
Le lecteur interessé pourra trouver dans [JBJ04] une description précise de l'approche MDE ainsi qu'une réflexion sur son insertion dans le paysage technologique actuel et futur.
Nos travaux s'insèrent dans cette mouvance tout en tentant d'adapter ces préceptes de développements principalement destinés au développement de logiciels, au cadre plus général de l'ingénierie des systèmes en y ajoutant d'autres aspects tels que la prise en compte de la plate-forme matérielle.

Le modèle d'architecture logiciel "4+1" vues

P. Kruchten dans [Kru95], présente un modèle pour décrire l'architecture des systèmes à logiciel prépondérant, basé sur l'utilisation de vues. L'utilisation de plusieurs vues permet de traiter séparément les préoccupations des différents intervenants de l'architecture : utilisateurs finaux, chefs de projet, etc. et de capturer séparément les exigences fonctionnelles et non-fonctionnelles.
Cette approche propose quatre vues s'articulant autour d'une cinquième appelée scénario (use-case).

1. La vue *logique*, qui est le modèle objet de la représentation (lorsqu'une méthode objet est utilisée) ;
2. La vue *processus* qui capture les aspects concurrence et synchronisation du système ;
3. La vue *physique* qui décrit l'application du logiciel sur le matériel, reflétant son caractère distribué ;
4. La vue *développement* qui décrit l'organisation statique du logiciel dans son environnement de développement.

Le modèle 4+1 vues décrit ici a été utilisé avec succès sur plusieurs grands projets. Il permet aux différents intervenants de trouver ce qu'ils veulent savoir sur l'architecture du logiciel. Cependant, les vues proposées n'ont pas été clairement formalisées (la méthode propose un découpage en vues et les informations sensées s'y trouver, mais pas de formalisme pour les exprimer) et les vues ne sont pas orientées "métier", chaque intervenant utilisant plusieurs vues (un ingénieur système utilisera la vue physique puis la vue processus).

Modélisation par aspects

La modélisation par aspects[2] [CM04, CMR03, Bez02, MGFA02, AT98] cherche à répondre à la problématique de la complexité des développements logiciel modernes en séparant les spécifications

[2] AOM : Aspect Oriented Modelling

fonctionnelles de celles considérées comme non-fonctionnelles (ou extra-fonctionnelles). Les spécifications fonctionnelles sont exprimées au sein d'un modèle appelé modèle primaire (primary model). Chaque propriété non fonctionnelle est modélisée par un modèle particulier nommé aspect. Un aspect indique comment intégrer la solution qu'il modélise dans le modèle primaire. Le modèle complet est obtenu en "tissant" les aspects sur le modèle primaire. Cette approche pose plusieurs problèmes. Dans le cadre d'un développement par sous-traitants, les spécifications fonctionnelles sont distribuées entre les différents intervenants, ce qui conduit à manipuler plusieurs modèles primaires, les aspects demeurant transversaux. La question de savoir comment effectuer le tissage des aspects dans ce cas n'a pas, à ce jour, de réponse technique satisfaisante. La modélisation par aspects prévoyant des raffinements successifs des modèles primaires et des aspects, conduit donc à la manipulation d'un modèle primaire complet du système, diminuant ainsi les avantages dûs à la distribution des spécifications fonctionnelles.

Approches par composants

Une autre approche répondant à la problématique du développement de grands systèmes logiciels est l'approche dite par composants. La philosophie générale de cette approche est de développer indépendamment des parties du système, puis de les lier pour former le système. Chaque composant possède une interface publique et une partie implantation privée. Le système complet s'obtient grâce à une "glue" logiciel inter-connectant les interfaces. L'objectif principal de cette approche est la réutilisation des composants. Il s'agit de développer des portions de logiciel indépendamment et de pouvoir les utiliser pour construire différents systèmes à différents moments. Il faut aussi citer CORBA (Common Object Request Broker Architecture) qui permet de développer des objets distribués indépendamment du langage et de la plate-forme cible, puis de les assembler.

PRISME

Le problème de la conception multi-vues a fait l'objet d'un travail de recherche à l'ONERA de 1997 à 2000 dont les résultats ont motivé la mise en œuvre de la présente thèse. Le but de ce projet, dénommé "PRISME Avionique Nouvelle", était d'étudier et de définir des modèles pour supporter la description de systèmes avioniques dans trois domaines *métier* différents : fonctionnel, sûreté de fonctionnement et performances temps réel [BBD$^+$].
L'hypothèse centrale du projet était l'existence d'une description pivot, appelée squelette, qui sert de référence aux descriptions *métiers* considérées (les vues).
Selon cette approche, un système est complètement décrit par trois modèles comportementaux, un par point de vue (fonctionnel, performances temps réel et sûreté de fonctionnement), notés respectivement MFct, MPTR et MSdF. Pour garantir que ces trois modèles comportementaux sont structurellement cohérents, i.e. adressent bien les mêmes objets, ils s'appuient sur une description architecturale unique, appelée squelette. Le squelette constitue ainsi le socle de la description, sorte de représentation *fil de fer* du système sur lequel viennent se greffer les descriptions comportementales par point de vue. Selon cette philosophie, un tel squelette est un modèle statique de l'ensemble des objets du système et de leurs relations. Il exprime notamment les topologies des architectures fonctionnelles et matérielles du système, ainsi que les relations d'allocation de la première sur la seconde. En revanche, une telle description est, à ce niveau, vide de sémantique. Une vue du système est ensuite construite, selon cette même philosophie, comme un enrichissement comportemental et sémantique de ce squelette. Cette organisation "multi-vues" peut être assimilée à un prisme dont les facettes sont les vues MFct, MPTR et MSdF, et le socle le squelette. Cette

approche ne répond cependant qu'à la première question : la cohérence structurelle des vues. Nous avons donc cherché à répondre au problème de la cohérence sémantique, laissée de côté par PRISME. Cette question constitue le point de départ de nos travaux, et bien que nous soyions largement détachés de ce contexte, une partie de la terminologie que nous employons est héritée de ce projet.

1.2.2 De l'ingénierie logiciel à l'ingénierie système

Au niveau de l'ingénierie système, des besoins identiques se sont manifestés pour la maîtrise de la complexité, l'analyse *a priori* et la qualification. Des démarches de modélisation se sont également développées selon les différents points de vue du système. De part la nature même de l'ingénierie système, différents modèles ont été créés pour certains domaines métiers précis comme la sûreté de fonctionnement, l'analyse de performances ou le dimensionnement, assurant ainsi des possibilités d'analyse. Le nombre de domaines modélisés doit augmenter au niveau système pour améliorer la maîtrise et surtout assurer une cohérence entre domaines. En effet, le contrôle de cohérence entre domaines n'est possible que si tous les domaines possèdent des modèles plus ou moins abstraits mais garantissant des possibilités d'analyse. Un besoin d'unification des définitions apparaît donc, comme en logiciel, pour faciliter le contrôle de cohérence. Pour cela, une ingénierie des modèles doit se développer pour la gestion des modèles et également pour les intégrer dans un processus de réalisation. Ce processus basé sur les modèles doit également garantir une continuité dans les modèles tout au long du processus pour obtenir une cohérence entre différents niveaux d'abstraction conduisant à la réalisation du système.

Modélisation système dirigée par les modèles à THALES : MDSysE

Thales, via un programme pilote, travaille sur l'IS dirigée par les modèles depuis 2002. Ces travaux ont conduit à l'élaboration d'une méthodologie « outillée » dédiée : MDSysE. Cette méthodologie organise les différents processus d'ingénierie autour de la construction et de l'exploitation d'un certain nombre de modèles interdépendants qui représentent différents aspects du système à différents niveaux d'abstraction. Les activités d'ingénierie impliquent la construction, l'analyse, et la transformation de modèles du système et la génération automatique de différents éléments du processus à partir de ces modèles. Cette méthodologie « outillée » comprend : La définition précise des produits de modélisation pour la définition du système. L'identification et la fourniture de règles et de techniques de vérification syntaxique. Le support à l'automatisation par la définition et la fourniture de règles de transformation entre modèles et de règles de génération à partir des modèles. L'outillage supportant la méthodologie inclut des assistants et autres éditeurs (« wizards ») étendant le modeleur pour permettre un travail au niveau conceptuel plutôt qu'au niveau langage de modélisation. Certains assistants aident également à la vérification, à la transformation, à la configuration ou à la génération automatique. MDSysE a été construit comme une implémentation générique de l'IS dirigée par les modèles pour Thales, basée sur différentes études conjointes avec les unités dans différents domaines.

La Modélisation multi-vues

La modélisation des systèmes intervient de plus en plus pour faciliter la prise de décision lors de la conception système pour effectuer des évaluations de performances et de SdF. Il faut étendre la

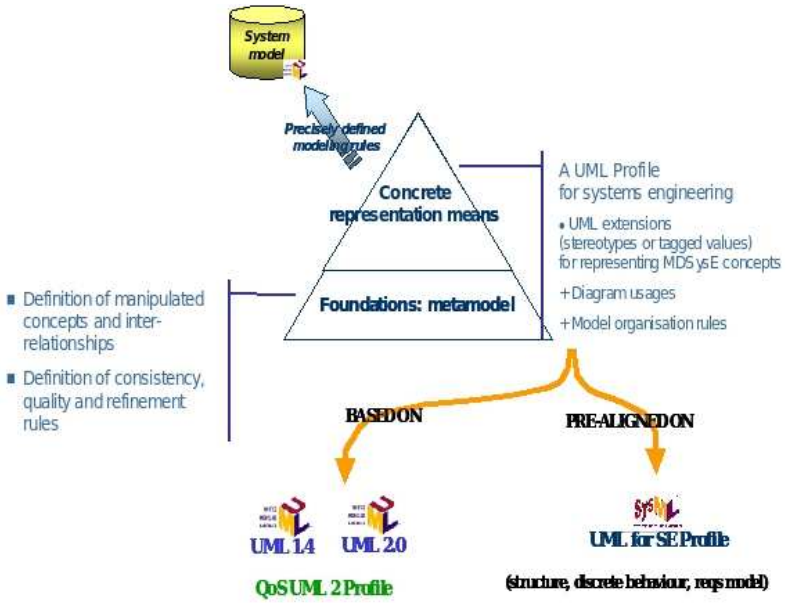


FIG. 1.1 – Approche pour la chaîne outillée MDSysE

testabilité des systèmes à la testabilité des modèles nécessaires à l'expression des systèmes.

Modèles pour les Systèmes : Les modèles pour les systèmes, la plupart complexes, impliquent des modèles variés, complexes et utilisent différents formalismes et sémantiques. De plus, suivant les composants ou les aspects que l'on veut tester sur le système, il faut utiliser différents formalismes ou certaines parties du système sans nécessairement intégrer la totalité des connaissances sur le système.

Modélisation multi-vues : L'idée consiste à formaliser un modèle de concepts associé au systèmes complexe s'appuyant sur une modélisation pivot. Le projet PRISME (ONERA) propose un modèle multi-vues pour des systèmes dirigé par des concepts. La mise en ?uvre de cette méthode est réalisée dans un second projet ONERA appelé CARLIT (Coeur Avionique ReconfigurabLe InTégré). Il s'agit d'étudier de nouvelles techniques d'aide à la conception de systèmes avioniques militaires en architecture IMA. La méthodologie consiste à aborder cette question en adoptant un point de vue systémier et en particulier en étudiant une démarche de conception mêlant plusieurs facettes métiers, dont une facette physique.

Meta-Modélisation : On ne peut formaliser ce modèle multi-vues intégrant toutes les composantes du système de manière complète. En revanche, la méta-modélisation issue du monde UML est un bon candidat : possibilité d'exprimer de larges modèles de données pouvant représenter des concepts suivant différents niveaux d'abstraction comme le sous-tend l'approche objet qui reste le cœur d'UML. Cette définition ainsi obtenue reste semi-formelle mais elle permet l'adaptation ou la transformation vers différents formalismes qui eux pourront apporter des réponses sur la testabilité du système.

Approche MDA : L'approche MDA classifie les modèles en 3 familles : modèle indépendant de la plate-forme d'exécution, modèle spécifique à une plate-forme ou modèle d'une plate-forme. Cela permet de concevoir des modèles exprimant les fonctionnalités système indépendamment de toute architecture support. Ensuite, les architectures support sont modélisées et les modèles systèmes peuvent être portés sur les architectures en effectuant des fusions de modèles.

Les techniques de fusion de modèles font aujourd'hui l'objet de travaux de recherche de la part de l'ENSIETA. Si le principe de fusion de modèles est bien connu, la mise en ?uvre de règles de fusion automatique reste un problème ouvert dans le cas général. Il doit donc être envisagé d'étudier des techniques d'aide à la fusion, et plus généralement d'aide à la transformation de modèles avec la prise en compte de contextes bien définis, dédiés, dans lesquels s'opèrent les transformations. Ce cadre générique MDA constitue actuellement la référence pour l'ingénierie des modèles (IDM) qui voit le jour après la définition du langage UML. On peut noter d'ailleurs que la conférence internationale UML, qui se déroule alternativement en Europe et aux Etats Unis s'ouvre actuellement au domaine de l'ingénierie des modèles.

Vers une standardisation des languages de modelisation des systemes : SysML (Systems Modeling Language)

SysML est un langage de modélisation dédié à l'ingénierie des systèmes. Il est particulièrement adapté pour décrire les exigences, la structure du système, le comportement fonctionnel et l'allocation durant les phases de spécification et de conception du système. La description du système d'articule autour des vues statiques, dynamiques et transverses. SysML 1.0 a été adopté par l'OMG la 6 juillet 2006. La description statique du système est organisée autour des diagrammes de classes, des diagrammes d'assemblages et des diagrammes paramétriques. Ces derniers permettent d'exprimer des contraintes mathématiques sur des objets du système. La description dynamique du

système spécifie l'évolution dans le temps des éléments de modèles décrits dans la vue statique et reprend principalement les diagrammes dynamiques d'UML 2.0. Notons que l'on peut spécifier des probabilités sur des arcs d'activités ou à des paramètres de sortie. On peut aussi spécifier des flots de données continus ou discrets. Les descriptions transverses permettent d'une part l'expression des besoins au travers des diagrammes d'exigences. D'autre part, elles permettent de spécifier l'allocation entre les vues statiques et dynamiques et entre différents niveaux d'abstraction. Cette notion générale peut être raffinée et l'exemple le plus courant est celui de l'allocation d'éléments logiciels sur des éléments matériels. Un diagramme d'exigences a été crée pour l'expression des besoins.

Chapitre 2

Contexte et approche

"Rien ne sert de penser, faut réfléchir avant."
Pierre Dac

Cette thèse porte essentiellement sur les aspects méthodologiques du développement des systèmes à logiciel prépondérant. Il ne nous a pas paru pertinent de traiter de méthodologie dans un contexte aussi général. Nous avons donc décidé de restreindre notre domaine d'étude aux systèmes embarqués qui, de part les nombreuses contraintes qui leur sont attachées et les enjeux majeurs qu'ils représentent, offrent un cadre idéal à nos travaux. Le présent chapitre décrit ce cadre.

2.1 Contexte : les systèmes embarqués

2.1.1 Caractéristiques générales

Les systèmes informatiques embarqués connaissent depuis plusieurs années un essor considérable dans tous les domaines : automobile, aéronautique, espace, contrôle industriel, télécommunications.... L'appel à l'informatique dans l'ensemble de ces domaines a tendance à se généraliser avec les progrès constants des équipements numériques. Pour autant, ces systèmes informatiques embarqués ne peuvent pas être assimilés à des systèmes informatiques généraux. D'une part, ils sont souvent soumis à des contraintes strictes imposées par leur environnement. Et d'autre part, en raison même de ces contraintes, ils répondent fréquemment à des caractéristiques particulières qui en rendent l'analyse à la fois plus spécifique et plus pointue.
La question est alors : peut-on caractériser les systèmes embarqués, notamment par les contraintes auxquelles ils doivent répondre ? De plus, peut-on identifier des classes particulières de systèmes embarqués selon leur structure et leur sémantique sous-jacente ?
Un système embarqué est avant tout un système réactif dont la fonction est de contrôler l'évolution de son environnement. A titre d'exemple, un pilote automatique est un système embarqué chargé de

guider un avion conformément à un plan de vol défini par l'équipage. De même, un système de commande de vol du type X-by-wire a pour mission de contrôler les mouvements de l'avion en agissant sur un ensemble de gouvernes en fonction de commandes de pilotage émises par l'équipage. On pourrait également citer un système de régulation d'une réaction nucléaire.... Tous ces systèmes sont soumis aux contraintes de leur environnement. Ces contraintes peuvent être de nature physique, telles que le poids, l'espace occupé, les vibrations, les variations de température, les rayonnements électromagnétiques.... Elles peuvent être aussi (et surtout en ce qui nous concerne dans cette thèse) de nature comportementale :

– Criticité : si la fonction du système dans son environnement est jugée critique, c'est-à-dire si ces défaillances sont catastrophiques, alors le système de contrôle informatique est également critique ; c'est le cas par exemple d'un système de régulation d'une réaction nucléaire ou d'un système de commande de vol du type X-by-wire. Notons que pour les systèmes dont la durée de vie (ou de mission) est courte, on peut parfois privilégier l'exigence de disponibilité opérationnelle ou de réalisation de la mission, à savoir la capacité du système à pouvoir répondre à tout moment à une sollicitation. C'est par exemple le cas des systèmes militaires ou des lanceurs spatiaux.
– Temps-réel : si leur environnement est dynamique, voire instable, alors le système de contrôle informatique est soumis à des exigences de temps de réponse, c'est-à-dire des exigences portant sur les délais introduits par le système informatique entre des stimuli et des actions correspondantes. Ces délais peuvent être provoqués par les temps d'exécution des tâches et les temps de transmission des messages (incluant les temps d'accès aux ressources CPU et réseaux). Le non-respect de ces exigences peut rendre la boucle de contrôle instable et provoquer des défaillances catastrophiques.
– Réactivité : un système embarqué doit par définition surveiller et contrôler un environnement. Si cet environnement présente lui-même un comportement changeant ou de type événementiel (ont dit aussi "sautillant" par opposition à des environnements plus réguliers qualifiés de "ronronnants"), il est nécessaire que le système informatique soit capable de s'adapter à ces *bouffées* d'événements et mettant en œuvre des calculs et des communications également événementiels.
– Robustesse : un système embarqué est autonome vis à vis de l'humain : il n'est généralement pas prévu qu'un opérateur intervienne sur le logiciel si celui-ci est perdu. Par conséquent, il est vital que le système soit capable de réagir de façon satisfaisante (et déterministe) même si son environnement se comporte de manière imprévue, improbable ou fantaisiste. Le concepteur du système ne peut en aucun cas faire d'hypothèse sur le comportement de l'environnement.

Ces contraintes comportementales ont un impact fort sur les architectures fonctionnelles et physiques des systèmes. De façon générale, et dans un souci de répondre à la fois à des contraintes de criticité, disponibilité, performance ou réactivité, les systèmes embarqués seront généralement composés de plusieurs entités répliquées. En revanche, selon que l'on privilégie le point de vue sécurité, disponibilité, ou réactivité, ou encore selon le domaine applicatif, ces architectures pourront :

– être vues comme une collection de sous-systèmes centralisés communicants, ou au contraire conçues comme un système réparti
– reposer sur une sémantique synchrone, ou au contraire asynchrone
– et être organisées selon une architecture statique ou au contraire dynamique (i.e., offrant des capacités de reconfiguration).

Les avioniques civiles par exemple, sont soumises principalement à des contraintes de criticité et de temps. Elles reposeront pour la plupart sur des architectures distribuées statiques, asynchrones et dans certains cas sur-dimensionnées. A l'inverse les avioniques militaires privilégient la réactivité et la disponibilité. Elles sont construites sur des architectures synchrones, offrant des capacités de

reconfiguration (sans pour autant être totalement dynamiques).

2.1.2 Architectures des systèmes embarqués

On peut distinguer trois types d'architectures embarquées : les architectures *centralisées* que l'on trouve plus fréquemment dans des engins tels que les missiles ou les lanceurs, les architectures *fédérées* plus communes aux aéronefs civils et que l'on peut interpréter comme étant des architectures localement centralisées et globalement asynchrones, et enfin les architectures *réparties* plutôt mises en œuvre dans les aéronefs militaires.

Les architectures centralisées

A l'époque des premières machines informatiques embarquées, le nombre et la complexité des fonctions mises en œuvre étaient relativement restreints. Un mini-calculateur suffisait à centraliser l'acquisition, le traitement et la mémorisation des informations. Les avantages de ce type d'architecture sont nombreux :

- Le calculateur est généralement situé dans un endroit accessible, facilitant ainsi l'intégration puis la maintenance du système embarqué.
- Le calculateur est généralement situé dans un endroit protégé, à l'abri des perturbations (interférences, risques d'explosion ...) pouvant être provoquées par les autres composantes de l'engin, simplifiant ainsi l'analyse des pannes de l'avionique et, par suite, sa certification.
- Le système étant centralisé, son architecture matérielle en est simplifiée.

En revanche, les inconvénients inhérents à une architecture centralisée sont également importants, et souvent rédhibitoires :

- Un système centralisé est plus vulnérable aux défaillances.
- Le système de traitement étant localisé en un endroit unique, il est par conséquent loin des capteurs ou des actionneurs (voire des deux). Une architecture centralisée nécessite donc un très grand nombre de bus analogiques de grande longueur.

Ces inconvénients sont rédhibitoires pour les systèmes de grande taille soumis à des contraintes de sûreté sévères, tels que les avions de transports civils ou les avions d'armes. L'informatique de ces appareils ne sera donc pas centralisée mais globalement fédérée.
En revanche, ces mêmes inconvénients ne sont pas prohibitifs pour des engins tels que les missiles ou les lanceurs qui ont une durée de vie très courte, et qui par conséquent doivent afficher des taux de pannes par heures moins importants que les aéronefs civils. Par exemple, l'avionique centralisée d'un missile est constituée d'un seul calculateur relié à un ensemble de capteurs (radar et capteur de mouvements), et contrôlant un ensemble d'actionneurs (le moteur, les ailerons, la charge ...). Les communications entre le calculateur et les capteurs, les actionneurs, ou éventuellement avec l'engin lanceur (un avion d'armes par exemple) sont assurées par un grand nombre de bus analogiques (puisque directement alimentés ou consommés par les capteurs et les actionneurs).
De même, bien que l'ensemble de l'avionique d'un aéronef civil soit de type fédéral, chacun de ses sous-systèmes est généralement conçu selon une approche centralisée. C'est le cas par exemple du système qui élabore les paramètres d'état de l'avion (altitude, vitesse...). On peut également citer le sous-système de contrôle des organes physiques du train d'atterrissage d'un avion d'armes. Ce système est composé de deux calculateurs redondants, fonctionnant chacun avec ses capteurs et ses actionneurs.
En revanche, vue à niveau global, la structure de ces systèmes répond à un principe différent : celui des architectures fédérées.

Les architectures réparties

Un autre type d'architecture a été développé, depuis plusieurs années dans les systèmes militaires ou automobiles : les architectures réparties. Dans de telles architectures, les fonctions ne sont plus réalisées par un calculateur, mais par un ensemble d'équipements informatiques communicants. On parle alors souvent de "chaîne fonctionnelle" (bien qu'une telle architecture ne constitue pas une chaîne à proprement parler mais un réseau de processus).
Une telle répartition est rendue nécessaire souvent par la géographie de l'application, mais aussi par l'utilisation de capteurs et actionneurs offrant localement des capacités informatiques (on parle de capteurs et actionneurs "intelligents"). Il est alors possible de déporter directement sur ces équipements des fonctions d'élaboration de données de haut niveau pour les capteurs, ou des fonctions de commande locale (asservissement par exemple) pour les actionneurs.
De même, on observe aujourd'hui une tendance au remplacement de "gros" calculateurs centraux, qui présentent de plus en plus de difficultés de certification en raison de leur complexité interne, par des "grilles" de petits calculateurs plus simples.
Ces raisons conduisent à l'émergence d'une classe d'architecture répartie. Se pose alors la question de la sémantique sous-jacente de cette classe d'architecture. Contrairement aux architectures fédérées qui reposent sur un couplage faible, donc asynchrone, entre des fonctions indépendantes, la répartition d'une fonction sur des équipements distants nécessite une synchronisation plus forte entre les différents traitements. Cette synchronisation peut être apportée soit par les communications elles-mêmes au moyen de mécanismes du type "rendez-vous", ou par le partage d'une base temporelle commune. On observe que c'est souvent la seconde solution qui est retenue dans les applications critiques, les "rendez-vous" pouvant introduire des points de blocages dangereux en cas de panne. On peut alors parler de systèmes globalement "faiblement synchrones". On utilise l'adverbe "faiblement" par opposition aux langages synchrones tels que Lustre, Esterel ou Signal qui non seulement supposent l'existence d'une base de temps globale mais reposent également sur l'hypothèse que les temps d'exécution et de communication sont nuls dans ce référentiel temporel commun. Par l'expression "faiblement synchrone", nous traduisons le fait que ces architectures reposent également sur l'hypothèse d'un référentiel temporal global, mais que les temps de communication dans ce référentiel sont supposés non nuls.
En pratique ces architectures sont construites autour de bus à diffusion synchronisant (virtuels ou réels). Ces bus offrent souvent des fonctionnements à trames dans lesquelles s'insère le trafic dans un ordre essentiellement déterministe (au moins pour le trafic critique).
L'intérêt de ce type d'architecture est de permettre une gestion quasi synchrone de l'ensemble du système avionique, donc plus déterministe, et par suite d'en réduire la combinatoire.

Les architectures fédérées

Le troisième type d'architecture rencontré et que nous avons adopté dans le cadre de cette thèse est dit "fédéré". Une architecture fédérée est généralement constituée d'un ensemble de sous-systèmes réalisant chacun une fonction embarquée (commandes de vol, pilote automatique ...), pilotant des actionneurs, ou élaborant des informations numériques à partir d'informations produites par des capteurs. Ces sous-systèmes sont localisés en divers endroits de l'engin embarquant (un aéronef par exemple), généralement à proximité des capteurs ou des actionneurs, et peuvent être considérés chacun pris séparément comme des sous-systèmes centralisés. En ce sens, une architecture fédérée peut être assimilée à un assemblage de sous-architectures centralisées.
D'un point de vue sémantique, les architectures fédérées reposent pour la plupart sur un modèle

globalement asynchrone. Les communications entre sous-systèmes sont assurées principalement par des flots de données asynchrones, la plupart du temps périodiques, et sont supportées par des canaux à diffusion et des mécanismes de files d'attentes en entrée de chaque sous-système.
C'est le principe général des architectures des avions de transports civils. L'avionique (c'est-à-dire l'ensemble du système informatique embarqué) est décomposée en plusieurs sous-systèmes, chacun ayant une fonction spécifique. On trouve par exemple, le sous-système "centrale anémométrique" qui élabore les paramètres de vol (position, vitesses, accélérations), un sous-système "pilote automatique", un sous-système de surveillance du domaine de vol, un autre de gestion des alarmes, d'autres pour le contrôle des organes physiques tels que les gouvernes, les moteurs, les réservoirs de fuel .... Chaque sous-système étant répliqué pour des raisons évidentes de sûreté de fonctionnement. En première approximation, ces sous-systèmes forment un graphe de diffusion acyclique dans lequel les nœuds sont les sous-systèmes et les arcs des canaux asynchrones transportant les flots de données (les paramètres de vol, les consignes de guidage, de pilotage, de régulation ...).
L'intérêt de ce type d'architecture est qu'elle permet naturellement la conception et le développement séparés de chaque sous-système (après une première spécification globale), puis l'évolution de l'architecture par ajouts, transformations ou suppressions de sous-systèmes. Le développement en sera donc plus modulaire, et incrémental. Cette indépendance conceptuelle entre sous-systèmes est intrinsèquement liée à l'hypothèse d'asynchronisme. En revanche, ce même asynchronisme est également synonyme d'explosion combinatoire par le simple entrelacement des comportements de chaque sous-système. Un tel entrelacement n'est pas sans poser quelques difficultés lors de l'analyse du système global.
Des différentes architectures que nous avons décrites, nous retiendrons comme sujet d'étude les architectures dites fédérées. Celles-ci offrent l'avantage d'illustrer des problématiques caractéristiques des deux autres tout en les généralisant, puisque l'on peut voir une architecture fédérée comme un ensemble de sous-systèmes centralisés organisés de façon répartie.

2.1.3 Les exigences de développement d'un système embarqué

En plus des contraintes environnementales énoncées précédemment, les systèmes embarqués industriels, principalement avioniques et automobiles, doivent également répondre à des exigences liées à leur processus de développement, et notamment des exigences ayant pour but de réduire leur temps de développement, de modification et de validation. On peut regrouper ces exigences en quatre catégories :

Besoins de modèles à la fois locaux et globaux.

Nous venons de le voir, un système embarqué est souvent constitué de plusieurs entités communicantes (ne serait-ce que pour des besoins de redondance). Une spécification d'un système embarqué doit donc mettre en œuvre ces deux niveaux de description : un niveau local correspondant par exemple à une tâche ou un ensemble de tâches d'un sous-système (le pilote automatique par exemple), et un niveau global décrivant l'interaction entre ces composants (par exemple au niveau d'un avion complet).

Paradoxalement, le niveau global est en pratique assez peu décrit, ou du moins ne fait l'objet bien souvent que de descriptions textuelles ou graphiques informelles. Ce constat peut s'expliquer par le fait que ce niveau n'est pas directement opérationnel. Le code généré automatiquement l'est à partir des spécifications des composants locaux, alors que le niveau global ne conduit généralement pas à un codage ou une réalisation automatique. Seuls les points de vue physique et intégration (notamment pour la description de l'utilisation des

moyens de communication lorsque ceux-ci sont partagés) peuvent faire l'objet d'une description globale et détaillée. Pour autant, dès lors que l'on souhaite valider le fonctionnement du système (en considérant donc à la fois son comportement fonctionnel et ses performances de temps et ses occurrences de pannes), l'utilisation d'un modèle global, de simulation ou de vérification, est incontournable. Quelles doivent être la structure et la sémantique de ce modèle ? C'est une des questions abordée dans le chapitre suivant.

Déterminisme :

Une des exigences minimales imposée pour la certification d'un système embarqué critique est son déterminisme, liée à la contrainte de robustesse évoquée précédemment. Il doit afficher un comportement prévisible, et toujours identique, lorsqu'il est soumis plusieurs fois à un même scénario. Or, on sait qu'un tel niveau de déterminisme est très difficile à atteindre. Plusieurs sources de non-déterminisme peuvent en effet affecter un système : d'une part les imprécisions liées à ses organes physiques (capteurs et actionneurs), et d'autre part les retards et éventuels décalages dus à l'asynchronisme du système (dans le cas d'une architecture globalement asynchrone bien entendu). En conséquence, le non déterminisme est souvent inhérent aux systèmes informatiques pour peu qu'ils soient un peu étendus et qu'ils contiennent des organes imprécis. L'exigence est donc, plutôt que de vouloir proscrire globalement un tel non-déterminisme, d'une part de l'interdire au niveau de chaque unité d'exécution élémentaire, puis d'autre part de montrer que le système dans sa globalité est stable ou tolérant à son indéterminisme intrinsèque.

Vérification :

On vient de voir que l'exigence de validation est essentielle dès lors que l'on considère un système critique. Pour autant, cette exigence n'est pas toujours aisée. En pratique, elle requiert des moyens de simulation, de test sur des maquettes ou sur des prototypes (par exemple des essais en vol). Il est rare en revanche que l'on puisse utiliser des techniques de vérification automatique sur un système global principalement à cause du problème bien connu de l'explosion combinatoire. C'est là une des difficultés persistantes des méthodes formelles pour la conception de systèmes embarqués.

Néanmoins, cette vérification peut être plus ou moins facilitée selon la nature des modèles développés, selon qu'ils soient synchrones ou asynchrones, temporisés ou discrets, par exemple. En réalité, modéliser, c'est déjà préparer la vérification. Et inversement, exiger de pouvoir vérifier c'est d'abord exiger des modèles se prêtant à cette activité. Cette question est la question principale abordée dans cette thèse, appliquée aux trois types d'architecture recensés précédemment.

Le problème de la modification par "delta" :

La dernière exigence que l'on mentionnera est liée au processus industriel. Certains systèmes sont parfois difficiles à spécifier précisément du premier coup, de part la complexité des fonctions mises en œuvre, mais aussi du nombre important de données d'environnement à traiter. C'est par exemple le cas d'un pilote automatique d'aéronef. De telles difficultés se traduisent concrètement par un nombre important de modifications et de réglages de la spécification du début du développement jusqu'à la phase de certification. D'autres systèmes peuvent être appelés à être modifiés en phase opérationnelle. C'est par exemple le cas des systèmes militaires auxquels on veut pouvoir demander de s'adapter en quelques jours sur le champ de bataille à une nouvelle situation tactique. Ces modifications peuvent aller de l'adaptation des moyens de communication d'un aéronef que l'on intègre dans un dispositif allié, jusqu'à l'embarquement d'un nouveau capteur ou d'une nouvelle arme. Dans ce cas, il

doit être possible de modifier le système embarqué (le plus souvent sa partie logicielle) sans que cela nécessite une longue phase de recertification.

Les modèles, support du développement et de la validation du système embarqué, doivent donc être suffisamment modulaires pour permettre de telles modifications locales.

Une fois présenté le contexte de nos travaux, nous pouvons maintenant fournir une description générale de notre approche servant de guide de lecture du reste du document.

2.2 Présentation générale de notre approche

2.2.1 Idée générale

L'objectif de cette thèse est de fournir un cadre méthodologique ainsi qu'un ensemble de techniques permettant d'aider à la maîtrise d'œuvre de systèmes embarqués mettant en jeu de nombreux spécialistes métier différents, chacun réalisant sa partie du système. Pour ce faire il est impératif que la méthodologie que nous proposons aie les propriétés suivantes :

– Tout d'abord il est souhaitable que chaque intervenant puisse travailler sur ses propres modèles et que sa connaissance des autres parties du système réalisées par les autres intervenants soit limitée aux strictes informations dont il a besoin pour réaliser sa partie. Il est aussi important de ne pas forcer les intervenants à opter pour un formalisme de modélisation commun : chaque domaine métier possède des outils de modélisation adaptés à ses problématiques (DSL[1]) et fournissant des fonctionnalités qui ne peuvent être incluses dans un langage de modélisation général.
– Ensuite, le maître d'œuvre doit disposer d'un modèle adapté à ses préoccupations. La vision qu'il porte sur le système à tout moment du cycle de développement lui est propre, elle doit être synthétique et refléter l'intégralité des exigences système et lui permettre de savoir quels sous-traitants sont impliqués dans la réalisation de chacune d'elles. N'étant pas spécialiste d'un domaine le MOE (Maître d'œuvre)[2] n'a pas besoin de connaître les détails de l'implantation des différentes parties de son système. Notre approche devra lui cacher le raffinement des modèles des intervenants tout en lui fournissant les informations qui lui sont nécessaires. Enfin notre approche doit fournir le moyen de vérifier la cohérence des différentes parties du système et permettre de gérer les préoccupations transverses[3] en calculant des informations générales à partir de données fournies par les intervenants concernant leur partie.

Pour réaliser ces objectifs, nous proposons une méthodologie reposant sur un ensemble de modèles organisés de la façon suivante :

Chaque intervenant (appelé facette) fournit un modèle abstrait -appelé modèle public de la facette- de la partie qu'il réalise. Le formalisme utilisé pour exprimer ce modèle ne contraint pas le formalisme utilisé par le sous-traitant dans sa réalisation de la partie du système qui lui incombe.

Nous caractérisons le modèle abstrait dédié au maître d'œuvre appelé pivot.

Nous proposons aussi un ensemble de modèles d'intégration, appelés layers, chargés de s'assurer que la cohérence des différentes parties du système est maintenue dès qu'un intervenant effectue un choix de conception.

[1] Domain Specific Languages
[2] Maître d'œuvre
[3] cross-cutting concerns

Chacun de ces modèles sera décrit dans un langage lui-même caractérisé par une grammaire appelée méta-modèle -du fait de son niveau d'abstraction-. Le sujet de cette thèse n'étant pas centré autour d'UML, nous adoptons la facilité d'écriture suivante : Dans la suite de ce document, plutôt que de définir des méta-modèles instances du MOF, nous utiliserons pour décrire ces grammaires une expression graphique utilisant la syntaxe des *modèles* UML. Ce choix permet une expression plus compacte de nos grammaires et, nous semble-t-il, plus intuitivement lisible par le profane.
Enfin nous fournissons un processus de développement intégrant notre méthodologie.
La suite de ce chapitre donne un aperçu général des différents éléments de notre approche. La figure 2.1 schématise l'ensemble des relations existant entre ces différents concepts.

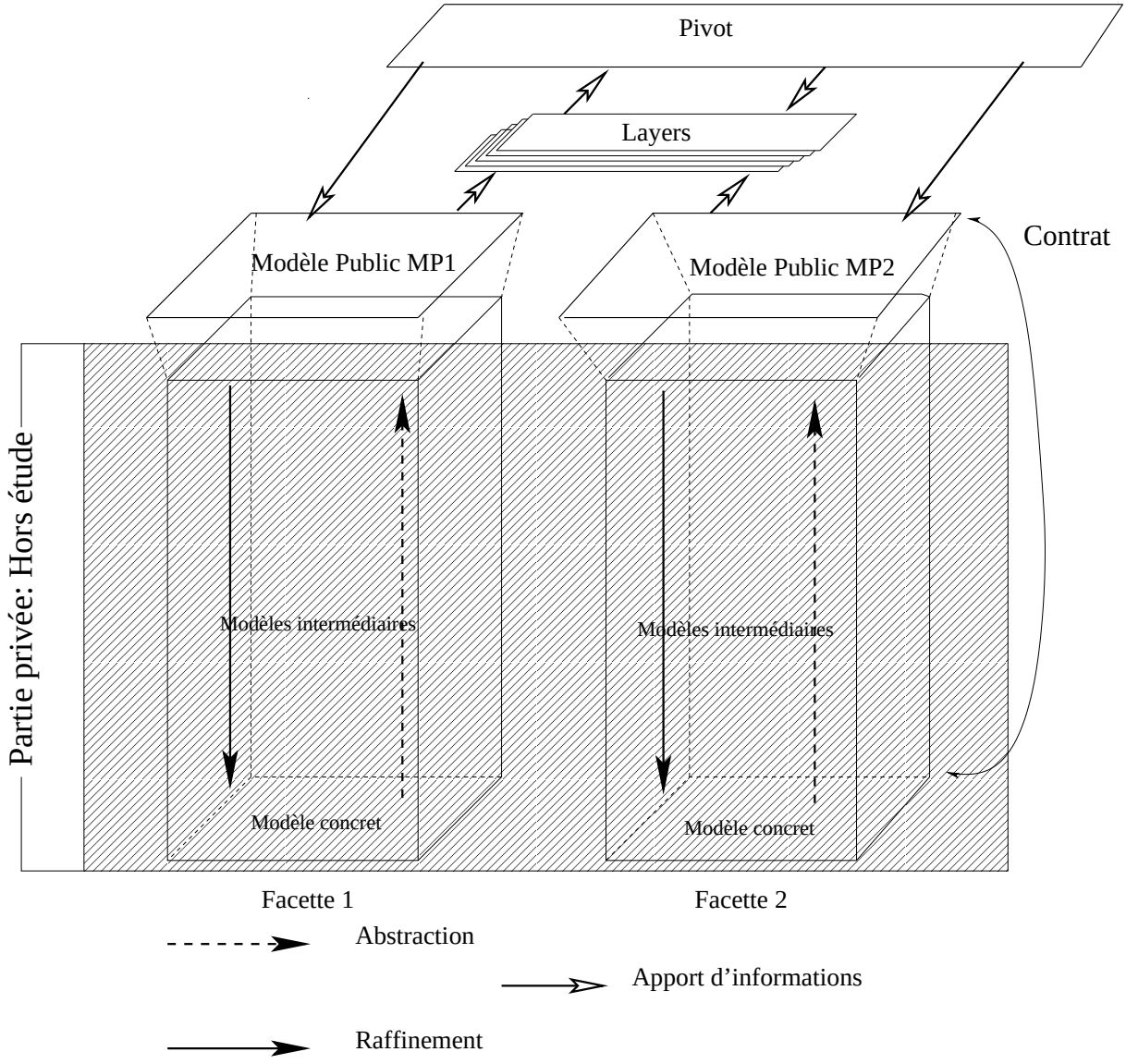


FIG. 2.1 – Schéma général

2.2.2 Les facettes

Le principal axe de séparation des préoccupations est la séparation par domaine métier. Nous dénommerons donc par le terme générique de facette à la fois les exigences liées à un métier, l'équipe physique spécialisée dans ce métier et l'ensemble de modèles du sous-système qu'elle conçoit. Par exemple, dans le cas du développement d'un avion on peut citer les facettes électronique, logiciel embarqué, hydraulique, automatique, aérodynamique, etc. Une facette est constituée d'une partie privée et d'une partie publique. La partie privée d'une facette, connue d'elle-seule, est composée de l'ensemble des modèles nécessaires à la réalisation de sa partie du système. Le choix des formalismes employés dans cette partie est laissé à l'entière discrétion de la facette. La partie publique appelée Modèle Public (MP) a plusieurs fonctions : elle sert de contrat entre l'intervenant et le MOE en spécifiant le modèle solution abstrait regroupant à la fois les informations architecturales et "comportementales" sur lesquelles portent les exigences de la facette. Il est en outre suffisamment précis pour permettre le calcul des informations transverses nécessaires au maître d'œuvre pour vérifier les exigences système et pour permettre l'intégration finale du sous-système développé par la facette. Dans cette thèse nous nous intéresserons exclusivement au Modèle Public. La notion de facette est présentée en détails au chapitre 4 page 45.

2.2.3 Le pivot

Étant données la taille et la complexité des grands systèmes actuels (avioniques, navals ...) le maître d'œuvre ne peut ni ne doit connaître en détail l'intégralité des modèles du système dont il a la charge. De même, les équipes métier impliquées dans le développement (ou le maintient en condition opérationnelle), si elles doivent maîtriser à 100% leur domaine, n'ont besoin que d'une quantité réduite d'informations sur le reste du système. Pour répondre à cette double problématique, nous définissons un modèle abstrait appelé pivot qui regroupe des informations de haut niveau permettant d'une part au maître d'œuvre d'avoir une vision synthétique du système et d'autre part de servir d'interface entre les Modèles Publics des différentes facettes. Le pivot est un modèle abstrait du système dans sa globalité : les entités du pivot sont conçues comme des blocs logiques sur lesquels portent une ou plusieurs exigences globales. En plus d'assurer le rôle de modèle du MOE, le pivot apporte une cohérence syntaxique en exprimant les informations fournies par chacun dans un langage maîtrisé par tous. D'un point de vue dynamique, le pivot permet tout au long du processus de développement de stocker des informations transverses extraites des facettes.
Le pivot est détaillé chapitre 5 page 65.

Exemple 1 *Considérons le cas d'un distributeur automatique de billets. Les spécifications de celui-ci seront, entre autres, de fournir une interface conviviale, de valider l'identité du client par les informations de sa carte et de son code secret et de contacter l'agence bancaire afin de mettre à jour le solde.*
Même si la solution choisie pour implanter ce distributeur est constituée d'un calculateur doté d'un OS multi-tâches et de plusieurs processus logiciels le pivot comportera un bloc IHM , un bloc "authentification", et un bloc "communication" qui représenteront chacun un processus logiciel et la charge de calculateur qu'il utilise. Ceci permet d'obtenir une vision fonctionnelle du système et de spécifier des exigences transverses telles que le temps de réponse sur un bloc logique. Nous verrons par la suite comment cette vision fonctionnelle est fournie par le pivot.

2.2.4 Layers

Un des problèmes majeurs induits par l'utilisation de multiples modèles pour représenter un même système est le maintient en cohérence de l'ensemble. Dans le contexte du découpage par métier des modèles, la relative orthogonalité des préoccupations des divers intervenants permet de réduire ces problèmes d'assemblage et de cohérence à la gestion des exigences transverses. Le choix de laisser le raffinement des modèles comme une information privée limite cette gestion à un unique niveau d'abstraction. Afin de gérer la cohérence sémantique entre les facettes, nous introduisons un ensemble de modèles appelés "layers[4]" liés aux préoccupations transverses, permettant soit de vérifier un ensemble de propriétés, soit de calculer des informations transverses destinées au pivot. Cette structure de layers que nous présentons au chapitre 6 page 75 s'appuie sur ces propriétés de notre découpage en modèles métier pour répondre à la problématique de la cohérence sémantique globale ainsi que la satisfaction des exigences par le système.

2.2.5 Processus de développement

Les concepts précédemment introduits prennent leur efficacité au sein d'un processus de développement. Le chapitre 7 page 103 propose une méthodologie de développement tirant partie de ces concepts dans le cadre d'un développement sans réutilisation de parties préexistantes. Celui-ci se déroule en deux phases :

1. Une phase de concertation permet de déterminer, pour chaque facette un cahier des charges. Les facettes proposent alors chacune une solution abstraite sous forme d'un Modèle Public. Un pivot temporaire est alors créé à partir des-dits modèles permettant d'en vérifier la cohérence. Ce processus est itéré jusqu'à l'obtention d'un ensemble Modèles Publics/Pivot cohérent, suffisamment détaillé et conforme aux exigences système[5].
2. Une fois la phase de négociations terminée et les Modèles Publics des différentes facettes déterminés, le processus de développement entre dans une phase dite "à modèles stables" dans laquelle les facettes réalisent effectivement leur partie du système. Au cours de cette phase la structure des Modèles Publics des facettes et celle du pivot ne changent plus. Les seuls changements pouvant intervenir sur ces modèles sont des gains de précision de leurs attributs.

Le chapitre 8 page 109 présente une application des concepts développés au cours de cette thèse au dimensionnement de plate-forme. Il s'agit de créer une plate forme matériel chargée d'exécuter un ensemble de logiciels applicatifs existant en vue de réaliser un système spécifié par un cahier des charges. Plus particulièrement, ce chapitre illustre une autre utilisation des layers, en vue non plus d'analyser un système en formation, mais de contraindre le développement d'une facette par rapport à l'existant.

2.2.6 Périmètre de la thèse

Après cette présentation générale de notre approche, il nous faut maintenant définir les frontières de notre étude. Les points critiques du développement par métier d'un système embarqué sont, d'une part la spécification de chaque facette et d'autre part l'intégration de l'ensemble en un système fonctionnel. Cette dernière est dépendante de la première : en théorie, il devrait être simple d'intégrer

[4]Nous utilisons ici l'anglicisme "Layer" car il reflète à la fois la notion de "calque" et de "couche" qui sont précisément celles que nous souhaitons exprimer.

[5]plus précisément permettant une réalisation conforme aux exigences.

des sous-systèmes parfaitement spécifiés et réalisés car cette intégration pourrait-être effectuée au niveau des spécifications puis appliquées aux sous-systèmes sensés avoir les mêmes propriétés que leur spécification, ce que l'on peut formuler ainsi : "Les spécifications d'un sous-système sont de bonnes abstractions du-dit sous-système".
Les deux propriétés permettant de se rapprocher de ce cas idéal sont les suivantes :

1. les sous-systèmes sont correctement spécifiés, c'est à dire de façon exhaustive et non ambiguë ;
2. les sous-systèmes concrets sont correctement réalisés... L'interprétation de cette phrase est largement sujette à débat. Par exemple, dans le cas des systèmes avioniques, "correctement réalisé" signifie développé via un processus normalisé par le DO178-B. Dans le cadre de cette thèse nous ne nous intéresserons pas aux activités internes des facettes (*i.e.* qui se déroulent au sein de leur partie privée). En revanche, nous étudierons la conformité entre les produits -livrables- des facettes et les Modèles Publics associés. Pour cela, nous nous appuyons sur un ensemble de propriétés dites contractuelles qui si elles sont respectées par l'implantation, garantissent la cohérence de l'intégration. La manière de garantir que le sous-système respecte bien ces propriétés n'est pas dans notre champs d'étude, mais à titre d'exemple, une application stricte du DO178-B dans le développement permet de le garantir.

Le périmètre de cette thèse se limitera donc aux modèles permettant l'intégration et la vérification de cohérence, à savoir le pivot, les layers et les Modèles Publics des facettes. Le développement des systèmes au sein des parties privées des facettes est donc hors de ce périmètre et nous n'y ferons que de rares incursions à titre *d'exemple*.

Chapitre 3

Présentation de l'étude de cas

"Méfie-toi des conseils mais suis les bons exemples."
Georges Courteline

Tout au long de la suite de ce document, nous illustrerons les différentes notions introduites par une même étude de cas qui servira de fil conducteur à cette thèse. Il s'agit de la réalisation d'un flotteur océanographique proche dans son concept de celui développé au sein du laboratoire DTN de l'ENSIETA dans le cadre du projet européen SWARM. Les informations techniques relatives à ce projet étant confidentielles, notre étude de cas ne fera que s'en inspirer librement. Ceci nous permettra en outre de modifier les solutions techniques adoptées afin de mieux illustrer certains de nos propos. Ce chapitre fournit une description générale de notre flotteur. Nous y illustrerons les concepts déjà introduits (en toute généralité) de façon à donner au lecteur une vue générale de notre approche afin de faciliter la lecture des parties plus détaillées.

3.1 Cahier des charges général

La mission de ce flotteur est de fournir une cartographie des couches de salinité de la zone dans laquelle il évolue. Pour ce faire il effectue des cycles plongée/remontée durant lesquels il stocke les données capteur afin de fournir un bilan lorsqu'il est en surface via un dispositif de communication hertzien (Iridium, Argos,...) ou de le délivrer lors de la récupération en fin de mission. Le système se décompose en deux parties fonctionnelles : d'une part un système de navigation/commande gérant les cycles plongée/remontée et pilotant les actionneurs et d'autre part un système de suivi de mission gérant les données utiles et gérant le stockage et la transmission des informations recueillies. Pour les besoins de cette thèse nous considérerons que le flotteur est un système critique à forte exigence de fiabilité. Nous ajoutons donc les exigences non fonctionnelles suivantes :

– Toutes les fonctions assurées par logiciel doivent être tolérantes à une panne ;
– L'actionneur ne doit pas être orphelin (non commandé) durant plus d'une seconde ;
– Au moins une donnée utile doit être stockée chaque seconde.

3.2 Architecture générale

Afin de pouvoir décrire statiquement les différents concepts que nous introduisons en s'affranchissant, dans un premier temps, de tout processus de développement, nous fournissons d'emblée un choix de solution architecturale. La figure 3.2 page suivante décrit cette architecture.

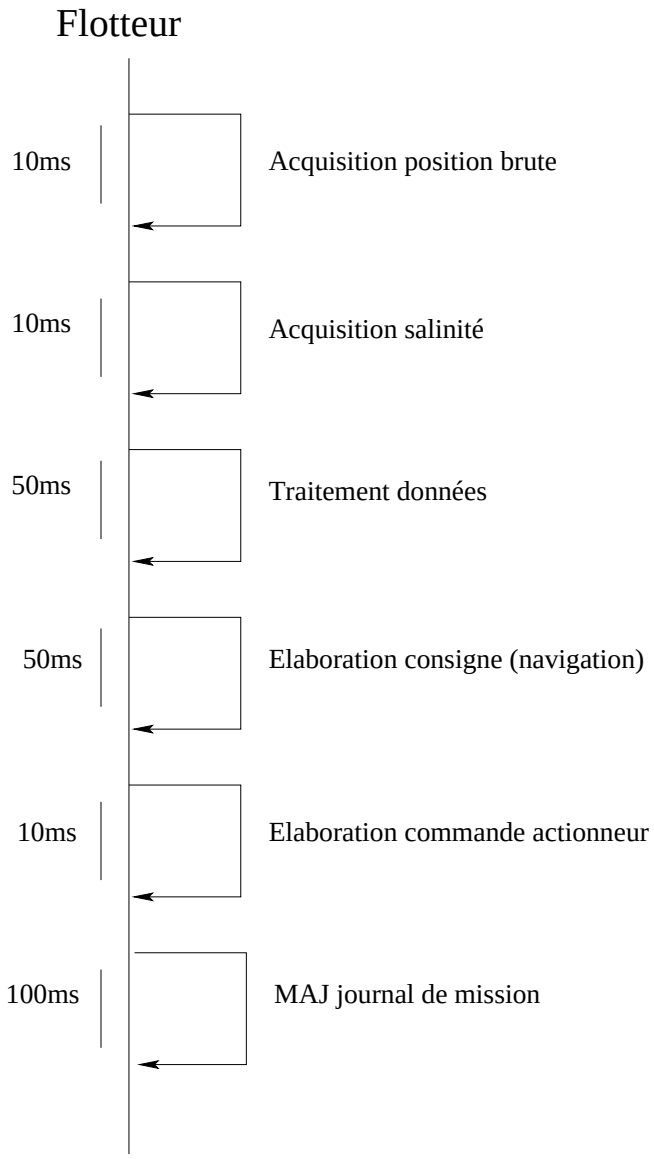


FIG. 3.1 – Diagramme de séquences des actions

Le flotteur est constitué des éléments suivants :

B1 et B2 deux bus numériques redondants chargés de la communication entre les divers éléments du système ;

C1 et C2 deux ressources de calcul redondantes exécutant chacune l'ensemble des logiciels applicatifs ;

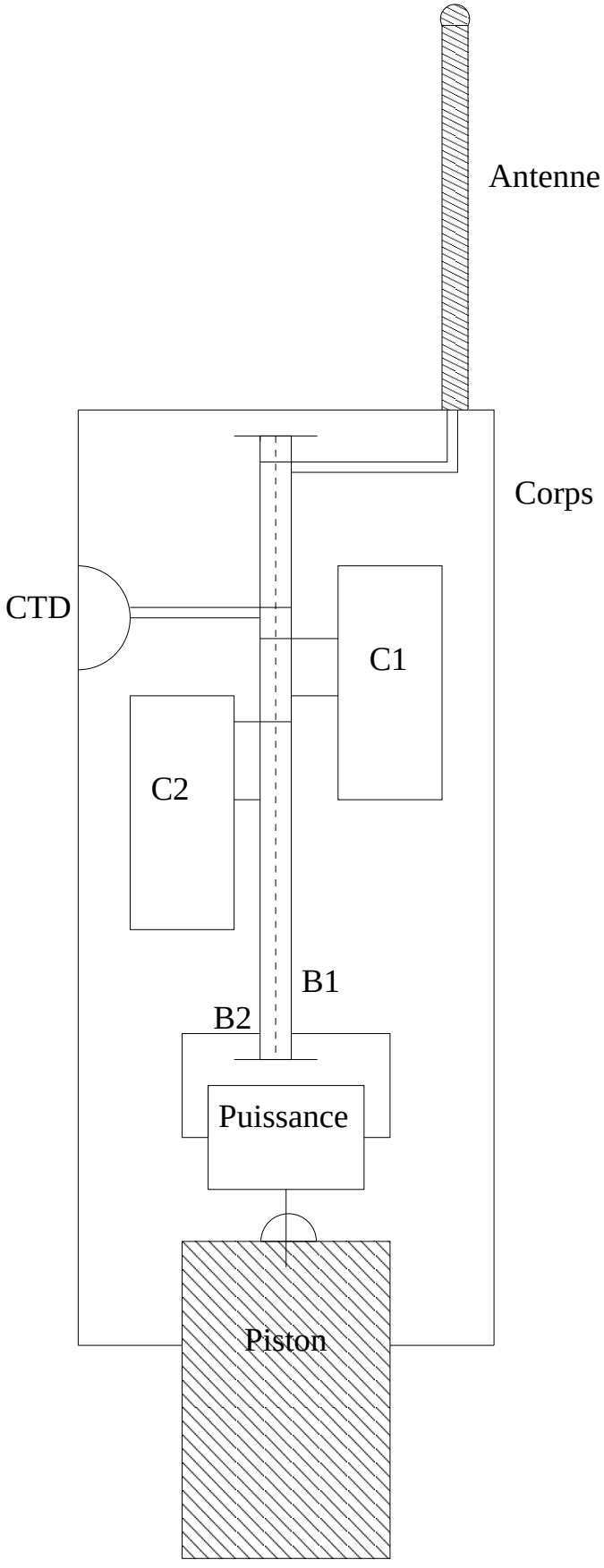


FIG. 3.2 – Schéma du flotteur

Antenne une antenne GPS/Iridium ;

Puissance une carte contrôlant le moteur du Piston (actionneur) ;

CTD un capteur océanographique utilisé pour recueillir les informations recherchées par la mission, et utilisées pour la navigation ;

3.3 Les facettes métier envisagées

Dans le cadre de cette étude nous avons identifié quatre métiers concourant au développement d'un tel flotteur :
– Les automaticiens mettent au point les lois de commande et pilotage de l'engin ;
– Les mécaniciens conçoivent les aspects mécaniques du projet (corps, piston, tapes, etc) ;
– Les électroniciens ont la charge de toute l'architecture électronique (calculateurs, bus, etc) ;
– Enfin des informaticiens développeront le logiciel embarqué.
Dans le cadre de cette thèse nous ne retiendrons que deux facettes : la facette matériel (électronique) et la facette logiciel. La partie "automatique" sera pour partie intégrée dans les spécifications du projet, le reste étant considéré comme faisant partie du logiciel. La conception mécanique reste considérée comme une facette métier mais ne sera pas détaillée car les problèmes qu'elle induit dépassent largement le cadre de notre étude. Dans la perspective d'étendre notre approche à des domaines métier non-informatiques, nous fournirons toutefois un certain nombre de points d'ancrage pour cette facette ainsi que des pistes pour les utiliser (cf. 4.3.4 page 63).

3.3.1 Facette logiciel

La facette logiciel développe l'ensemble des logiciels embarqués "applicatifs" (par opposition avec les logiciels de très bas niveau tel les ordonnanceurs et les protocoles de communication couche basse, qui sont à la charge de la facette matériel). Le chapitre 4.2 page 47 décrit cette facette de façon détaillée. Cette facette possède ses propres spécifications (induites par les exigences globales) que nous détaillons maintenant.

Spécifications

Les spécifications fonctionnelles de cette facette sont les suivantes :
– fournir une commande aux actionneurs permettant de suivre une trajectoire de consigne (donnée de mission) avec une précision suffisante ;
– traiter les données fournies par les différents capteurs ;
– calculer et émettre un bilan de mission à destination de l'antenne Iridium à chaque remontée.

La facette devra fournir des processus de secours et expliciter la gestion de cette redondance.
Ces exigences de haut niveau induisent des sous-exigences qui seront fonction des choix d'implantation de la facette logiciel et aussi de ceux des autres facettes, éventuellement arbitrés par le MOE.

3.3.2 Facette matériel

Cette facette qui conçoit l'architecture matérielle du flotteur est décrite en détail
paragraphe 4.3 page 59. Elle conçoit les ressources de calcul, les liens de communication et les

capteurs/actionneurs du système. Elle fournit en outre une partie des logiciels couche basse, tels que les ordonnanceurs et les protocoles de communication.
Nous explicitons maintenant les spécifications de cette facette.

Spécifications

La facette matériel devra satisfaire les exigences suivantes :
– fournir une puissance de calcul suffisante pour exécuter les codes fournis par la facette logiciel ;
– l'architecture devra permettre la réalisation des exigences globales de tolérance aux fautes et de probabilité de pannes (ce qui impliquera des solutions de redondance des ressources) ;
– tenir compte des contraintes d'encombrement et de consommation électrique ;
– fournir les capteurs nécessaires.

3.4 Le pivot

Le pivot, modèle du MOE, sera constitué principalement des éléments suivants :
– Une fonction de traitement des données capteur brutes ;
– Une fonction de navigation calculant le nouveau volume du piston permettant d'atteindre la profondeur de consigne en fonction des données filtrées issues de la fonction de traitement ;
– Une fonction de commande calculant le nombre de tours à effectuer par le moteur pas à pas du piston pour obtenir le volume de consigne issu de la fonction de navigation ;
– Une fonction capteur ;
– Une fonction de suivi de mission chargée de gérer l'élaboration d'un bilan de la mission à envoyer à chaque remontée en surface
Pour chacune de ces fonctions, le pivot comportera une fonction de secours.

3.5 Les layers

Les layers que nous développerons pour cette étude de cas sont les suivants :
– Le layer mapping, chargé de créer les éléments du pivot en liant des objets de facettes matériel et logiciel.
– Le layer performances temps-réel chargé d'évaluer les temps de réponses des différentes fonctions du système ;
– Un layer vérification chargé de vérifier des propriétés temps réel en s'interfaçant avec un logiciel de vérification formelle ;
– Le layer reconfigurabilité permettant d'étudier le comportement du système en cas de perte d'une partie de ses fonctions.
Nous évoquerons rapidement d'autres layers envisageables, sans toutefois les détailler.

3.6 Conclusion de la partie introductive

La suite de ce document est structurée en trois parties. Au sein de la première, nous développerons les différents modèles et techniques que nous proposons et sur lesquels s'appuie notre méthodologie. Dans une deuxième partie nous exposerons notre méthodologie au travers d'un processus de développement mettant en œuvre les modèles et techniques précédemment évoqués.
La dernière partie fournit une synthèse du document et tente d'élargir le champ d'application de nos travaux.

Deuxième partie

Modèles et structures

Introduction

Cette partie décrit les concepts et les structures qui forment les fondements de notre approche :

– Les facettes métier : elles sont relatives aux intervenants et sont donc la cheville ouvrière du système. Nous décrirons de façon générale comment nous modélisons une facette, puis nous traiterons deux exemples : la facette matériel et la facette logiciel ;
– Le pivot : dédié au maître d'œuvre, ce modèle peut être vu comme un modèle d'analyse, dans la mesure où c'est lui qui supporte l'analyse de la conformité du système en regard des exigences globales, ou encore un modèle de stockage puisqu'il reçoit les données transversales calculées par les layers ;
– Les layers : ces modèles d'intégration ont la charge de maintenir la cohérence du système et de calculer les informations de type "transverse" nécessaires à l'analyse globale du système.

Ces concepts sont décrits, au sein de cette partie, dans un contexte purement statique, hors de tout processus de développement, les aspects dynamiques faisant l'objet de la partie suivante.

Chapitre 4

Facettes métier

"La solution du bon sens est la dernière à laquelle songent les spécialistes.
Bernard Grasset.

4.1 Généralités

Comme évoqué précédemment, dans le cadre du développement d'un système comportant plusieurs intervenants métier (sous-traitants) ou même l'utilisation de COTS[1] pour des pans métier complets du système, les parties correspondantes sont développées (presque) indépendamment les unes des autres. La méthodologie de développement et les structures de modèle qu'elle met en œuvre doivent refléter cette relative autonomie des intervenants. Dans ce but nous introduisons la notion de facette qui modélise un pan métier du système. Par abus de langage nous dénommerons par ce terme tant la partie du système concernée (modèles et réalisation) que l'équipe qui en a la charge, c'est pourquoi il nous arrivera de doter les facettes d'une volonté propre. La figure 4.1 page suivante illustre l'organisation générale d'une facette. Celle-ci est composée (à l'image d'un composant logiciel) d'une partie interface publique et d'une partie réalisation privée.
La partie publique d'une facette s'appelle un Modèle Public (MP). Celui-ci a une double fonction :

1. Il fournit au pivot les informations dont il a besoin concernant la facette ;
2. Il demande au pivot les informations dont la facette à besoin pour son développement cohérent par rapport au reste du système.

Les Modèles Publics des facettes (ainsi que le pivot) doivent être exprimés dans un langage commun compris de tous les intervenants.

Remarque 1 *D'autres langages peuvent, bien sûr, être utilisés, surtout dans le cas où les facettes considérées sont suffisamment connexes pour pouvoir utiliser un langage plus spécialisé.*

[1] Composants sur étagère

Les Modèles Publics comportent deux types d'objets :

1. Les objets offerts qui fournissent une abstraction des objets développés par la facette.
2. Les objets requis sont des objets virtuels (abstraction d'objets développés par d'autres facettes) dont les attributs sont calculés par des layers (voir chapitre 6 page 75) à partir d'éléments de pivot et d'objets offerts par d'autres MPs.

Exemple 2 *Si l'on considère un système formé d'un capteur, d'une commande simple et d'un actionneur, l'implantation de la commande (en C par exemple) sera un objet offert par la facette logiciel. En revanche l'actionneur et le capteur seront des objets requis pour cette même facette. Ils seront vus par la facette comme des processus logiciels et lui permettront de spécifier les données échangées entre eux et les objets qu'elle développe.*

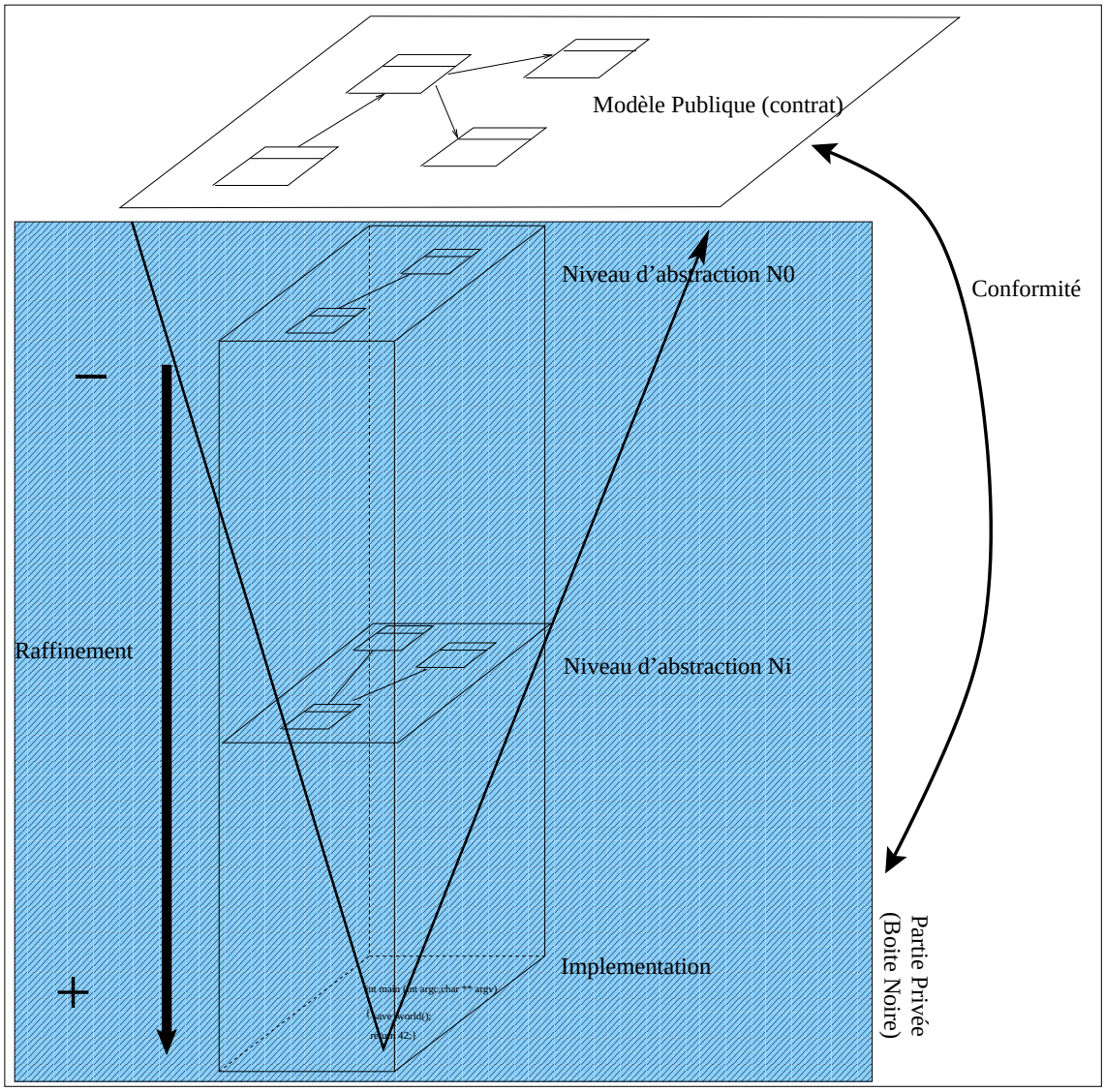


FIG. 4.1 – Organisation d'une facette

Le développement effectif de la facette s'opère au sein de sa partie privée. Au sein de celle-ci la facette développe son sous-système en utilisant un (ou plusieurs) langage(s) spécialisé(s) ainsi qu'une méthodologie de son choix. La relation liant les modèles de la partie privée et le Modèle Public d'une facette sont d'ordre contractuel. Le MP spécifie les éléments sur lesquels porte la conformité. La définition de la conformité dépend de l'élément considéré et du contexte de développement du système. Elle résulte généralement d'un consensus entre le donneur d'ordre et le maître d'œuvre. La facette s'engage à ce que sa réalisation *finale* soit conforme à son MP, non à ce qu'à tout instant du développement ses modèles soient en relation de raffinement avec le MP.
Le périmètre de cette thèse s'arrête aux modèles d'intégration, donc aux Modèles Publics. La partie privée des facettes, et par suite la relation de conformité entre MP et partie privée, se situe en dehors du champ de notre étude, nous la verrons donc comme une boîte noire.

Remarque 2 *Si la structure et l'organisation générale d'une facette sont proches de celles de la notion de composant, il existe une différence philosophique forte entre les deux concepts. Les composants ont pour vocation première la réutilisation de ce qu'ils encapsulent (du code le plus souvent). Leurs interfaces prétendent donc à une forte généricité et déplacent l'effort de développement vers l'intégration. Les MPs des facettes, au contraire, se veulent être un outil d'intégration continue des métiers dans le développement d'un système. Un MP est conçu pour un système donné et ne peut, en général,pas être réutilisé. Les problèmes de réutilisation (évoqués au chapitre 8 page 109), nécessitent de recréer un MP a posteriori, à partir de l'existant par des techniques de retro-ingénierie.*

Nous allons maintenant détailler deux exemples de facettes, la facette logiciel et la facette matériel. Pour chacune de ces facettes, nous donnerons d'abord un modèle général détaillé puis son instanciation sur notre étude de cas. Ce que nous appelons modèle général doit être vu comme un guide méthodologique permettant de créer ce type de facette. En effet, notre but est de disposer d'un modèle d'une part adapté aux systèmes embarqués proche de notre étude de cas, d'autre part suffisamment simplifié pour être instancié "sur papier" tout en étant assez explicite pour permettre une généralisation aisée de la méthode. Enfin, il est important de noter que la construction d'une facette dépend des autres facettes impliquées dans la réalisation du système et des exigences à respecter. La règle générale étant que toute information qui n'est pas explicitement demandée par le MOE ou les autres facettes doit être cachée.

4.2 Facette logiciel

La facette logiciel a la charge de développer le logiciel "applicatif" du système, une partie du logiciel de bas niveau pouvant être développée au sein de la facette matériel. La définition du domaine de cette facette dépend de la nature du système et, de l'appréciation du MOE et des divers intervenants impliqués. Son Modèle Public fournit une abstraction des processus qu'elle développe et ainsi que des données qu'ils échangent entre eux et avec d'autres objets du système. Les informations qu'elle fournit sont d'ordres quantitatif, qualitatif, structurel et comportemental. Dans le cadre de cette thèse nous ne considéreront que des processus périodiques.

4.2.1 Éléments contractuels

Le MP de cette facette fait office de contrat entre la facette et le MOE. Par conséquent les éléments contractuels suivants doivent y figurer :

- La granularité minimum : chaque objet de type `Process` sera implémenté de telle sorte qu'il puisse être vu par les autres facettes comme un processus indépendant, en particulier deux `Process` pourront être distribués sur deux calculateurs différents ;
- Les échanges de données : les objets `Process` s'échangent exactement les données (`S_DATA`) spécifiées dans le MP.
- Les attributs des objets : tous les attributs spécifiés dans le méta-modèle devront décrire le logiciel concret, éventuellement sous forme d'encadrement si une tolérance est acceptée :
  - taille : la taille de l'objet en octets ;
  - coût : le coût financier du développement de l'objet ;
  - nbInstrMax : le nombre d'instructions élémentaires maximum effectuées par un `Process` au cours de sa période ;
  - période : la période d'un `Process`.
  - le comportement : un automate décrivant le comportement attendu de chaque processus.

Une fois listés les éléments contractuels liant la facette et le reste du système, nous sommes en mesure de produire un méta-modèle du MP de la facette logiciel.

4.2.2 Méta-modèle

La figure 4.2 montre le méta-modèle du MP de la facette logiciel.

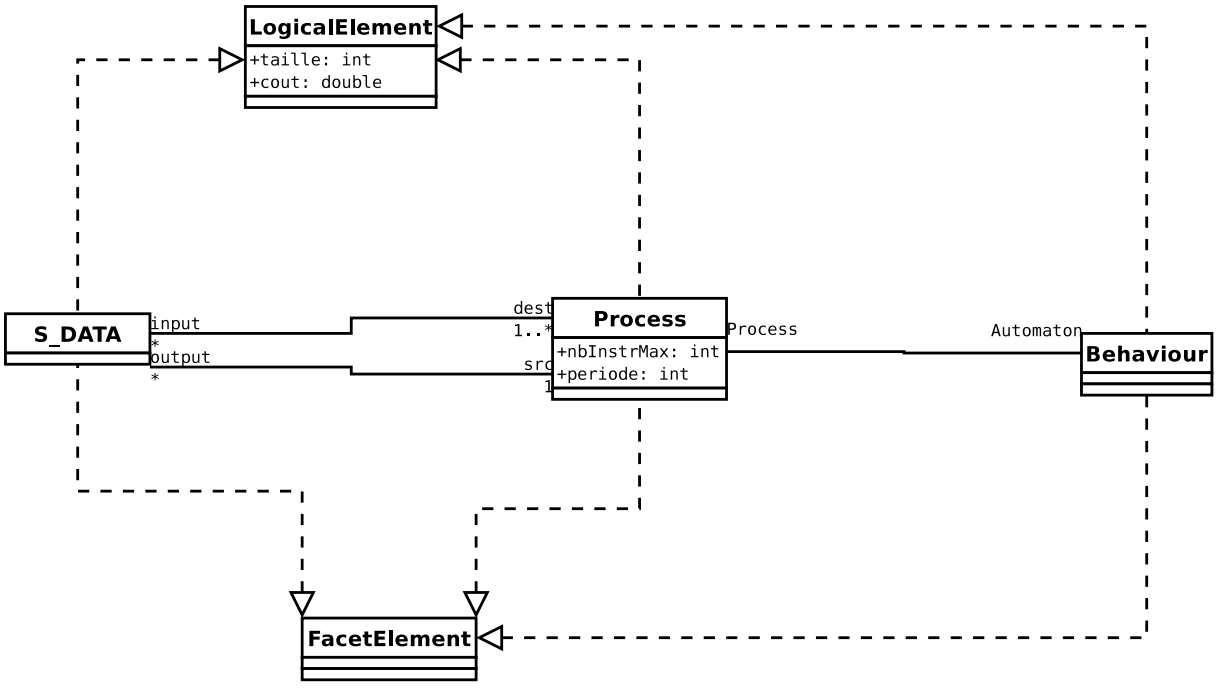


FIG. 4.2 – Méta-modèle du MP de la facette logiciel

Ce méta-modèle est composé des classes `Process` et `S_Data`. Tous deux réalisent les interfaces `LogicalElement` et `FacetElement`. `LogicalElement` permet de manipuler les objets logiques ou immatériels. Cette interface trouve son utilité lors de l'étude des propriétés physiques du système

(voir 4.3.4 page 63). Elle est dotée des attributs `coût` et `taille`. Le premier désigne le coût financier du développement de l'objet (en euros), le second la taille de l'objet (en octets).
`Process` décrit les processus logiciels. Ses attributs sont :
– `nbInstrMax` : le nombre d'instructions "élémentaires" maximum nécessaires à la terminaison du processus.
Cette information peut être par exemple, fournie par des techniques d'analyse statique du code. Le jeu d'instructions élémentaires et leur signification sont alors supposés connus de tous les intervenants concernés ;
– `periode` : tous les processus sont supposés périodiques. Cet attribut désigne la période d'un processus en millisecondes.

La facette logiciel ne contient pas de définition explicite des liens de communication. Les communications entre processus sont modélisées par les données échangées. Nous considérons que les liens de communication sont du domaine de la facette matériel. D'un point de vue purement logiciel, une communication est entièrement spécifiée par deux processus (émetteur/récepteur) et deux données (sortante/entrante). Les données sont modélisées par la classe `S_Data`.
La structure de ce méta-modèle indique qu'un processus peut émettre et recevoir plusieurs données et que les communications se font "de un vers plusieurs".

Le problème particulier de la spécification du comportement des processus.

L'information principale que la facette logiciel doit fournir aux autres facettes est le comportement abstrait de ses processus. En effet cette facette a la charge de développer un ensemble de logiciels ayant chacun une fonctionnalité exprimée par le cahier des charges. Le comportement abstrait de chaque processus constitue alors une sorte de contrat décrivant aux autres corps de métier ainsi qu'à l'architecte système ce que les processus finaux réaliseront.
Se pose alors la question de la sémantique de ces processus. Plusieurs principes apparaissent assez clairement :

1. Le comportement d'un grand nombre de systèmes embarqués critiques est du type échantillonné. Les processus sont périodiques, collectent leurs données d'entrée en début de chaque période, réagissent à la fois à la présence mais aussi l'absence de certaines entrées, puis s'exécutent et produisent leurs sorties au cours de la période.
2. En revanche, dans certains cas (notamment dans un contexte de temps réel souple), et selon le dogme de la séparation des métiers et des préoccupations, le concepteur logiciel peut être amené à ne pas faire d'hypothèse sur le temps d'exécution réel de ses processus, et notamment à considérer que ce temps d'exécution n'est pas nul ou négligeable. Dans ce cas, selon les performances de la plateforme d'exécution et de la stratégie d'ordonnancement (performances inconnues dans la facette logiciel), il est possible que les processus soient interrompus par la fin de la période (on dit alors qu'ils ratent leur échéance), et doivent repartir dans un état prédéterminé.
3. Enfin, toujours selon le dogme de la séparation des métiers et des préoccupations, les comportements temporels exprimés dans la facette logiciel embarqué ne peuvent être que qualitatifs et non quantitatifs. Les précisions quantitatives, du type attente d'un délai de temps exprimé en temps physique (en seconde, miliseconde…), n'ont pas de sens dans la mesure ou ni les performances ni la stratégie d'ordonnancement ne sont (ni ne doivent être) connues du concepteur logiciel. Ces informations font l'objet des préoccupations du layer "temps réel". Seuls des comportements temporels discrets tels que "attente de *n* périodes", ou de façon

générale “attente de *n* occurrence d’un événement *e*” sont autorisés par le dogme de la séparation des préoccupations.

Malheureusement, ces trois principes tels qu’exprimés ci-dessus sont incompatibles avec les formalismes et les sémantiques des langages temps réel classiques existants :

– Les langages synchrones (tels que SyncChart, Lustre, Signal, Esterel...), bien que conformes aux principes 1 et 3, sont incompatibles avec le principe 2. Ils reposent en effet sur l’hypothèse forte que les temps d’exécution sont négligeables, et par suite que les processus ne rateront jamais leur échéance. Nous ne sommes pas là dans un schéma de conception collaboratif par séparation des préoccupation, mais dans un schéma descendant, à sens unique, du logiciel vers le matériel, ce dernier héritant des contraintes de performances exigées par le logiciel.
  Notons en revanche que sous l’hypothèse que le troisième principe puisse être relâché (aucun processus ne rate son échéance - temps réel dur), alors la programmation synchrone du type SyncChart ou Lustre serait compatible avec le besoin exprimé.
– Les langages asynchrones (tels que SDL, Lotos...) sont de leur coté en général incompatibles avec le principe 1 (réaction à l’absence d’une entrée, hormis programmation d’un chien de garde temporisé ce qui est contraire au troisième principe).

L’incompatibilité de ces langages avec le problème posé dans cette thèse tient principalement au fait qu’ils n’ont pas été conçus pour modéliser des comportements abstraits (du type contractuel) dans un cadre dirigé par les points de vue. L’exotisme devient alors ici une nécessité. Pour ce faire, nous avons donc été amenés à définir notre propre formalisme, décrit ci-après, afin de spécifier le comportement des processus de la facette logiciel. Outre le fait de coller ainsi exactement aux besoins de la spécification par séparation des préoccupations, le second avantage de cette démarche est d’être aussi indépendant que possible de choix technologiques *a priori*. De plus, le formalisme que nous proposons a pour vocation d’être suffisamment simple pour pouvoir être traduit dans nos différents layers, notamment le layer vérification, dans des langages existants, par exemple le langage des automates temporisés.

Remarque 3 *notons que le fait de vouloir traduire le comportement des processus décrit dans le formalisme introduit ci-après dans un langage existant n’est pas contradictoire avec les arguments développés ci-dessus démontrant l’incompatibilité du besoin issu de la séparation des préoccupations avec les langages existants. La cible d’une telle traduction est en effet un “layer”, réceptacle formel pour un point de vue particulier à la fois des spécifications de la facette logiciel et des performances issues de la facette matériel.*

Syntaxe

Soit P un processus du MP de la facette logiciel ; son comportement est défini par l’automate $P.Automaton =< s_0, S, \Sigma, T >$ où S désigne l’ensemble des états de l’automate, Σ l’ensemble des étiquettes des transitions, T la fonction de transition et s_0 l’état initial.
L’ensemble Σ est défini par : $\Sigma = \bigcup_{i \in P.input}(\{i?\} \cup \{\#i?\}) \cup \bigcup_{i \in P.output}(\{o!\}) \cup \{\gamma\}$ où :

– $i?$ est une étiquette signifiant que la donnée i est disponible et donc peut être lue ;
– $\#i?$ est une étiquette signifiant que la donnée i n’est pas disponible. Ceci permet de modéliser une attente non-bloquante de la donnée i ;
– $o!$ est une étiquette traduisant l’émission de la donnée o ;
– enfin γ est une étiquette modélisant un événement “tick” présent à chaque début de période du processus P. Si au début d’une période, l’automate est dans un état source d’une transition étiquetée par γ, alors (sous réserve que la garde de cette transition soit satisfaire), cette transition

peut être franchie. Cette étiquette γ permet ainsi de spécifier le comportement attendu de l'automate lors du début de chaque période.

Les transitions t de T sont définies comme des quadruplet : $\forall t \in t, t = (s, e, o, s')$ avec :
- $s \in S$: l'état d'origine de la transition ;
- $s' \in S$: l'état de destination de la transition ;
- e est la macro étiquette de la transition définie par la syntaxe $e ::= \gamma | g$, c'est-à-dire définie comme étant soit l'étiquette γ, soit une garde (une étique composée) définie elle même par la syntaxe : $g ::= i? | \#i? | g \wedge g | g \vee g | true$ où $i?$ et $\#i?$ sont des étiquette de Σ ;
- $o \in (\bigcup_{o \in P.output} \{o!\})$: l'étiquette de sortie de la transition.

Une transition est franchissable si et seulement si, sont état d'origine est occupé et si sa macro étiquette e est satisfaite par le contexte courant du processus (ce contexte est formé par les données d'entrée lues en début de période, ainsi que l'événement γ lors du début de la période).

Sémantique

La sémantique sous-jacente de ce modèle s'appuie sur les principes suivantes :
- Les processus acquièrent leurs entrées une fois par période par polling (la donnée est présente ou non) en début de période. Ces entrées sont stockées dans un "contexte" $C \subseteq \bigcup_{i \in P.input}(\{i\}) \cup \{\gamma\}$.
- Les transitions gardées par une étiquette γ ne peuvent être franchies que lors d'un début de période.
- Enfin, les transitions de l'automate sont tirées en séquence en fonction de la présence ou l'absence de données dans le contexte courant. Les labels $i?$ et $\#i?$ permettent ainsi de modéliser le comportement d'un procesus réagissant à la présence ou l'absence de la donnée i dans le contexte acquis au début de la période courante.

Afin definir un peu plus précisément la sémantique d'un processus, il est nécessaire de définir la sémantique des gardes de ses transitions. Cette sémantique est donnée par la relation "Le contexte C satisfait la garde g" (notée $C \models g$) définie par induction comme suit :
- $C \models true$
- $C \models i? \Leftrightarrow i \in C$
- $C \models \#i? \Leftrightarrow i \notin C$
- $C \models g \wedge g' \Leftrightarrow C \models g \; et \; C \models g'$
- $C \models g \vee g' \Leftrightarrow C \models g \; ou \; C \models g'$

La sémantique de l'automate d'un processus (et notamment le mécanisme de polling des entrée en début de chaque période) est définie par les règles d'inférence suivantes. Ces règles définissent un graphe dont les noeuds sont des configurations de l'automate du processus P : ces configurations étant constituées d'un état de l'automate $< s, S, \Sigma, T >$ ainsi que d'un contexte $C \subseteq \bigcup_{i \in P.input}(\{i\})$.

Règle 1 : Lors du changement de période, soit I l'ensemble des entrées lues par polling lors de ce début de période. Cet ensemble contient l'événement γ (début de la période). Si une transition étiquetée par γ est franchissable, alors elle est franchie. Le contexte sans l'événement γ est ensuite mis à jour dans le processus pour permettre le tir des transitions suivantes.

$$\frac{\begin{array}{c} I \subseteq \bigcup_{i \in P.input}(\{i\}) \cup \{\gamma\} \\ \gamma \in I \\ \exists \, o, s' \; tel \; que \; s \xrightarrow{\gamma, o!} s' \in T \end{array}}{(< s, S, \Sigma, T >, C) \xrightarrow[\{o\}]{I} (< s', S, \Sigma, T >, I/\{\gamma\})}$$

Règle 2 : Si en revanche, lors du changement de période, aucune transition étiquetée par γ n'est franchissable, alors aucune transition n'est franchie et seul le contexte est mis à jour (sans l'événement γ).

$$\frac{\begin{array}{c} I \subseteq \bigcup_{i \in P.input}(\{i\}) \cup \{\gamma\} \\ \gamma \in I \\ \forall\, e,o,s' \text{ tel que } s \xrightarrow{e,o} s' \in T,\ e \neq \gamma \end{array}}{(< s,S,\Sigma,T >,C) \xrightarrow{I}_{\emptyset} (< s,S,\Sigma,T >,I/\{\gamma\})}$$

Ces deux premières règles expriment le comportement du processus lors de l'acquisition d'un ensemble d'entrée en début de période. Inversement, elles expriment également les conditions de franchissement d'une transition étiquetée par γ. En particulier, ces transitions sont de fait prioritaires par rapport aux autres lors des débuts de période.

Règle 3 : Les transitions intra-période (non étiquetées par γ) franchissables sont tirées sans changement du contexte.

$$\frac{\begin{array}{c} C \subseteq \bigcup_{i \in P.input}(\{i\}) \\ \gamma \notin I \\ s \xrightarrow{g,o!} s' \in T \\ C \models g \end{array}}{(< s,S,\Sigma,T >,C) \xrightarrow{\emptyset}_{\{o\}} (< s',S,\Sigma,T >,C)}$$

Le comportement d'un processus est donné par l'enchaînement des transitions du graphe de configurations défini par les trois règles inductives ci-dessus.

Exemple 3 (Illustration) *La figure 4.3 page suivante fournit un exemple d'automate. Le processus qu'il décrit reçoit les entrées i_1 et i_2, effectue ensuite un calcul puis envoie o_1, o_2 et o_3. Si l'une au moins, des entrées n'est pas disponible il passe en erreur. Le processus est réinitialisé lorsque le début de période survient (transitions γ). En particulier si celui-ci arrive durant le calcul, il ne se terminera pas et les sorties ne seront pas émises.*
Lors du démarrage du processus celui-ci est dans l'état `INIT`

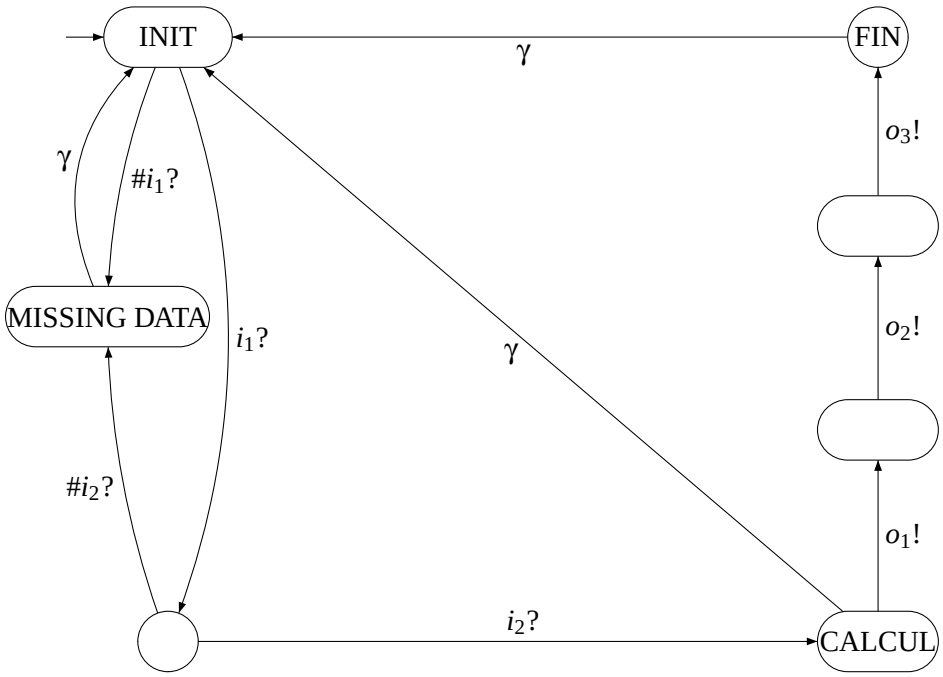


FIG. 4.3 – Exemple d'automate

4.2.3 Facette logiciel du flotteur

Nous allons maintenant détailler le contenu du MP de la facette logiciel de notre étude de cas. La figure 4.4 fournit le diagramme de séquence illustrant les communications entre les processus, celui-ci est donné à titre indicatif afin de faciliter la compréhension des modèles structuraux et comportementaux fournis dans la suite du document.

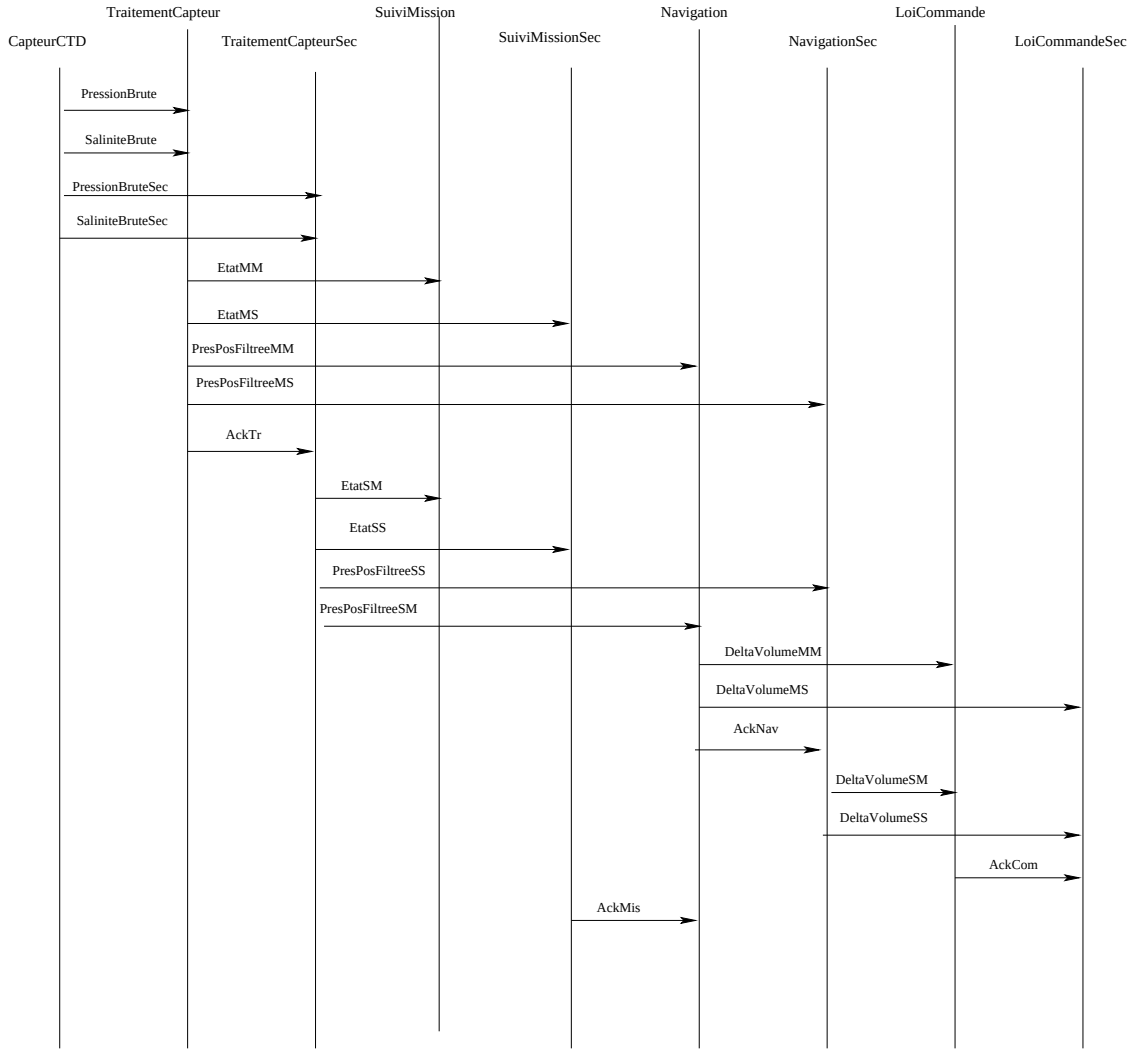


FIG. 4.4 – Séquencement des communications

Modèle structurel

La figure 4.5 page suivante constitue la partie structurelle du Modèle Public de la facette logiciel de notre étude de cas. Les processus qu'il fournit sont :

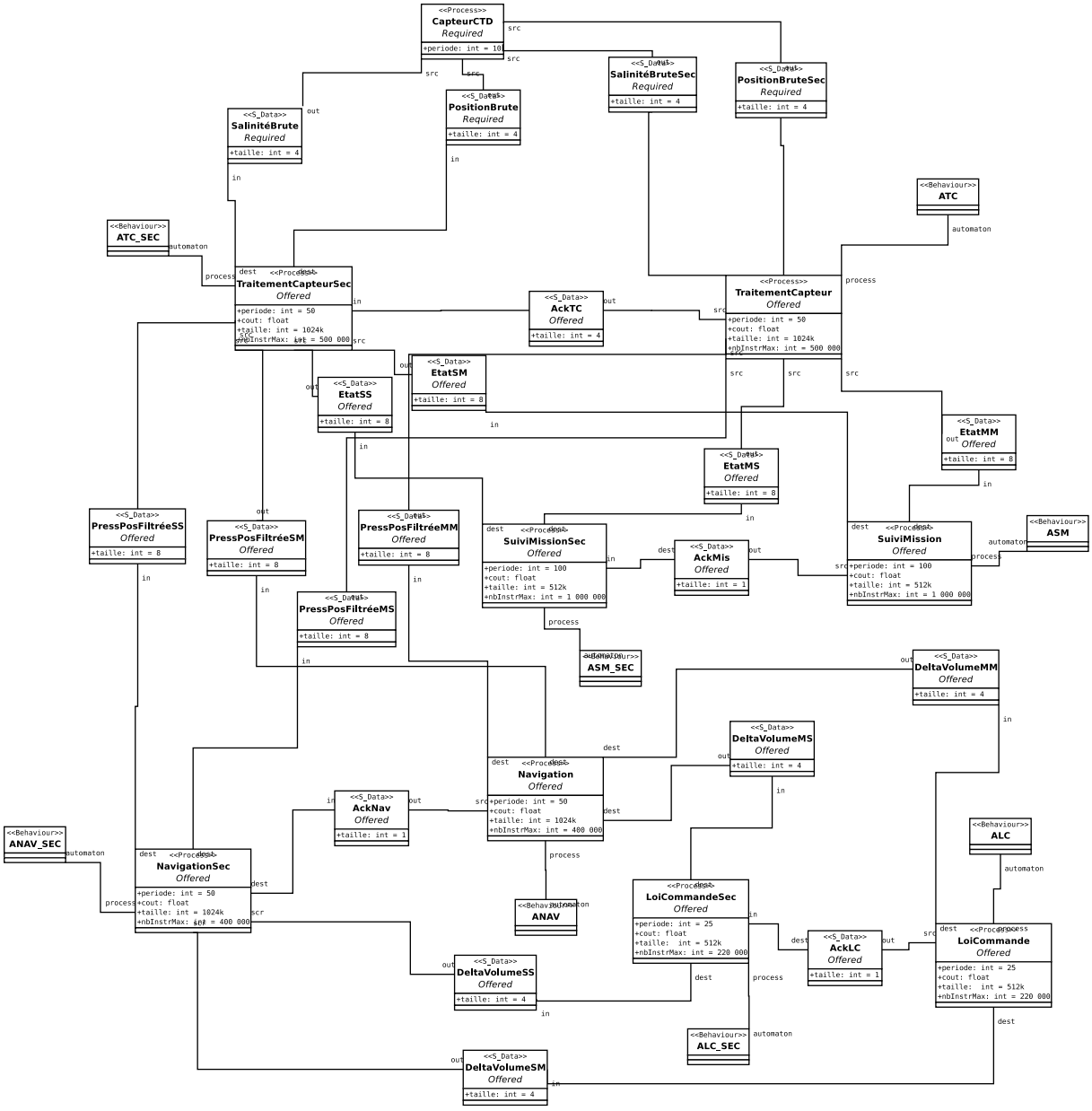


FIG. 4.5 – Facette logiciel - Modèle Public

– `TraitementCapteur` : ce processus reçoit les données brutes provenant du capteur (`PressionBrute`, `SaliniteBrute`) puis les filtre et les formate à destination des autres processus ;
– `SuiviMission` attend l'état (`Etat`) du système (pression, salinité) filtré et formaté. S'il le reçoit, il le stocke dans le journal de mission, sinon il ne fait rien.
– `Navigation` calcule la variation de volume à appliquer au piston pour atteindre la prochaine profondeur à partir de la donnée `PressPosFiltree`. Si la donnée n'est pas présente, il n'envoie aucune consigne.
– `LoiCommande` commande le système (aucune sortie) pour obtenir la variation de volume ΔV demandée par `Navigation`. Si la consigne n'est pas reçue il applique la même consigne (dans ce cas le système est considéré comme non-réellement commandé).

En outre, pour chaque processus `P` le MP fournit un processus `PSEC`. Chaque processus maître émet un acquittement (les données `Ack*`) à destination de son processus de secours. Si celui-ci ne reçoit pas cet acquittement durant deux périodes, il considère que son processus maître est en défaillance et se comporte comme lui (à l'exception, bien sûr, de l'envoi de l'acquittement).
Le tableau 4.6 récapitule les principales informations concernant les processus du MP présenté figure 4.5 page précédente.

Nom	periode	taille	nbInstrMax
TraitementCapteur	50	1024k	500 000
TraitementCapteurSec	50	1024k	500 000
SuiviMission	100	512k	1 000 000
SuiviMissionSec	100	512k	1 000 000
Navigation	50	1024k	400 000
NavigationSec	50	1024k	400 000
LoiCommande	25	512k	220 000
LoiCommandeSec	25	512k	220 000

FIG. 4.6 – Synthèse des informations du MP logiciel

La figure 4.7 page suivante représente l'automate du processus `TraitementCapteur`. Détaillons le fonctionnement de cet automate :
– au départ le processus est dans l'état `Attente`. Lorsqu'il a acquis ses entrées il passe dans l'état `CALCUL` ; s'il se trouve encore dans cet état à la fin de la période, il se réinitialise ;
– l'état `CALCUL` abstrait le calcul des données filtrées. Il émet la donnée `PresPosFiltree` qui contient à la fois la pression et la position (puisque ces deux données sont à destination du même processus et envoyées quasi-simultanément) ; Les deux données sont ensuite envoyées simultanément
– il passe ensuite dans l'état `FIN`. De cet état il peut tirer une transition γ, qui le ramène dans l'état `Attente` dès la fin de la période.

La figure 4.8 page 58 représente l'automate du processus de secours du processus `TraitementCapteur`. Si celui-ci n'a pas reçu d'acquittement de son processus maître durant deux périodes consécutives, il se comporte alors comme lui. Le comptage des périodes sans réception d'acquittement est réalisé par des transitions γ.

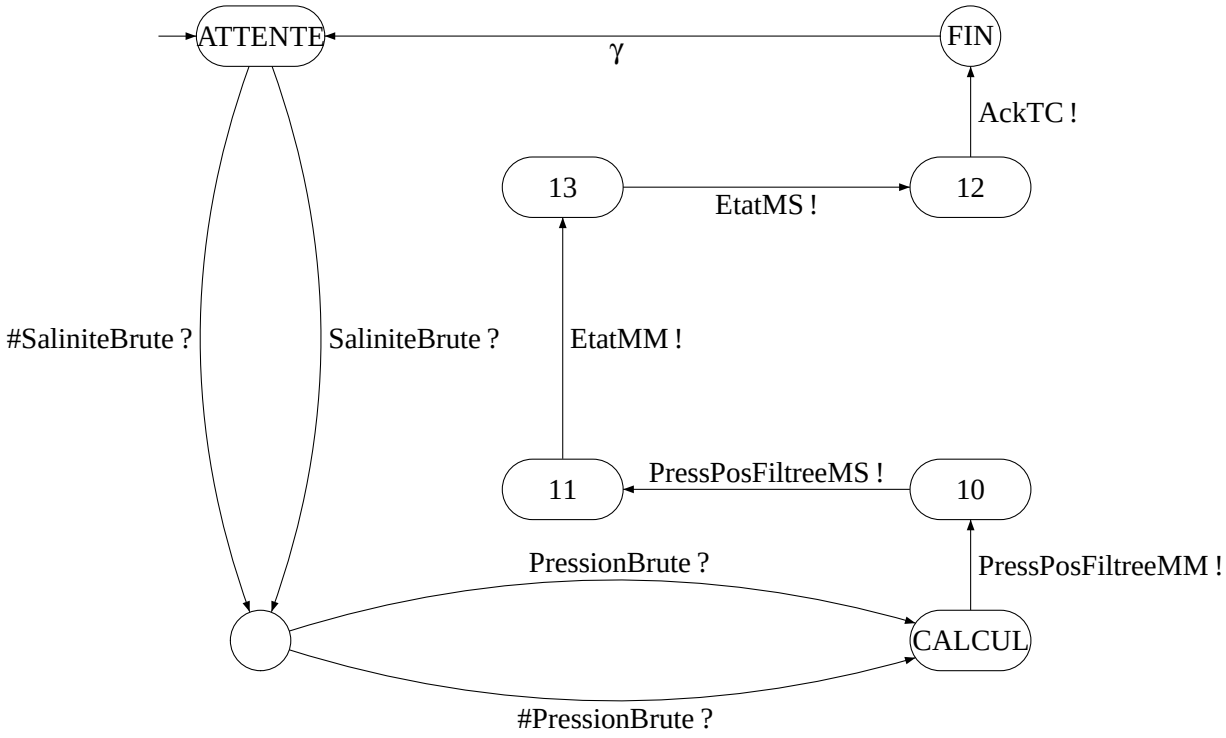


FIG. 4.7 – ATC : Automate du processus TraitementCapteur

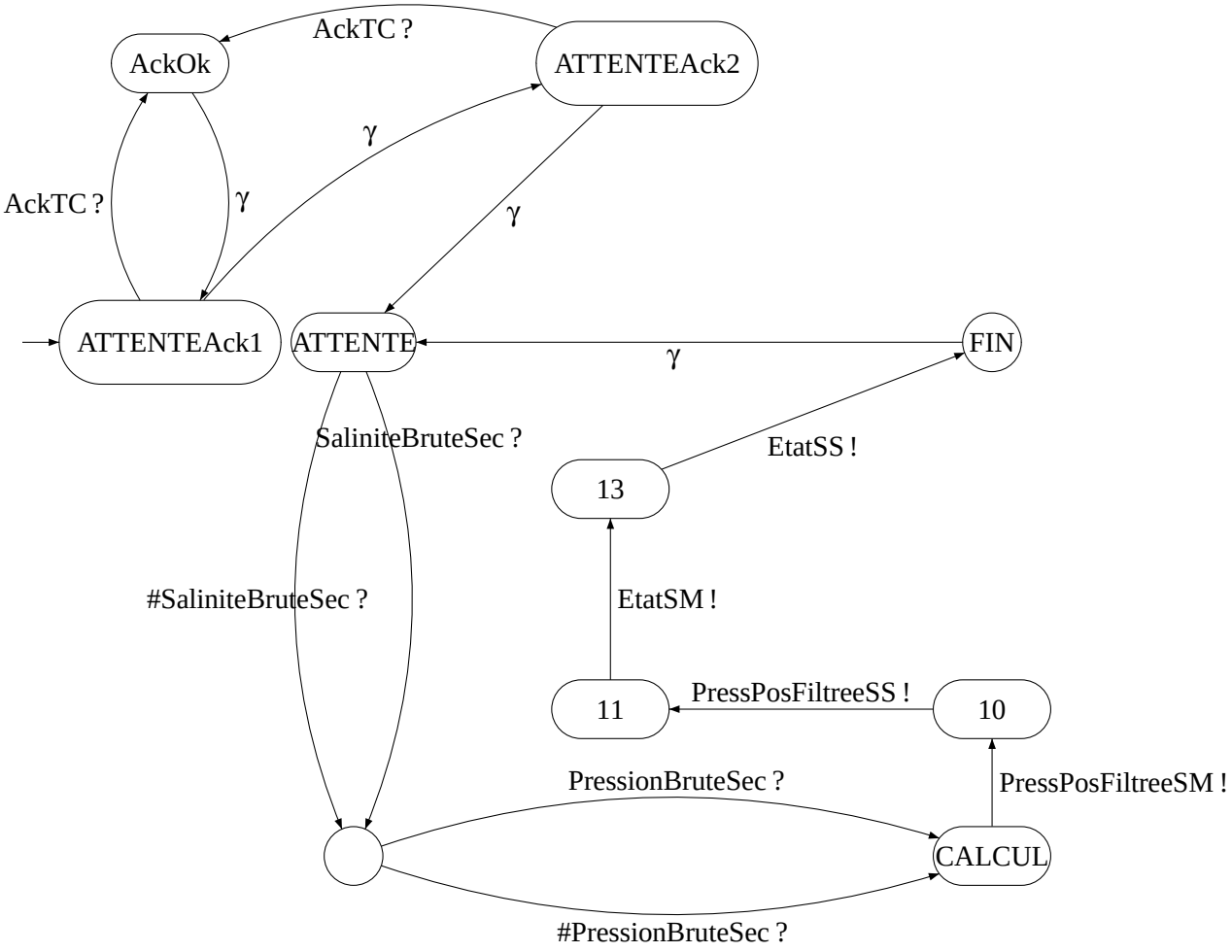


FIG. 4.8 – ATC_SEC : Automate du processus TraitementCapteurSec

4.3 Facette matériel

La facette matériel a la charge de la plate-forme exécutive du système. Cette facette, dans sa globalité, et plus particulièrement sa partie privée, conçoit les ressources de calcul, les liens de communication et les capteurs/actionneurs du système. Elle fournit en outre une partie des logiciels couche basse, tels que les ordonnanceurs et les protocoles de communication.

4.3.1 Éléments contractuels

Le MP de cette facette fait office de contrat entre la facette et le MOE. Par conséquent les éléments contractuels suivants doivent y figurer :
- la granularité minimum : chaque objet de type `CR` sera implémenté de telle sorte qu'il puisse être vu par les autres facettes comme une ressource de calcul indépendante ;
- les échanges de données : les objets `Device` manipulent exactement les données (`HWData`) spécifiées dans le MP ;
- les attributs des objets : tous les attributs spécifiés dans le méta-modèle devront décrire la plate-forme concrète, éventuellement sous forme d'encadrement si une tolérance est acceptée :
  - taille : la taille de l'objet en octets ;
  - coût : le coût financier du développement de l'objet ;
  - `RAM` quantité de RAM disponible ;
  - `ROM` quantité de ROM disponible ;
  - `tauxDePannes` nombre de pannes maximum par heure d'utilisation
  - `toleranceAuxPannes` nombre de pannes supportées sans perdre la fonctionnalité de l'objet ;
  - `Puissance` puissance de la ressource de calcul en instructions élémentaires par seconde. La définition des "instructions élémentaires" doit être compatible avec celle utilisée par le MP de la facette logiciel (le mécanisme de passage de l'une à l'autre doit être connu) ;
  - `Debit` débit nominal d'un lien en bits par seconde.

4.3.2 Méta-modèle du MP de la facette matériel

La figure 4.10 page 61 décrit le méta-modèle du Modèle Public de la facette matériel. Celui-ci est organisé de la façon suivante : les ressources de calcul, `CR`, sont liées entre elles et à divers mécanismes, `Device`, (ici des capteurs `Sensor` ou actionneurs `Actuator`) par des liens physiques `H_Link`. En outre les ressources de calcul possèdent une politique d'ordonnancement `SchedPolicy`, bien que la modélisation précise de celle-ci ne soit pas clairement définie. En effet, le formalisme utilisé pour décrire les politiques d'ordonnancement dépend des analyses qui en seront faites par les layers et des méthodes qu'elles impliquent. Bien souvent, les méthodes d'analyse, par exemple des temps de réponse, se font à partir d'une description informelle de la politique d'ordonnancement. De nombreuses politiques d'ordonnancement sont décrites dans la littérature (EDF, Rate Monotonic, Round Robin, ...) (voir [CG03]) ainsi que leurs variantes. Dans le cas d'une de ces politiques "connues" il suffit souvent de la nommer (en ajoutant une référence à un ouvrage) la décrire.
Dans le cadre de cette thèse nous considérons que les ressources de calcul utilisent un ordonnancement statique (hors ligne) à tranche de temps fixe sans priorité (modélisée par la classe `TimeSlicing` dont le seul attribut est la tranche de temps `tt`). La figure 4.9 page suivante illustre le séquencement de cette politique sur trois processus (P1,P2 et P3) :
- la tranche de temps *tt* vaut 5ms ;

– la période de P1 est de 15ms et son temps d'exécution de 3ms ;
– la période de P2 est de 30ms et son temps d'exécution de 8ms ;
– la période de P3 est de 30ms et son temps d'exécution de 8ms ;
Chaque processus s'exécute durant les tranches de temps qui lui sont allouées. La dernière tranche d'une période n'est que partiellement utilisée du fait que le temps d'exécution des processus n'est pas un multiple de la tranche de temps

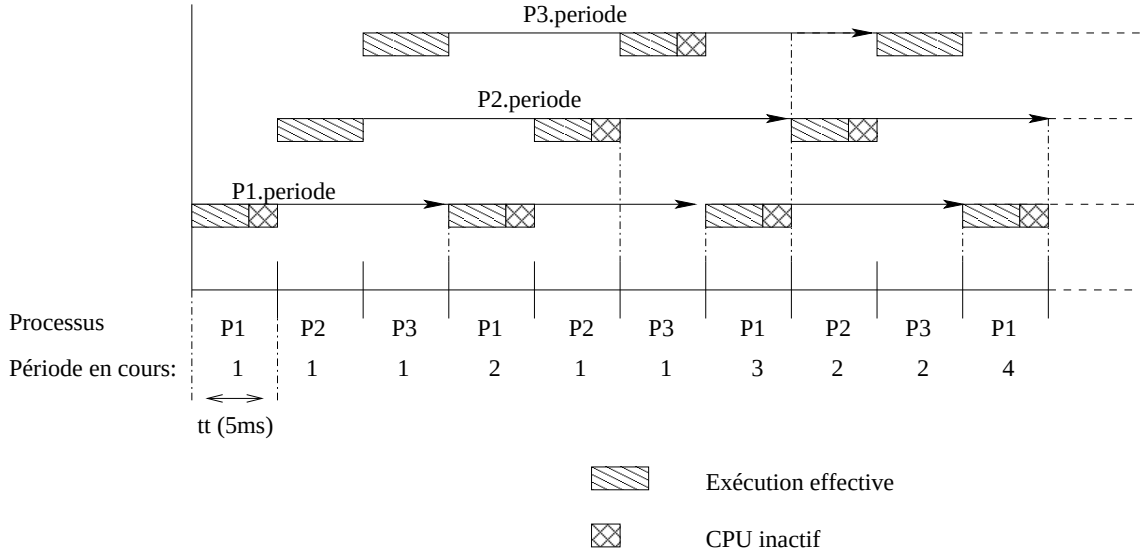


FIG. 4.9 – TimeSlicing

A l'image des ressources de calcul, les liens physiques peuvent comporter des informations de type "comportemental" comme le protocole de communication, la gestion des priorités, etc. Dans le cadre de notre étude nous utiliserons des bus CAN dont le protocole est normalisé (cf [CAN]).
Les objets `Device` émettent ou reçoivent des données modélisées par la classe `HWData`. Cette classe implémente l'interface `LogicalElement` car elle représente une entité logique (une donnée), c'est à dire sans masse ni volume. A l'inverse, `CR`, `Sensor` et `H_Link` implémentent `PhysicalElement` et sont donc dotés de caractéristiques de masse et de volume nécessaires à l'éventuelle intégration d'une facette "mécanique" et des layers associés, ce que n'étudierons pas dans cette thèse.

4.3.3 Facette matériel du flotteur

La figure 4.11 page 62 représente la facette matériel du flotteur. Elle comporte les éléments suivants :
– `CR1` et `CR2` sont les deux ressources de calcul redondantes, bien que différentes ;
– `CAN1` et `CAN2` sont deux bus de communication numériques redondants assurant la majorité du trafic du flotteur ;
– `A1` et `A2` sont deux liens de communication analogiques non redondants liant les capteurs/actionneurs aux bus numériques ;
– `CTD` est le capteur salinité/pression, il émet les données `SaliniteBrute` et `PressionBrute` ;
– `ModemIridium` est l'émetteur Iridum, il est considéré comme un actionneur (`Actuator`)
– `Piston` représente l'actionneur principal du système ;

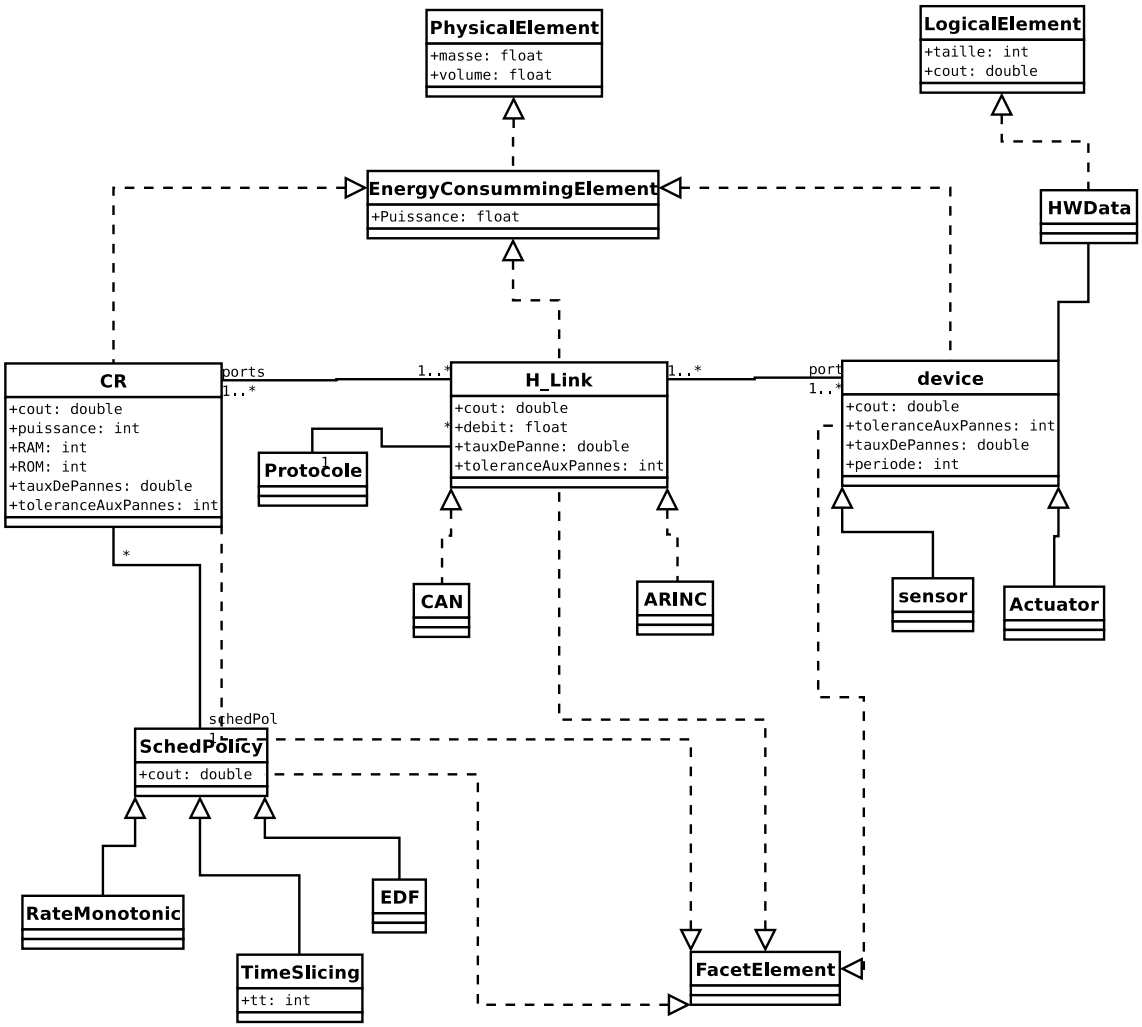


FIG. 4.10 – Meta-modèle du MP de la facette matériel

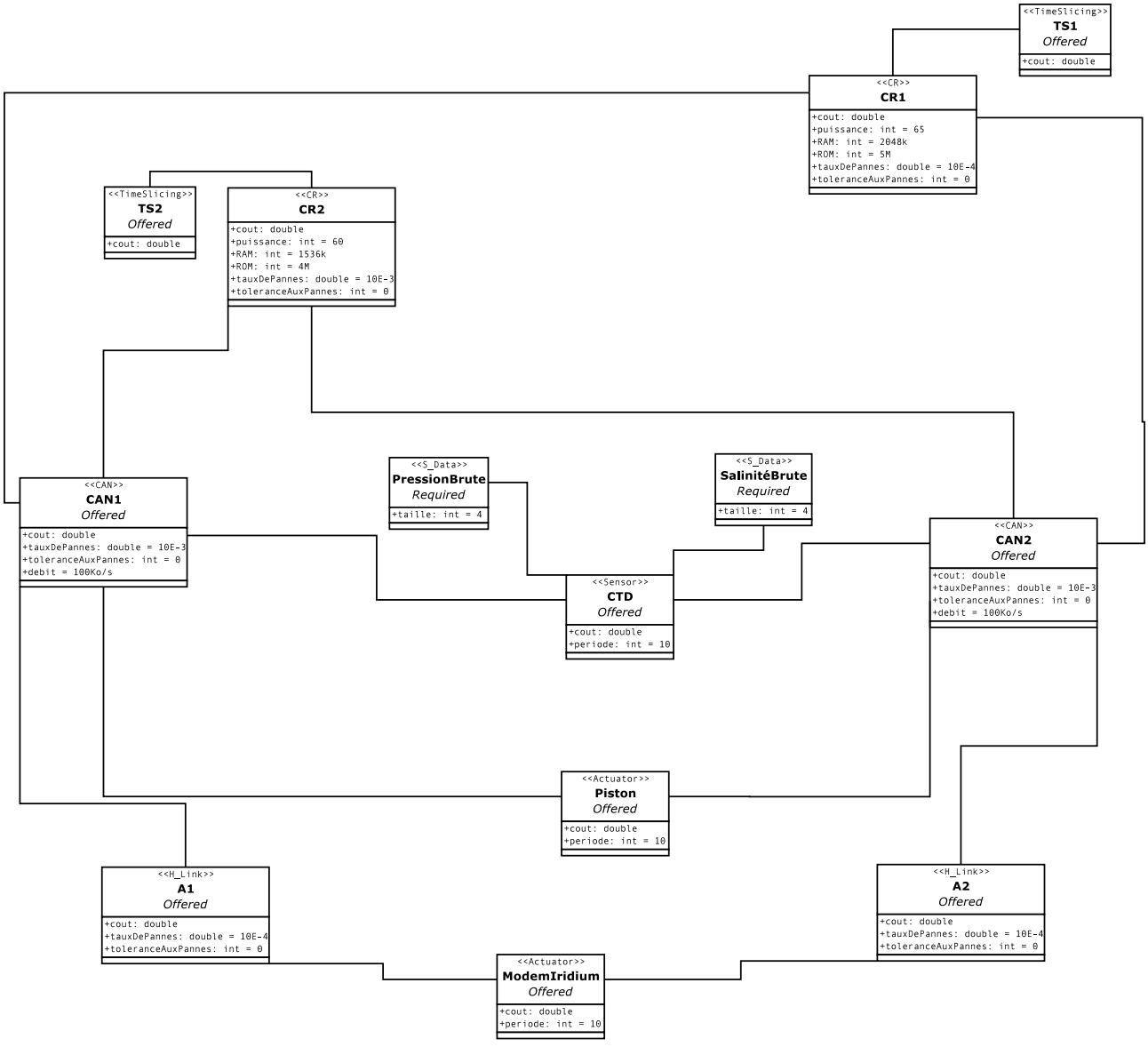


FIG. 4.11 – Facette matériel - Modèle Public, de l'étude de cas

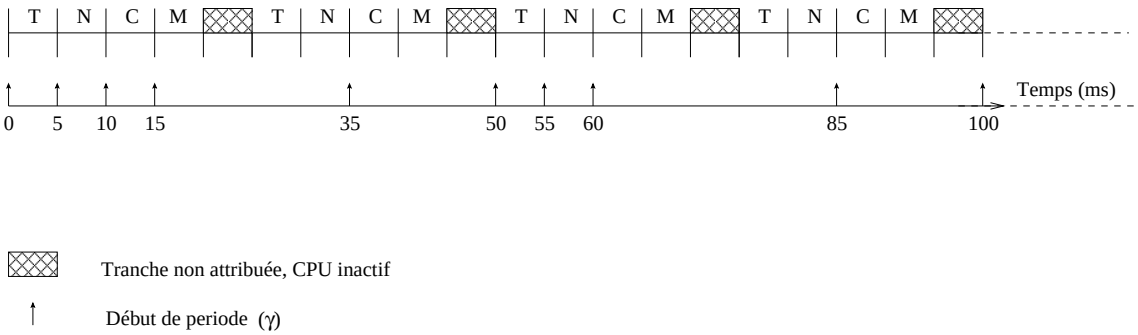


FIG. 4.12 – Facette matériel - Ordonnancement

La figure 4.12 fournit l'ordonnancement de la ressource `CR1` sur une hyper période. Les lettres T,N,C et M désignent respectivement les fonctions `BlockTraitement`, `BlockNavigation`, `BlockCommande` et `BlockMission`. La tranche de temps choisie est de 5ms. Cet ordonnancement laisse une marge CPU de 20% ce qui est généralement demandé pour une première version de système. Nous considérerons que la ressources `CR2` est équipée d'un ordonnanceur similaire.

4.3.4 Autres facettes

La réalisation d'un système embarqué peut faire appel à d'autres types de facettes qui bien que moins liées aux problématiques informatiques n'en demeurent pas moins importantes dans la cohérence globale du système. Parmi celles-ci, il en est une dont les interactions avec la facette matériel sont importantes : la facette mécanique. En effet, cette facette doit, entre autres préoccupations, faire face à des problèmes d'encombrement et de masse. Or les éléments de la facette matériel sont des objets physiques et à ce titre possèdent une masse et un volume. Bien que nous n'ayons pas détaillé cette facette, nous avons prévu dans nos modèles des points d'ancrage pour celle-ci en introduisant les classes abstraites `PhysicalElement` et `LogicalElement` dont héritent respectivement les objets matériels tels que les calculateurs, bus, etc. et les objets immatériels tels que les processus logiciel ou les données. Ces classes abstraites permettent la mise en place de layers calculant des informations de masse ou d'encombrement. Ces derniers paraissent ardus à définir, du fait de la complexité de l'expression de telles contraintes "topologiques".

Chapitre 5

Pivot : point de vue du maître d'œuvre

"Leadership is a matter of having people look at you and gain confidence, seeing how you react. If you're in control, they're in control."
Tom Landry

5.1 Généralités

Lors du développement multi-intervenants d'un système, si chaque spécialiste possède une connaissance totale de sa partie et une connaissance minimale des autres, le MOE, lui, a une vision transverse du système. Son rôle n'est pas de connaître les détails d'implantation de tel ou tel élément de son système, mais de veiller à ce que ses équipes métier (facettes) respectent individuellement leur cahier des charges et que leur intégration globale respecte les exigences initiales. Pour réaliser cette tâche, il lui faut un modèle analytique du système juste suffisamment détaillé pour que toutes les exigences puissent être exprimées sur ses entités. Nous appelons ce modèle "le pivot".
Conceptuellement, le pivot est à rapprocher du modèle primaire de la modélisation orientée aspects dans la mesure où il est organisé en blocs fonctionnels. En revanche, contrairement au modèle primaire, le raffinement du pivot est borné, il reste très abstrait, les détails étant gérés par les parties privées des facettes.
Cette borne du niveau d'abstraction des objets décrits par le pivot définit la granularité des objets fournis par les MP des facettes, nécessaire à l'intégration et à la collaboration entre les facettes.

Le choix de cette granularité est associé à une négociation entre les intervenants dans les phases préliminaires du développement (phase d'initialisation, cf. 7.1 page 103).

Exemple 4 *A chaque objet* `Fonction` *du pivot correspond exactement un objet* `Process` *du MP logiciel (cf. 5.2 et 6.3 page 78). Ceci implique que chaque* `Process` *quelque soit son implémentation finale devra pouvoir être exécuté de façon distribuée comme une tâche indépendante sur un calculateur éventuellement différent de celui exécutant un autre* `Process`.

5.2 Description architecturale du pivot

Le pivot est défini par un méta-modèle. La figure 5.1 page suivante présente un tel méta-modèle adapté à notre domaine d'étude. En toute généralité, le méta-modèle du pivot est dépendant du système considéré, et plus particulièrement, des facettes impliquées dans sa réalisation et de l'ensemble des exigences ; toutes les exigences du cahier des charges global doivent pouvoir être exprimées en termes d'objets du pivot. Le méta-modèle que nous fournissons est spécifiquement dédié aux systèmes temps réel embarqués à logiciel prépondérant. En effet il fournit des méta-classes permettant de décrire des fonctions logicielles intégrées sur une plate-forme matérielle et les liens de communication qui les relient. En outre, il a vocation à illustrer la construction d'un tel méta-modèle, pouvant être plus complexe lors de la modélisation spécifique d'un cas réel.
Le méta-modèle décrit par la figure 5.1 page suivante est composé des éléments détaillés ci-après, lesquels implémentent tous l'interface `PivotElement`, dont le but est de fournir une référence unique aux éléments constitutifs du pivot, utilisable dans d'autres méta-modèles (particulièrement ceux des layers, cf. figure 6.1 page 76).

– `Fonction` : cette classe décrit un "bloc fonctionnel" du système. Dans notre domaine d'étude, chacun de ces blocs sera réalisé par un processus logiciel associé à une partie des ressources de calcul chargée de les exécuter. Par "une partie" nous exprimons le fait que les ressources de calcul étant multi-tâches, seule une fraction de celle-ci est consacrée à exécuter un processus particulier ;
– `LienPivot` : cette classe représente un lien de communication abstrait entre deux fonctions.
– `P_Data` : cette classe décrit les données échangées par les fonctions du pivot. Elle est réalisée, soit par des données de la facette logiciel, soient par celles de la facette matériel (dans le cas ou la fonction représente un capteur, par exemple) ;
– `LatenceCom` : cette classe décrit la latence de communication d'une donnée sur un lien.

Éléments contractuels

Le méta-modèle présenté à la figure 5.1 page suivante indique les éléments contractuels à partir desquels la conformité du système pourra être établie en regard des exigences de haut niveau. Ces éléments sont les attributs des objets ainsi que les fonctions, liens, et données rendus nécessaires par la granularité fixée au démarrage du projet (cf. 5.1 page précédente), qui doit bien entendu être suffisante pour formuler toutes les exigences du cahier des charges en termes d'objets du pivot.

5.3 Sémantique du pivot

Afin de pouvoir exprimer, et vérifier des propriétés transverses d'ordre "performances temps réel", il

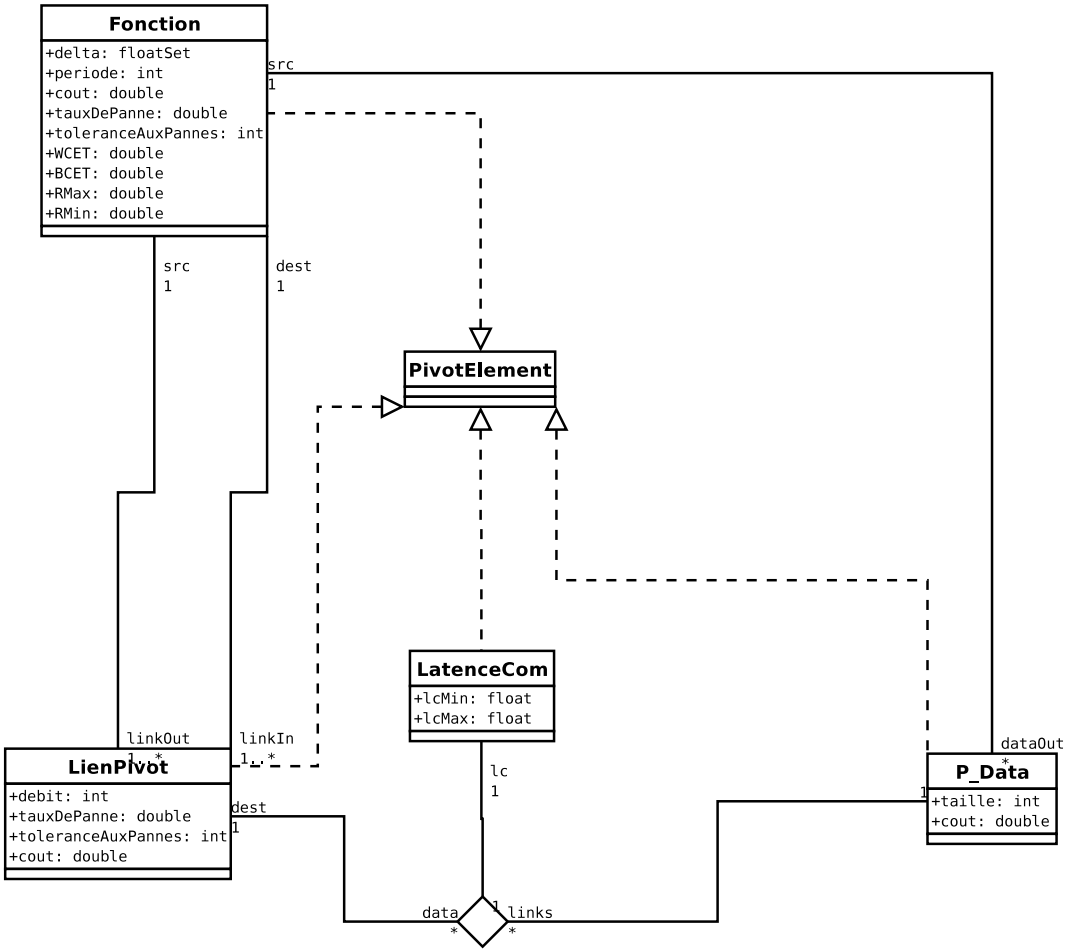


FIG. 5.1 – Méta-modèle du pivot

est nécessaire de préciser la sémantique des fonctions et des liens de communication. Pour ce faire, nous définissons d'abord une syntaxe abstraite du pivot exprimée sous forme d'une grammaire de type BNF équivalente au méta-modèle de la figure 5.1 page précédente, puis nous associons à cette syntaxe une sémantique opérationnelle exprimée par un ensemble de règles d'inférence.

Remarque 4 *Certains attributs du méta-modèle du pivot ne sont pas utilisés par la sémantique. Cela s'explique par le fait que : soit ils ne concernent pas les aspects comportementaux et/ou temps réel (tel que le coût par exemple), soit ils sont simplement des réceptacles de résultats intermédiaires manipulés par des layers (par exemple le WCET n'est pas utilisé dans la sémantique mais est nécessaire au calcul du temps de réponse qui lui, entre dans la sémantique).*

Remarque 5 *Le choix de passer par une syntaxe abstraite de type BNF se justifie par un allègement significatif de l'écriture des règles d'inférence. En effet, nous aurions pu écrire celles-ci uniquement à partir du méta-modèle, mais les noms des objets auraient dû être associés à des informations de navigation dans le méta-modèle.*

5.3.1 Syntaxe abstraite

Le pivot $\mathcal{P}$ est défini comme l'exécution parallèle de l'ensemble des fonctions et de l'ensemble des liens.

$$\mathcal{P} \quad := \quad \mathcal{F} \quad \|_{\mathcal{P}} \quad \mathcal{C}$$

Un ensemble de fonctions $\mathcal{F}$ est défini comme une fonction F ou un ensemble de fonctions exécuté en parallèle ($\|_F$) avec une fonction.

$$\mathcal{F} \quad := \quad (\mathcal{F} \quad \|_F \quad F) \,|\, F$$

Un ensemble de liens $\mathcal{C}$ est défini comme une fonction C ou un ensemble de liens exécuté en parallèle ($\|_C$) avec un lien.

$$\mathcal{C} \quad := \quad \mathcal{C} \quad \|_C \quad C \,|\, C$$

Une fonction est un n-uplet $< L, O, Rmin, Rmax, \Delta, A, T >$ où :

- L : ensemble des liens entrants de la fonction. La fonction consomme tous les messages provenant de ces liens et uniquement ceux-ci. L'ensemble L d'une fonction F est construit en récupérant dans le modèle l'ensemble des `F.linkIn`.
- O : ensemble des sorties de la fonction. L'ensemble O d'un fonction F est construit en récupérant dans le modèle l'ensemble des F.dataOut.
- *Rmin* et *Rmax* : le temps de réponse minimum et le temps de réponse maximum. Ce sont des attributs du modèle de la fonction.
- $\Delta \subseteq 2^{\mathbb{Q}^+}$ encadrement du décalage d'horloge de la fonction δ, fixé à chaque exécution du système. Il s'agit d'un attribut du modèle de la fonction.

– A : automate non-temporisé donnant le comportement abstrait de la fonction. Cet automate est fourni par la facette logiciel. Il n'apparaît pas dans le modèle du pivot mais il est connu du pivot par l'intermédiaire du Layer Mapping (cf. 6.3 page 78).
– T : période de la fonction.

$$F \quad := \quad < L, O, Rmin, Rmax, \Delta, A, T >$$

Un lien est un couple $< I, lc >$ où :
– I désigne l'ensemble des données transitant par ce lien. I se retrouve dans le méta-modèle du pivot dans le rôle `data` associé à `LienPivot` ;
– lc désigne la fonction donnant le temps de latence de chaque message. Elle renvoie l'intervalle $[lcMin, lcMax]$ attribut de la classe `LatenceCom` en association ternaire avec un lien et une donnée.

$$C \quad := \quad < I, B, lc >$$

5.3.2 Sémantique

Lors de sa première activation une fonction subit un retard d'activation créant un décalage d'horloge fixe et constant δ.
Une fonction acquière ses entrées au début de chaque période. Elle passe ensuite dans une phase de calcul C, et enfin, émet ses sorties dans l'intervalle O.
La figure 5.2 page suivante illustre une période d'une fonction. Elle se découpe en quatre phases :
– La première phase se déroule en temps nul. Elle consiste en l'acquisition des données d'entrée. Ainsi, durant toute la période, la fonction travaille sur le contexte de l'environnement à l'instant t_0. La cohérence du contexte logiciel est ainsi assurée pour la période ;
– La phase suivante (C sur la figure) est la phase durant laquelle la fonction effectue les calculs nécessaires à la production de ses sorties. Durant cette phase, la fonction ne communique pas avec l'environnent ;
– La phase dénotée O! qui à l'instar de la première phase est instantanée consiste en l'émission des sorties vers l'environnement (liens). Le moment où se produit cette phase se situe dans l'intervalle O délimité par les temps de réponse minimum et maximum.
– La dernière phase correspond au temps de CPU non utilisé par la fonction avant la nouvelle période.

Remarque 6 *En pratique, durant la phase de calcul, le logiciel émet ses sorties au fur et à mesure de ses calculs vers des buffers logiciels qui sont vidés simultanément une fois par période. Ce qui explique que les automates de la facette logiciel peuvent effectuer des émissions à n'importe quel moment.*

La sémantique opérationnelle formelle du pivot est disponible à l'annexe B.1 page 127

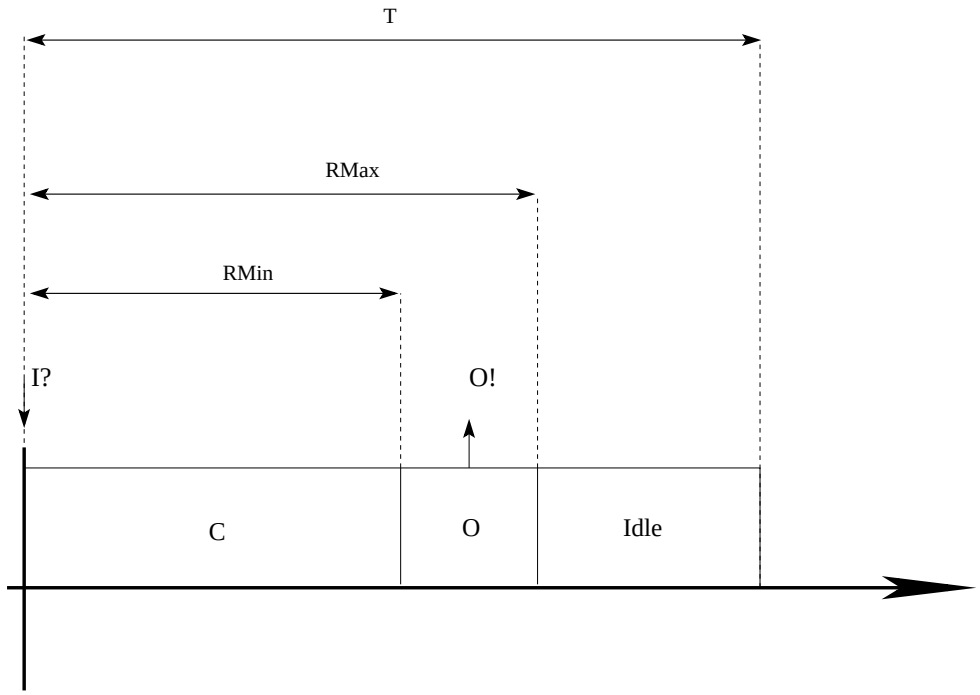


FIG. 5.2 – Déroulement d'une période

5.4 Pivot de l'étude de cas

La figure 5.4 page suivante fournit une vision informelle du pivot de l'étude de cas. Il s'agit d'un modèle purement documentaire n'ayant aucune valeur contractuelle. Elle permet aux intervenants de préciser leur vision du pivot en synthétisant les informations nombreuses fournies par le modèle formel du pivot. L'interprétation de ce diagramme est illustrée sur l'exemple suivant.

Exemple 5 *La figure 5.3 se lit de la façon suivante : Il s'agit d'un pivot constitué de deux fonctions* `F1` *et* `F2`*, du lien* `lien1` *et de la donnée* `donnée`*. La flèche entre les deux blocs représentant les fonctions indique que* `F1` *transmet* `donnée` *à* `F2` *via* `lien1`*.*

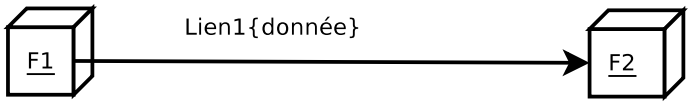


FIG. 5.3 – Exemple de diagramme synoptique

La figure 5.5 page 73 montre une partie du pivot de l'étude de cas. Il s'agit des fonctions "maîtres", des canaux qui les relient et les données qu'elles échangent. Le reste du pivot se trouve à l'annexe B.2 page 130. Les fonctions "maîtres" sont les suivantes :

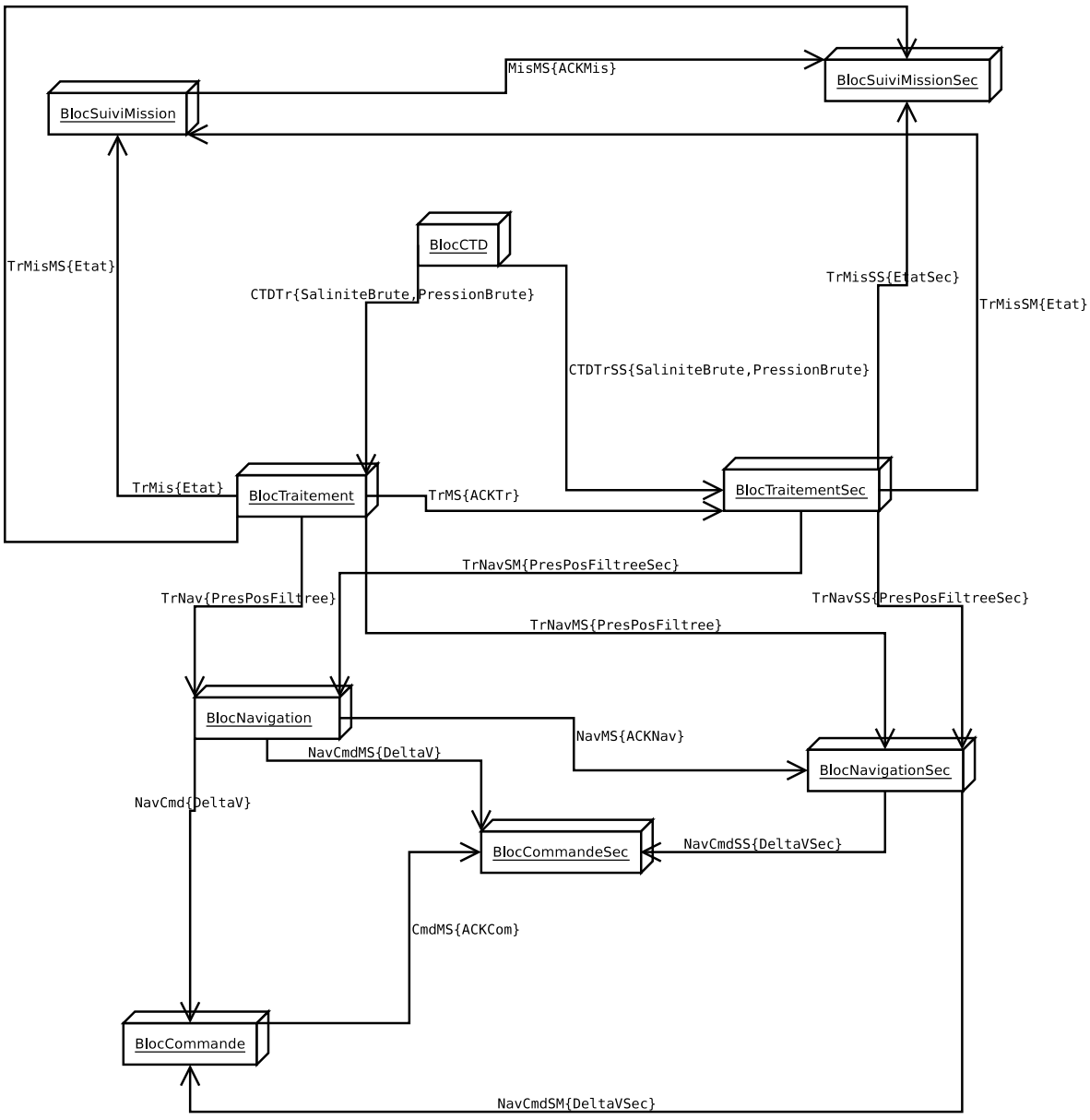


FIG. 5.4 – Diagramme synoptique informel du pivot de l'étude de cas

– `BlocCTD` : décrit le capteur salinité/pression ;
– `BlocTraitement` : décrit la fonction de traitement des données capteur ;
– `BlocSuiviMission` : décrit la fonction de suivi de mission ;
– `BlocNavigation` : décrit la fonction de navigation fournissant l'immerssion de consigne ;
– `BlocCommande` : décrit la fonction calculant la commande à destination des actionneurs pour atteindre la profondeur demandée par la navigation.

Ces fonctions sont réalisées par un processus logiciel exécuté sur une ressource de calcul partagée. Ce point est détaillé dans la suite du document. 6.3 page 78

Ces fonctions communiquent via les liens suivants :
– `CTDTr` : ce lien relie le capteur à la fonction de traitement des données brutes ;
– `TrMisMM` : ce lien relie la fonction de traitement des données brutes au suivi de mission ;
– `TrNavMM` : ce lien relie la fonction de traitement des données brutes à la navigation ;
– `NavCmd` : ce lien relie la fonction de navigation à la fonction de commande ;

Les données échangées sont les suivantes :
– `PressionBrute` ;
– `SaliniteBrute` ;
– `PressPosFiltreeMM` ;
– `EtatMM` ;
– `DeltaVMM`.

Les attributs des objets du pivot sont calculés et utilisés par des layers. Ces layers ont pour but d'une part de produire des informations transverses à destination du pivot, et d'autre part d'exprimer les relations de conformité existant entre les différentes facettes et les exigences système. Le chapitre suivant est consacré aux layers.

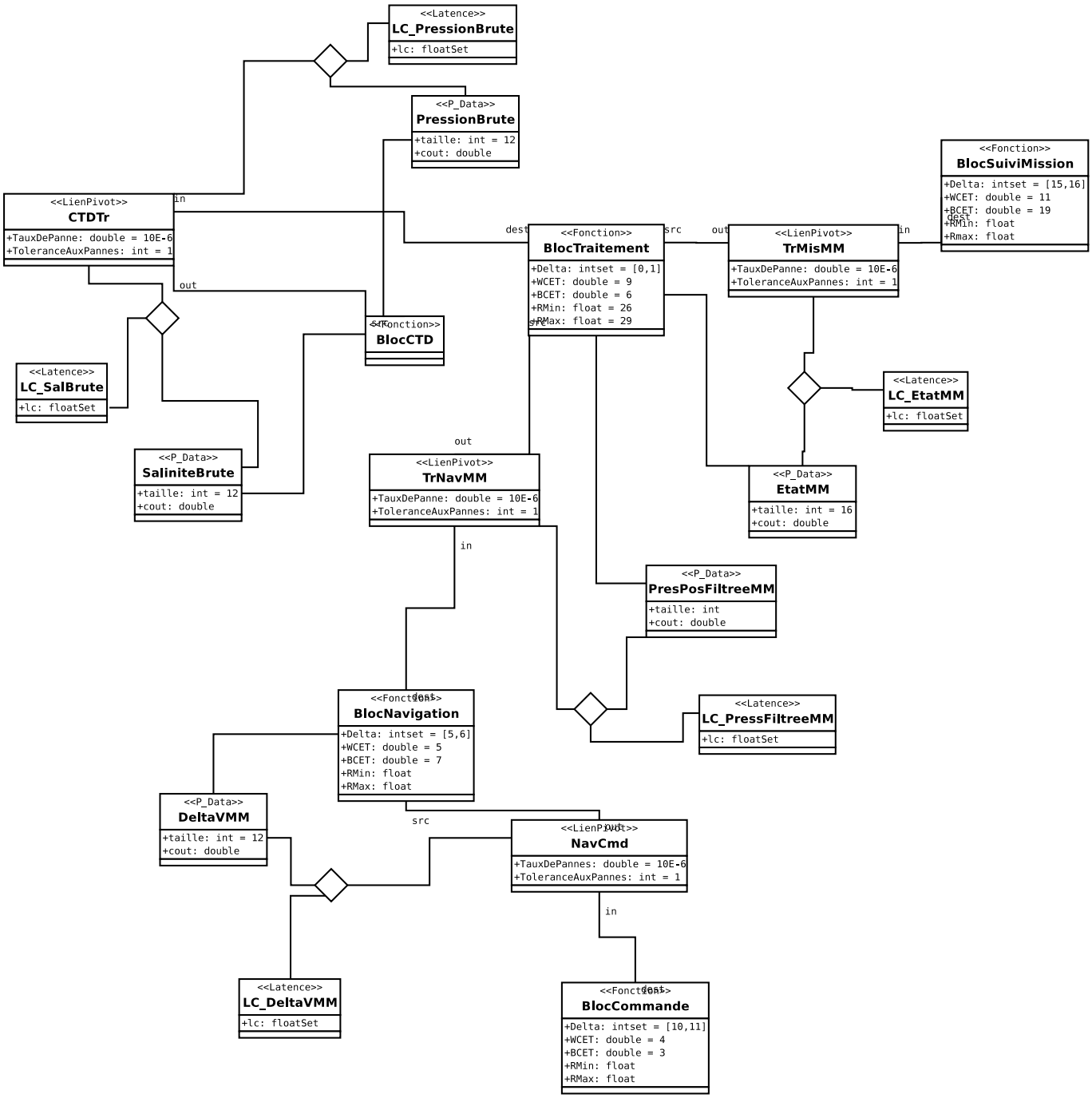


FIG. 5.5 – Pivot de l'étude de cas Maître vers Maître

Chapitre 6

Layers

"Leadership has a harder job to do than just choose sides. It must bring sides together."
Jesse Jackson

6.1 Layers : généralités

Nous avons présenté les modèles des facettes métier puis celui du pivot en assurant que ce dernier fournit une vision transverse des premiers, en exhibant des propriétés du système obtenues à partir des MP des facettes. Il nous faut maintenant fournir les mécanismes permettant d'une part de construire ces informations et d'autre part, de garantir la cohérence inter-facettes et avec le pivot. Nous introduisons pour ce faire une famille de modèles nommés "layers". Le rôle est de relier les objets des MP des différentes facettes avec les objets du pivot et de qualifier les relations ainsi induites par des contraintes. Celles-ci permettent soit le calcul d'informations transverses, soit la vérification de la cohérence.

Exemple 6 (Layer comptable) *Lors d'un développement par facette métier, l'évaluation des coûts est faite par les intervenants, soit de façon globale pour toute leur partie, soit plus précisément pour chacun des objets qu'ils fournissent. Du point de vue du MOE, en revanche, il est plus intéressant de voir ces coûts associés à des fonctionnalités du système ou même aux exigences, afin d'évaluer l'influence de chaque besoin exprimé dans le coût global du système. Par exemple dans l'exemple du distributeur de billets (cf exemple 1 page 31), il est important de savoir combien coûte la fonctionnalité IHM ou la fonction authentification, surtout en cas de dépassement du budget par les devis des sous-traitants. Pour évaluer le coût de chaque fonction, il faut, d'une part connaître le coût de développement du processus logiciel qui la réalise, de la ressource de calcul qui l'exécute et le pourcentage de celle-ci que la fonction occupe. Un layer comptable sera associé à chaque fonction*

du pivot et au processus logiciel ainsi qu'à la ressource de calcul associée, et grâce aux informations de répartition de la charge sur les ressources fournies par d'autres layers, calculera le coût réel de chaque fonction et le stockera dans l'objet du pivot correspondant. Ceci permet au maître d'œuvre d'identifier précisément quelle fonctionnalité (et donc quelle exigence) crée le dépassement de budget (une IHM trop riche par exemple).

La figure 6.1 montre le méta-modèle des layers.

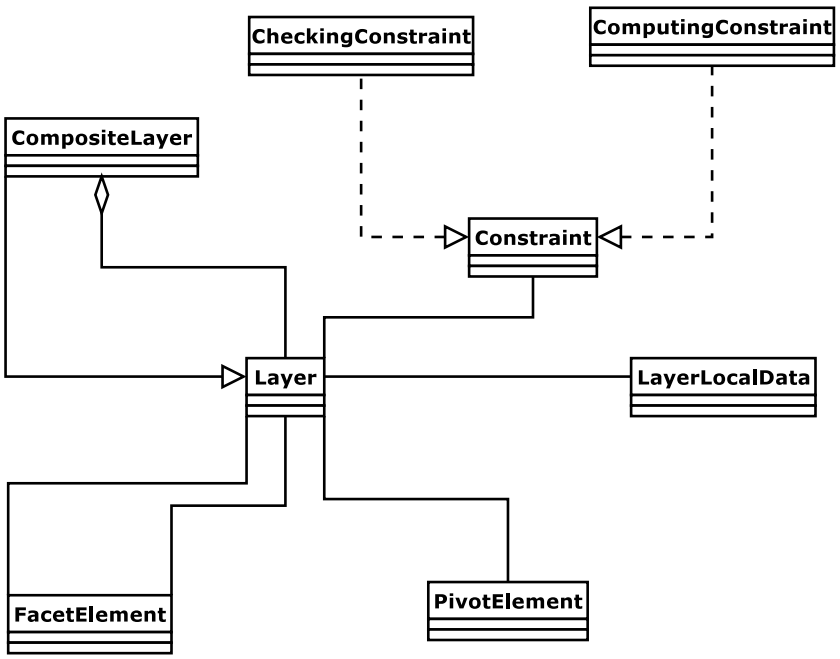


FIG. 6.1 – Méta-modèle des layer

Ce méta-modèle indique qu'un layer applique une ou plusieurs contraintes à au moins un objet offert par une facette, au moins un élément de pivot et parfois des éléments requis d'une facette. En effet un layer se sert obligatoirement d'un élément interne d'une facette car eux seuls fournissent de l'information, il s'applique aussi sur un élément de pivot dans la mesure où, soit il vérifie une exigence système et donc se réfère à un élément du pivot, soit il calcule une information et doit la stocker dans un élément de pivot, ainsi que dans les éléments demandeurs d'autres facettes concernées par les-dites informations. De plus une exigence transverse globale peut parfois être liée à des exigences transverses locales. Pour exprimer ce fait, le méta-modèle des layers comporte une classe `CompositeLayer` qui permet de lier des exigences locales au sein d'un même layer. Ce mécanisme est à rapprocher du pattern visiteur, dans la mesure où l'appel à un layer composite revient à appeler successivement les layers qui le composent.
Les contraintes seront exprimées dans un langage exécutable en tant que méthodes de la classe `Constraint`. Ces méthodes s'appliqueront sur les objets liés au layer de rôle `informative` et stockeront leur éventuel résultat dans les objets de rôle `storage` (ceux-ci seront soit des objets du pivot soit des objets requis de facettes). Il apparaît deux types de contraintes :

– les contraintes de validation (`CheckingConstraint`) ont pour but d'exprimer une exigence qui doit être vérifiée. Ce sont des prédicats et leur évaluation n'entraîne aucune modification des différents objets du layer associé à la contrainte ;

– les contraintes de calcul (`ComputingConstraint`) ont en charge le calcul d'informations transverses telles que les attributs des objets du pivot, ou la création et le renseignement d'objets requis par les facettes (par exemple la création d'un processus logiciel virtuel à partir d'un capteur de la facette matériel. Ces contraintes modifient certains objets du layer associé.

Enfin, un layer possède des "données de travail" éventuellement liées à des éléments de facette ou du pivot. Ces données modélisées par la classe `LayerLocalData` servent de zone de stockage de résultats intermédiaires et à la transmission de ces informations entre un layer et ses sous-layers (`CompositeLayer`). Elles traduisent le fait que certains éléments de description d'un système ne sont pertinents que durant une partie du cycle de vie. Le paragraphe 6.5 page 85 du présent chapitre fournit un exemple d'utilisation de ces données locales.

6.2 Dépendance entre les layers

Les différents layers associés à un système, à l'image des exigences, ne sont généralement pas indépendants les uns des autres. Les dépendances entre les layers interviennent principalement dans un contexte dynamique de processus de développement puisqu'elles posent la question du "quand" ; quand doit-on appliquer tel où tel layer ? De quelles informations transverses la vérification de telle exigence dépend-t-elle ? ...

Bien que nous n'évoquons les processus dynamiques qu'au chapitre 7 page 103, il semble important pour la compréhension du présent chapitre de traiter la question de la dépendance dès à présent.

Il nous faut pour cela, évoquer les différentes relations de dépendance pouvant exister entre les layers d'un même système. Ces relations peuvent être classées en deux catégories, verticales ou horizontales.

6.2.1 Dépendances horizontales

Deux layers sont dits dépendants horizontalement s'ils doivent être appliqués simultanément. Autrement dit, il s'agit de layers locaux portant sur des objets particuliers, généralement de même type, qui contribuent à la réalisation d'une exigence globale. Cette relation est spécifiée par la classe `CompositeLayer` du méta-modèle évoqué précédemment.

Exemple 7 *Pour calculer la masse totale d'un système, il faut additionner les masses de tous les objets du pivot. Les instances du layer "masse" liées aux différents objets du pivot sont en relation de dépendance horizontale et sont regroupées par le layer composite "MasseGlobale"*

6.2.2 Dépendances verticales

Un layer A est dit dépendant verticalement du layer B si sa contrainte se sert explicitement de données calculées par A. Dans le cadre d'un processus de développement, A doit être appliqué avant B. Cette forme de dépendance est plus problématique que la dépendance horizontale dans la mesure où elle peut introduire des raisonnements circulaires. Ces problèmes peuvent être inhérents à la formulation des exigences ; Si les exigences ne sont pas contradictoires (le système est réalisable) alors il suffit de construire les layers, soit en laissant des valeurs symboliques pour les données

interdépendantes qui seront précisées au fur et à mesure des choix de conception effectués par les facettes, soit, le cas échéant, en remplaçant un ou plusieurs paramètres par une borne “dans le pire des cas” issue des exigences. Cette dernière méthode permet d’obtenir des exigences induites locales, brisant localement la circularité des exigences.

6.3 Le layer mapping

Ce layer joue un rôle particulier dans le modèle général. En effet, son rôle est d’établir des liens entre les objets des MPs des facettes via les éléments du pivot. Ce rôle particulier pose la question de savoir s’il s’agit vraiment d’un layer ou bien d’un autre type d’entité.
En dehors du fait qu’ajouter un nouveau concept à ce stade alourdirait notre propos (principe du rasoir d’Occam), il existe des arguments pour attribuer le statut de layer au mapping ; tout d’abord, structurellement le mapping est bien conforme au méta-modèle des layers (voir figure 6.1 page 76). Ensuite conceptuellement, le mapping, à l’instar des autres layers, enrichit le pivot à partir d’informations extraites des facettes. Le fait que dans le cas du mapping ces informations soient de nature structurelle plutôt que numérique ne contredit en rien la définition que nous donnons des layers. Nous considérerons donc dans la suite de ce document que le mapping est un layer à part entière, sans toute fois éluder ses spécificités. La plus importante d’entre-elles étant que de part sa nature même, le layer mapping est celui dont dépendent tous les autres. Il doit être créé en premier.

6.3.1 Méta-Modèle de mapping

La figure 6.3 page 80 présente le méta-modèle du layer mapping. Ce méta-modèle indique que la mapping est un layer composite. Il est l’association de trois types de mapping :
– `MappingFonction` : il relie une fonction du pivot à un processus de la facette logiciel et une ressource de calcul de la facette matériel. Réciproquement une fonction du pivot et un processus logiciel sont liés à un seul `MappingFonction` alors qu’une ressource de calcul peut être associée à plusieurs `MappingFonction`. La raison en est que les ressources de calcul peuvent être multi-tâches ;
– `MappingLien` : Un lien pivot (`P_Link`) abstrait plusieurs routes (`Chemin`) en parallèle (introduction de redondance fonctionnelle pour des raisons de tolérance aux pannes). Un chemin est un objet du mapping, il est composé d’une succession de liens physiques (`HW_Link`) ;
– `MappingData` : il relie une donnée pivot (`P_Data`) à une donnée logiciel (`S_Data`) et une donnée physique (`HW_Data`).

6.3.2 Mapping de l’étude de cas

La figure 6.3 page 80 représente le mapping général de l’étude de cas. Celui-ci se contente de lister les divers mappings associés au système (leurs détails étant masqués) dont un exemple est détaillé ci-dessous, les autres étant renvoyés à l’annexe C.1 page 135.

La figure 6.4 page 81 est une extraction du mapping complet permettant d’obtenir la liste des fonctions exécutées par la ressource `CR1`. Pour obtenir cette information il suffit, lors de la navigation dans le modèle complet du mapping d’extraire les objets de type `MappingFonction` liés à cette ressource de calcul.

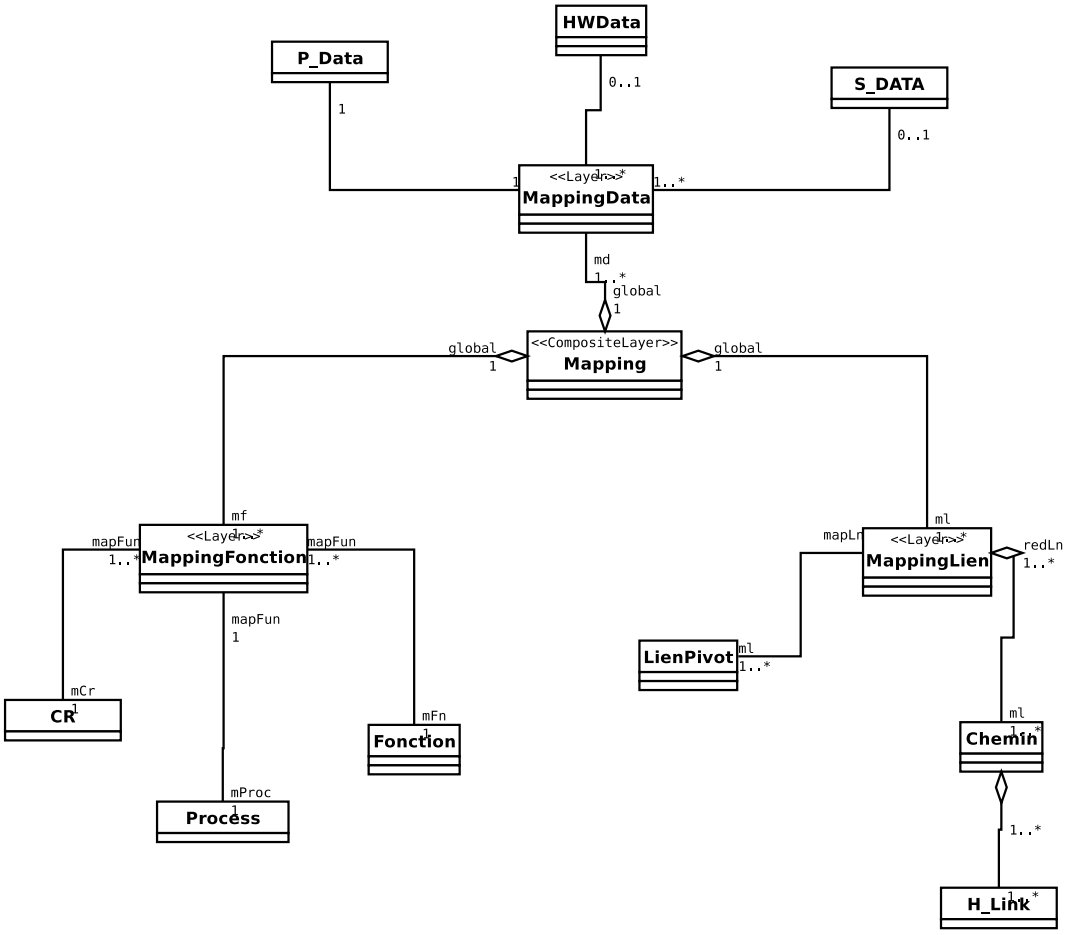


FIG. 6.2 – Méta-modèle du layer mapping

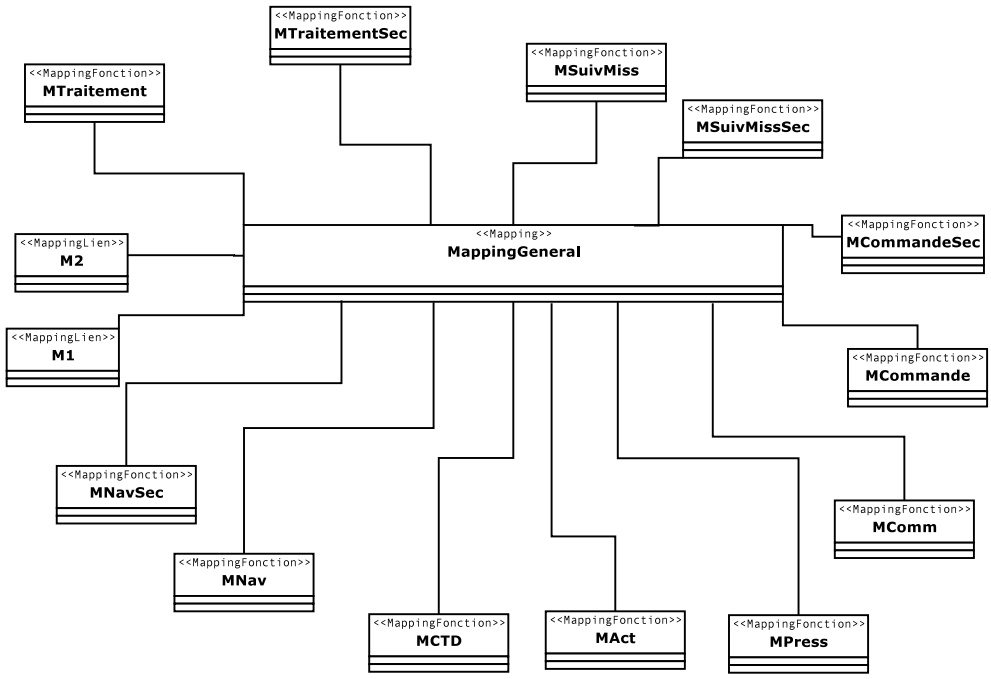


FIG. 6.3 – Mapping général

6.4 Performances temps réel

Ce layer (ou famille de layers) a pour buts, d'une part de calculer les données de type performances temps-réel du pivot et d'autre part de vérifier l'adéquation des-dites données avec les exigences système.
Nous allons décrire les layers calculant le temps de réponse des fonctions et les latences de communication sur les bus. Nous nous appuierons sur les résultats exposés dans [Ric03].

6.4.1 Temps de réponse

Les performances temps-réel d'une fonction sont décrites principalement par deux notions clefs ; Le temps d'exécution d'une part indique le temps que met un processus logiciel à calculer ses sorties s'il est seul sur sa ressource de calcul. Ce temps n'étant pas constant en fonction des entrées, il convient de distinguer le meilleur temps d'exécution (BCET, Best Case Execution Time) du pire, (WCET, Worst Case Execution Time). Diverses techniques existent pour évaluer le WCET et BCET que nous n'évoquerons pas, nous supposerons ces informations connues et rendues disponibles dans le pivot par un layer dédié. L'autre notion permettant de décrire les aspects temps-réel d'une fonction est le temps de réponse. Dans le cas d'un système faisant appel à des ressources de calcul multi-tâches (donc munies d'un ordonnanceur), le temps de réponse correspond au temps effectivement mis par un processus logiciel particulier pour calculer ses sorties en tenant compte des autres processus partageant sa ressource de calcul. En pratique c'est bien sur le temps de réponse que s'exprime les exigences temps-réel du système dans la mesure où c'est lui qui décrit le fonctionnement de la fonction *au sein du système* et non le temps d'exécution, qui exprime une propriété de la fonction

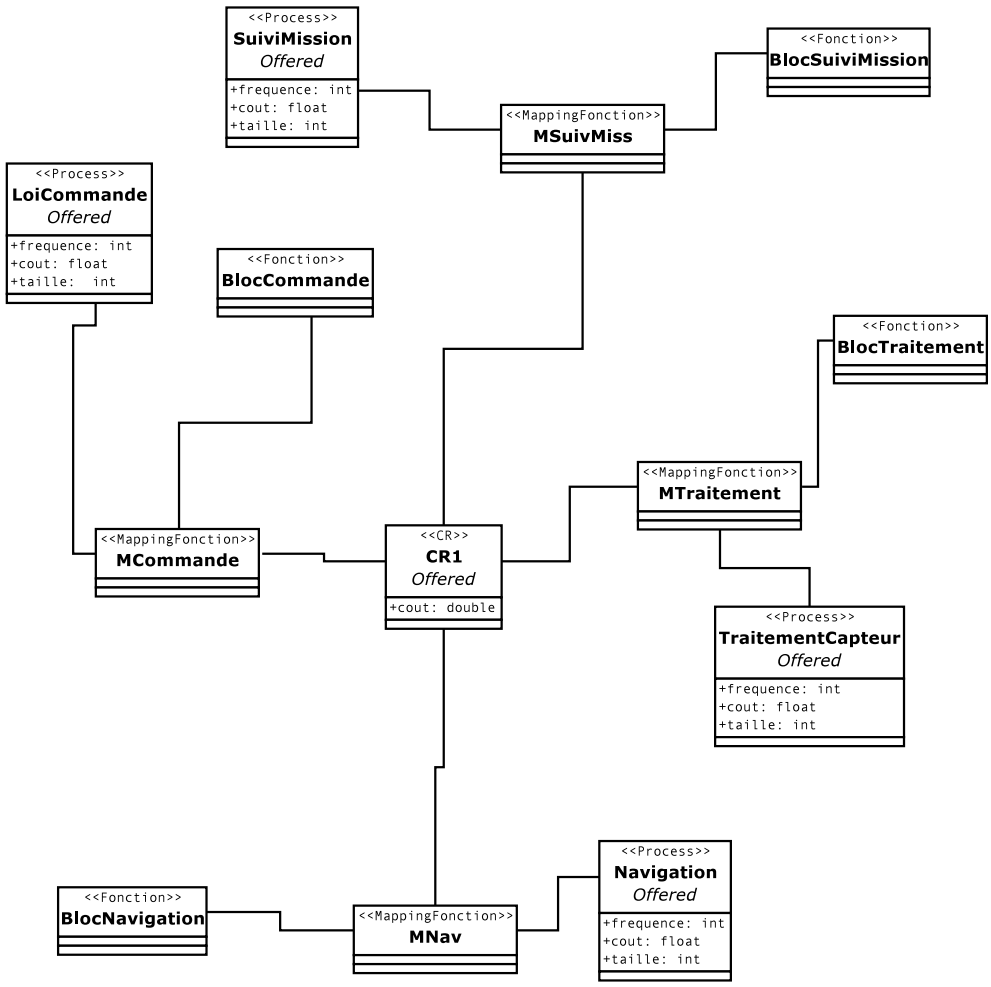


FIG. 6.4 – Mapping du calculateur principal

indépendamment de son environnement. Nous distinguerons comme dans le cas du temps d'exécution, un pire et un meilleur temps de réponse. Nous allons maintenant décrire le layer chargé de calculer ce temps de réponse. Notons tout d'abord que ce layer est dépendant du layer calculant le temps d'exécution (supposé connu ici). En effet le calcul du temps de réponse nécessite de connaître les temps d'exécution d'une part, et d'autre part de posséder une description suffisamment précise de l'ordonnancement en vigueur sur la ressource de calcul. Bien entendu, ce layer dépend du layer mapping puisqu'il faut savoir quelles fonctions se partagent la ressource de calcul étudiée. [Ric03] propose une méthode de calcul du temps d'exécution dans le cas d'un ordonnancement *en-ligne* de type rate-monotonic. Pour notre part nous nous limiterons à l'étude de l'ordonnanceur fourni par la facette matériel de notre étude de cas. S'agissant d'un ordonnancement *hors-ligne* (statique et pré-calculé) de type time-slicing à tranche de temps fixe, le calcul est grandement simplifié.
Les contraintes de ce layer utilisent la méthode $calculCR()$ définie comme suit :
En notant,
– $s \triangleq this.layer.process$ le processus logiciel du layer ;
– $cr \triangleq this.layer.cr$ la ressource de calcul impliquée ;
– $\#F \triangleq this.layer.cr.mapFun.size()$ le nombre de fonctions mappées sur la ressource.
– $F \triangleq this.layer.function$

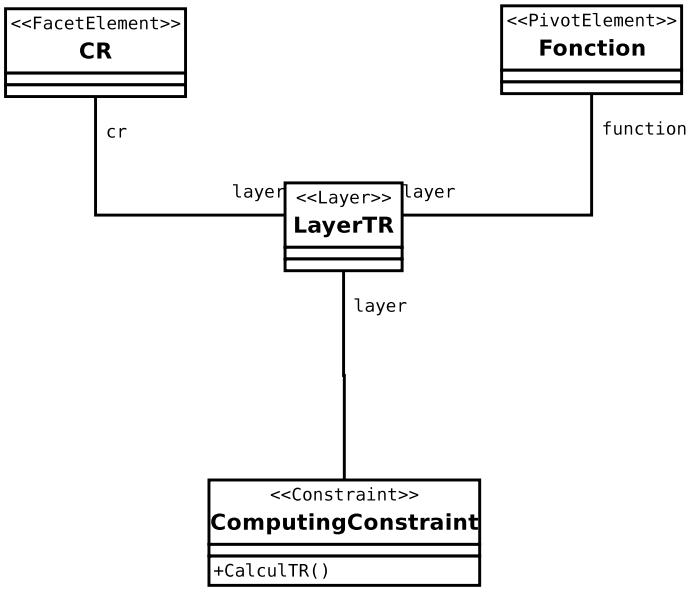


FIG. 6.5 – Méta-modèle du layer Temps de réponse

$$calculCR() \triangleq$$

$$F.R_{MAX} = \left\lfloor \frac{F.WCET}{cr.schedPol.tt} \right\rfloor \times (cr.schedPol.HP - cr.schedPol.tt) + F.WCET$$

$$F.R_{MIN} = \left\lfloor \frac{F.BCET}{cr.schedPol.tt} \right\rfloor \times (cr.schedPol.HP - cr.schedPol.tt) + F.BCET$$

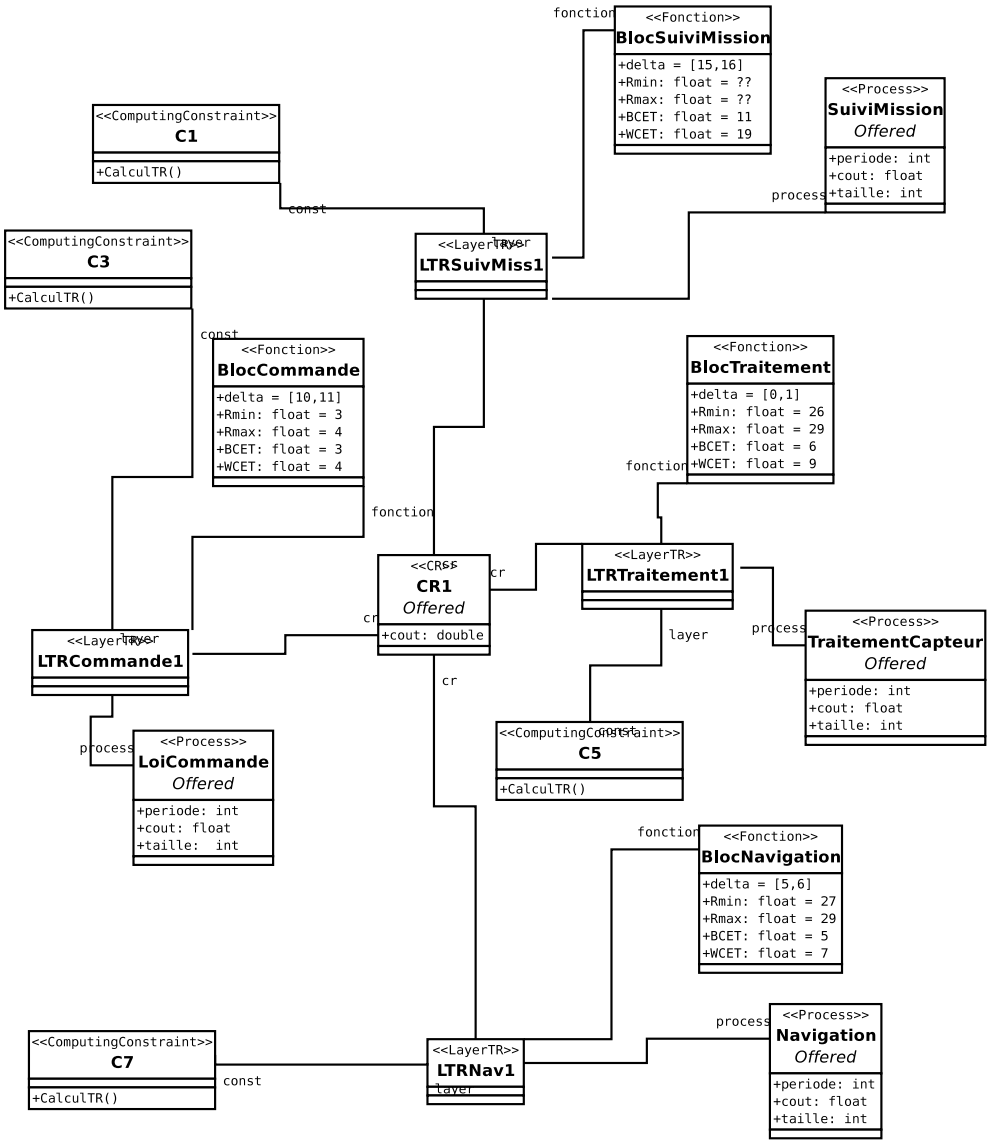


FIG. 6.6 – Layer Temps de réponse

Les résultats des calculs du layer `LayerTR` viennent renseigner le pivot et sont présentés dans le tableau 6.7 :

Fonction	Inf(delta)	Sup(delta)	BCET	WCET	R_{MIN}	R_{MAX}
BlocTraitement	0	1	6	9	26	29
BlocNavigation	5	6	5	7	25	27
BlocCommande	10	11	3	4	3	4
BlocMission	15	16	11	19	51	79
BlocTraitementSec	20	21	6	9	26	29
BlocNavigationSec	25	26	5	7	25	27
BlocCommandeSec	30	31	3	4	3	4
BlocMissionSec	35	36	11	19	51	79

FIG. 6.7 – Résultats du layer LayerTR

Remarque 7 *Les gigues sur activation (delta) des fonctions de secours s'expliquent par le fait que le calculateur de secours démarre 20ms après le calculateur principal. Il s'agit d'un choix de conception.*

6.4.2 Latences de communication

La communication de données par un bus de communication partagé entraîne des délais de transmission dûs au partage de la ressource, en général plus importants que le simple temps de transit de chaque donnée. [Ric03] fournit une méthode de calcul de ces latences dans le cas d'un bus CAN. Les priorités des données sur le bus sont supposées affectées par un layer que nous ne détaillerons pas. Le calcul s'effectue de la façon suivante (les détails de la méthode sont fournis dans [Ric03]), en considérant les n données classées par priorité décroissante :
Soit d_i la donnée de priorité i, notons
- $C_i = (\lfloor \frac{34+8b}{5} \rfloor + 47 + 8b)\tau_{bit}$ le temps de traversée de d_i où b_i est la taille de d_i en bits et τ_{bit} le débit du bus en octets par seconde ;
- J_i la gigue sur activation de la donnée d_i (décalage entre l'échéance de sa période et sa disponibilité pour le bus) ;
- T_i la période d'émission de d_i
- $W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{t+J_j}{T_j} \right\rfloor + \max_{j=i+1,\ldots,n}(C_j)$ la fonction de travail de d_i. Le dernier terme modélise le fait que si une donnée moins prioritaire est déjà en transit sur le bus au moment où d_i en demande l'accès, celle-ci ne peut être interrompue et que toutes les tâches plus prioritaires serons retardées du temps de transit de la donnée en cours de transmission.

La latence maximum de communication de d_i est le plus petit point fixe de la fonction de travail $W_i(t)$.
La latence minimum se produit lorsque d_i est seul à demander l'accès au bus, elle est donc égale au temps de transit de d_i, c'est à dire C_i.

Latences de l'étude de cas

Le tableau 6.8 fournit les résultats du calcul des latences de communication des liens du pivot de l'étude de cas.

Donnée	Priorité	Gigue (J_i)	Taille (b_i)	Periode (T_i)	Transit(C_i)	LatenceMax
SaliniteBrute	1	0	12	10	0.664062	1.376953
PressionBrute	2	0	12	10	0.664062	2.041016
PresPosFiltreeMS	3	3	16	50	0.712891	2.753906
DVMS	4	2	12	50	0.664062	3.417969
EtatMS	5	3	16	50	0.712891	4.130859
AckTr	6	3	9	50	0.625000	4.755859
AckNav	7	2	9	50	0.625000	5.380859
AckComm	8	1	9	25	0.625000	6.005859
AckMis	9	28	9	100	0.625000	6.630859
PresPosFiltreeSM	10	3	16	50	0.712891	7.343750
DVSM	11	2	12	50	0.664062	8.007812
EtatSM	12	3	16	50	0.712891	8.720703

FIG. 6.8 – Résultat du calcul des latences de communication

6.5 Le layer vérification

Un problème majeur posé par la modélisation distribuée est la validation comportementale temps réel du système. En effet, s'il existe de nombreuses techniques formelles (model-checking, test, etc.) (cf. [Lar03]) permettant de modéliser et valider les logiciels temps réel, il est très ardu d'y intégrer les effets de la plate-forme matérielle. Cette prise en compte du hardware nécessite une grande part d'intervention humaine (cf. [JE03, Erm02]). Cet écueil est contourné en introduisant une temporisation des modèles des traitements (comme les automates temporisés cf. [AD94]), ce qui revient à construire un modèle unique contenant des informations fonctionnelles (comportementales) et des informations de l'ordre de l'exécutif. Même si ces méthodes sont prometteuses, elles sont en contradiction avec notre volonté de séparer les modèles par métier. Nous proposons un layer validation qui, à partir de données extraites des informations comportementales et exécutives provenant des facettes logiciel et matériel, construit un système d'automates temporisés communicants sur lequel s'appliquent les techniques de validation formelle évoquées précédemment. Ce layer est basé sur un algorithme de traduction vers le formalisme cible (automates temporisés d'UPPAAL par exemple).

6.5.1 Traduction du pivot en automates temporisés

Définition 1 (Automate temporisé) *Un automate temporisé est un n-uplet* $A =< \mathcal{N}, \mathcal{V}, I, \mathcal{C}, n^i, \mathcal{E}, \iota, \rho >$ *où, en notant* $\mathcal{B}(\mathcal{X})$ *l'ensemble des contraintes sur un ensemble* $\mathcal{X}$ *de variables quelconques :*

$$\mathcal{B}(\mathcal{X}) \triangleq \{x \triangleright n | n \in \mathbb{N}, x \in \mathcal{X}, \triangleright \in \{=, \leq, \geq, >, <\}\} \cup \emptyset$$

et $Inc(\mathcal{I}) \triangleq \{(i \leftarrow 0),(i \leftarrow i+1),(i \leftarrow i-1) | i \in \mathcal{I}\}$ *les actions possibles sur un ensemble* $\mathcal{I}$ *de variables entières (incrément ou réinitialisation),*

- $\mathcal{N}$ *représente un ensemble fini de nœuds,* n^i *est le nœud initial de l'automate*
- $\mathcal{V}$ *est un ensemble fini d'horloges*
- $\mathcal{I}$ *est un ensemble fini de variables entières*
- $\mathcal{C}$ *est l'ensemble des canaux (permettant la synchronisation entre automates).*
- $\mathcal{E} \subseteq \mathcal{N} \times (\mathcal{B}(\mathcal{V}) \cup \mathcal{B}(\mathcal{I})) \cup \{\mathbf{tt}, \mathbf{ff}\} \times \mathcal{C} \times 2^{\mathcal{V}} \times Inc(\mathcal{I}) \times \mathcal{V} \times \mathcal{N}$ *est un ensemble fini de transitions.* $e =< n, g, \alpha, i, r, n_e > \in \mathcal{E}$ *est une transition ayant pour source le nœud n et pour destination le nœud* n_e*,* α *est l'émission de la donnée associée au canal* α*, g est la garde de la transition c'est à dire la condition à respecter pour pouvoir franchir la transition, i les incréments/décréments ou remise à zéro de variables entières, et r l'ensemble des horloges à réinitialiser lors de la transition.*
- ι *est une fonction* $\mathcal{N} \rightarrow \mathcal{B}_c(\mathcal{V}) \cup \{tt\}$ *associant un invariant à chaque nœud.*
- ρ *est une application* $\mathcal{N} \rightarrow \{-, C\}$ *associant un type à chaque nœud :*
  - $-$ *: noeud normal ;*
  - C *: noeud "committed"*

La sémantique des automates temporisés considérés est celle des automates du model-checker UPPAAL *([BY]).*

Nous ne donnerons ici que le principe de la traduction, l'algorithme formel dépassant le cadre de cette thèse. La traduction du pivot en automate temporisé se fait en deux phases, la première consiste en la traduction de fonctions, la deuxième en celle des liens de communication.

6.5.2 Traduction d'une fonction

La traduction d'une fonction se fait à partir de l'automate du processus logiciel associé en y ajoutant les gardes temporisées extraites du pivot. L'automate résultat sera constitué des éléments suivants :

$\mathcal{N}$ **:** L'ensemble des nœuds de l'automate temporisé est construit de la façon suivante : à chaque état de l'automate du processus -provenant du MP de la facette logiciel- correspond un unique nœud de l'automate temporisé résultat, nous appellerons cet ensemble les nœuds "natifs" par opposition au nœuds "induits" par la traduction.

$\mathcal{V}$ **:** chaque automate se voit doté d'une unique horloge *x*.

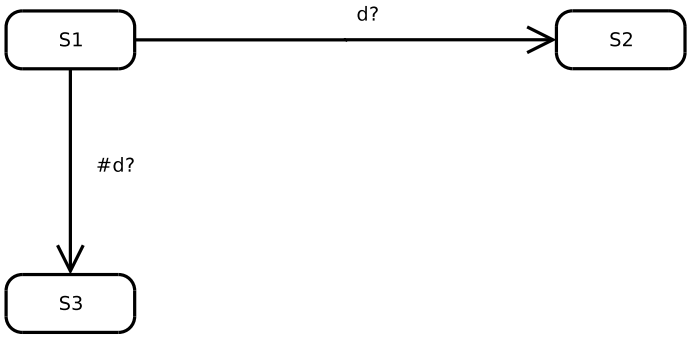
$\mathcal{C}$ **:** à chaque donnée *D* (`P_Data`) émise par la fonction est associé un canal *DOut*

$\mathcal{I}$ **:** à chaque donnée *D* (`P_Data`) reçue par la fonction est associée une variable entière *DIn*. Cette variable, partagée avec le lien émetteur, est décrémentée par la fonction au moment de sa consommation de la donnée *D*.

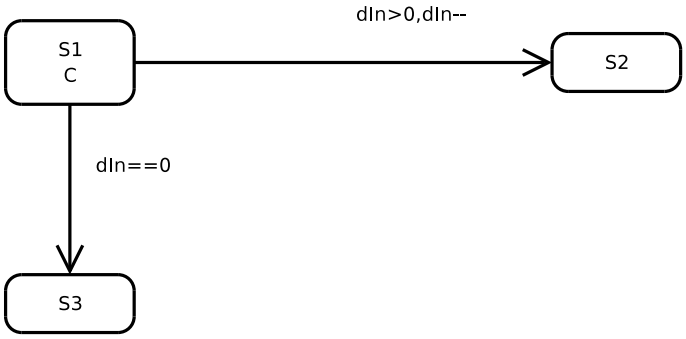
Remarque 8 *En pratique, les canaux et variables seront déclarés globalement pour le système en UPPAAL puisqu'ils sont partagés par (au moins) deux automates pour communiquer (un lien et une (des) fonction(s))*

La construction des transitions, gardes et invariants est détaillée ci-après, en étudiant les configurations locales rencontrées en parcourant l'automate du processus associé à la fonction étudiée.
La figure 6.9 page suivante décrit la réception d'une donnée.

Automate Facette logiciel	Automate Pivot	Commentaire
Si D ?	Si DIn > 0	Cas où D est disponible
transition vers S3	DIn <- DIn - 1	La donnée est consommée
	transition vers S3	(une occurrence de DIn en moins disponible)
Si #D ?	Si DIn == 0	La donnée n'est pas disponible
transition vers S3	transition vers S3	L'automate passe dans l'état spécifié



Automate du Processus associé



Automate temporisé résultat

FIG. 6.9 – Traduction de la réception

La figure 6.10 page suivante décrit l'émission des données. La sémantique des fonctions est telle que les données sont émises *simultanément*, c'est pourquoi lorsque l'automate du processus émet ses données (séquenciellement), le nœud associé au premier état "émetteur" est doté d'un invariant $x <= WCET$ et la transition correspondante une garde $x => BCET$ (la séquence d'émission doit être

effectuée entre le BCET et le WCET par définition). Les autres nœuds de la séquence sont déclarés `Committed` pour que celle-ci se déroule en temps nul.

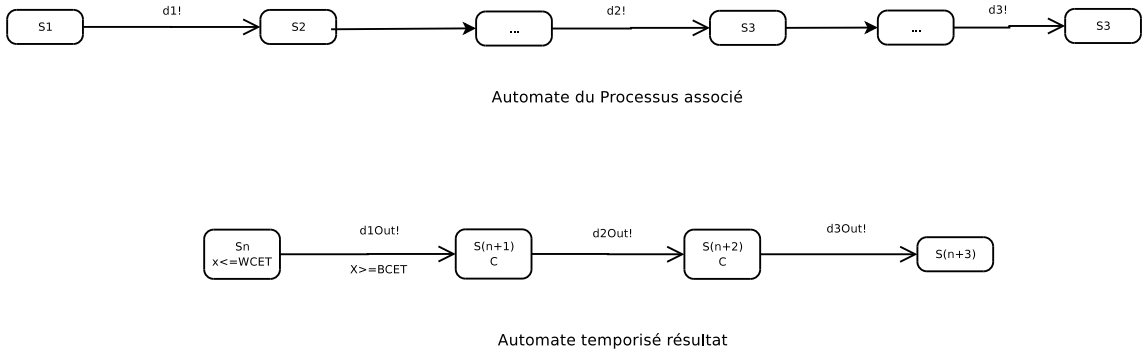


FIG. 6.10 – Traduction de l'émission

Le dernier cas à traiter est celui du changement de période (transition γ figure 6.11). Le nœud de départ de la transition associée se voit doté de l'invariant $x <= T$, où T désigne la période de la fonction. La transition est gardée par $x == T$ (tirable si l'horloge vaut exactement la période). L'horloge x est ensuite remise à 0.

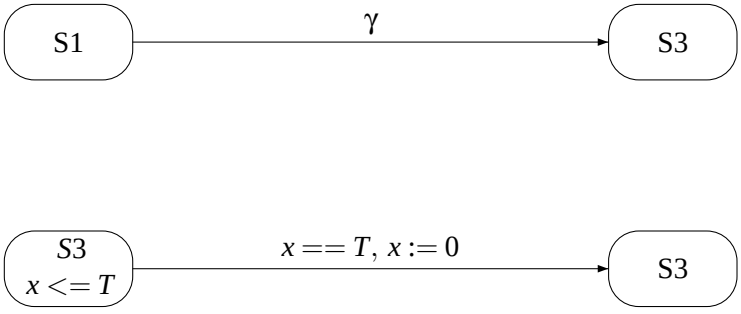


FIG. 6.11 – Traduction de la fin de période

La figure 6.12 illustre le traitement de la gigue d'activation δ. Il faut ajouter un état initial à l'automate temporisé muni de l'invariant $x <= deltaMax$ (l'automate est bloqué dans cet état durant un temps au plus égal à la borne supérieure de la gigue). Une transition est ensuite tirée vers le premier état natif de l'automate avec la garde $x >= deltatMin$ (l'automate est bloqué dans l'état initial au moins pendant la borne inférieure de la gigue), l'horloge est alors réinitialisée.

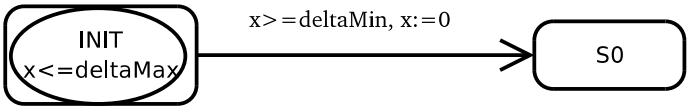


FIG. 6.12 – Initialisation

6.5.3 Traduction d'un lien

La traduction d'un lien du pivot en automate temporisé est l'exécution parallèle d'instances du pattern décrit figure 6.13 : une instance pour chaque place du buffer et pour chaque donnée acheminée par le lien. Lorsqu'une fonction émet sur le canal associé au lien l (composé de $l_1, l_2, \ldots, l_n$ des patterns de la figure 6.13 en parallèle), l'un des l_i se synchronise avec la fonction et passe dans l'état d'attente et y reste durant un délai compatible avec la latence de communication du lien. A l'issue de celui-ci, soit il reste une place dans le buffer du lien et la donnée est rendue disponible (incrément de la variable entière associée), soit le buffer est plein, auquel cas l_i passe (instantanément) par l'état FULL, le message émis par la fonction étant alors perdu.

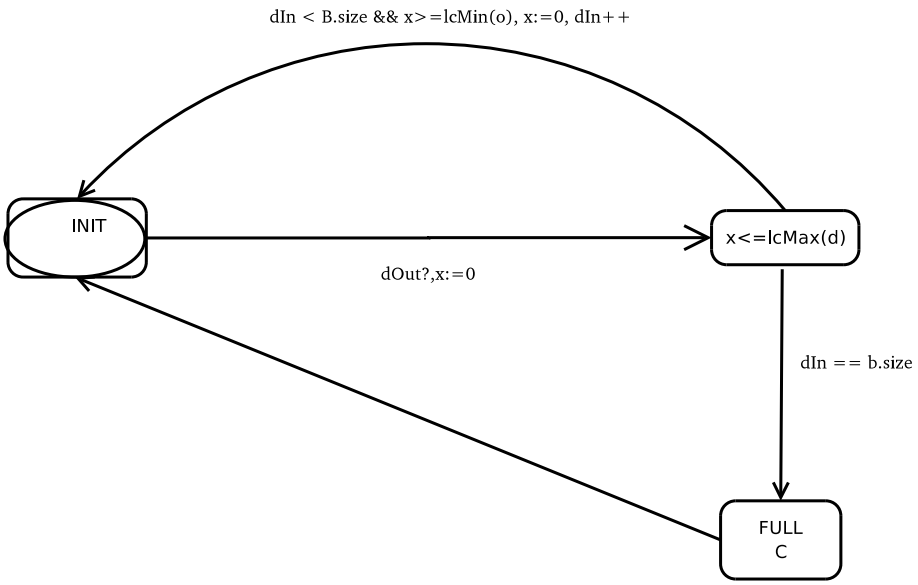


FIG. 6.13 – Traduction d'un lien

Remarque 9 *L'algorithme informel que nous venons de présenter a pour vocation de décrire la traduction vers UPPAAL dans le cas général. Les automates de l'étude de cas présentés à l'annexe A page 121 intègrent des optimisations liées à notre connaissance du système. Les communications asynchrones, par exemple, se font par des variables booléennes et non entières car les liens du pivot ont un buffer d'une place.*

6.5.4 Vérification de propriétés

Les `CheckingConstraints` du layer model-checking expriment des exigences dans le formalisme logique de l'outil de vérification cible (la logique temporelle d'Uppaal dans notre cas). La limitation de ces logiques et le caractère non fonctionnel des exigences que l'on souhaite vérifier nécessitent l'introduction d'automates observateurs d'une part et de points d'observation (variables) d'autre part. Par exemple, la logique d'Uppaal n'étant pas temporisée, s'il est possible d'exprimer que l'état X de l'automate A est toujours atteint en temps fini (`A<> A.X`), il est en revanche impossible d'exprimer que cet état est atteint en moins de n unités de temps. Pour vérifier cette propriété, il faut introduire un automate observateur passant dans un état d'échec si durant le délai fixé, l'automate observé n'est pas passé dans X. Les figures suivantes illustrent cet exemple.

Exemple 8 *Il est facile de vérifier que l'état X de l'automate A est toujours atteint en temps fini :*

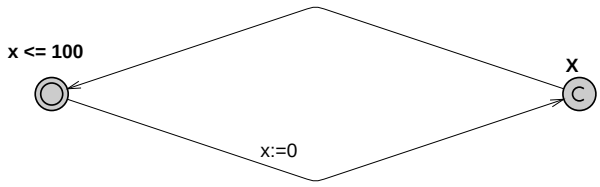


FIG. 6.14 – l'automate A

```
A<>␣A.X
property␣is␣satisfied
```

Pour quantifier ce temps, en revanche, il faut introduire l'observateur de la figure 6.16 page suivante et ajouter la synchronisation `ok!` *à l'automate observé pour rendre observable la transition vers X (figure 6.15).*

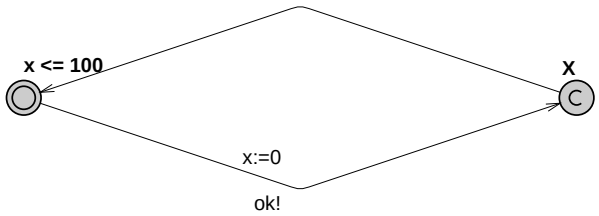


FIG. 6.15 – l'automate A augmenté

La démonstration que X est atteint au plus toutes les 100 unités de temps (`delai:=100`) est effectuée par la vérification de la formule

```
A[]␣not␣obs.UNHAPPY
```

qui se lit : "il est toujours vrai que l'automate obs ne se trouve pas dans l'état UNHAPPY" Dans ce cas UPPAAL répond :

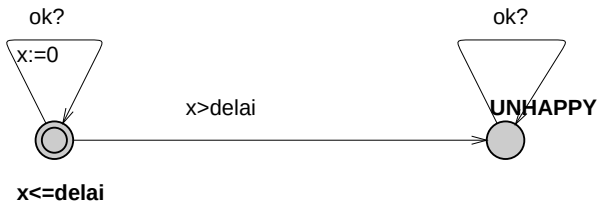


FIG. 6.16 – l'automate observateur

`property␣is␣satisfied`

Si en revanche nous tentons de montrer que X n'est pas atteint toutes les 50 unités de temps (en fixant `delai:=50`) avec la même formule, UPPAAL répond cette fois :

`property␣is␣not␣satisfied`

La création des observateurs ainsi que l'ajout des points d'observation est une tâche effectuée manuellement.

6.5.5 Application à l'étude de cas

Les automates Uppaal de l'étude de cas sont présentés dans l'annexe C.2.1 page 140.

6.6 Layer reconfigurabilité

Les automates générés par le layer model-checking présenté paragraphe 6.5 page 85, modélisent le comportement nominal des fonctions. En effet, les processus de la facette logiciel sont supposés fiables et s'ils ne l'étaient pas, il serait impossible pour la facette de fournir une description des erreurs dans ses automates (si les erreurs étaient identifiées, elle les corrigerait). Nous proposons un layer chargé de modéliser des pannes matérielles basé sur une modification des automates temporisés du layer model-checking et l'introduction d'un "serveur de pannes" modélisant les senarii de défaillance. Les automates temporisés des fonctions sont modifiés de la façon suivante : à chacun de leurs états est ajouté une transition menant à un état puit. Ces transitions sont déclenchées par la réception d'un message lié à la perte spécifique de la ressource de calcul qui exécute la fonction considérée. Cette information est extraite du layer mapping.

Remarque 10 (Cas des états commited) *Il n'est pas nécessaire d'ajouter ces transitions aux états* `commited` *dans la mesure où ceux-ci sont traversés sans délai, et que, par conséquent le message de panne ne peut être reçu au moment de la traversée d'un tel état.*

Les senarii de défaillance sont modélisés par un automate temporisé appelé serveur de pannes. Celui-ci peut envoyer des messages signifiant la perte de ressources de calcul à tous les autres automates du système.

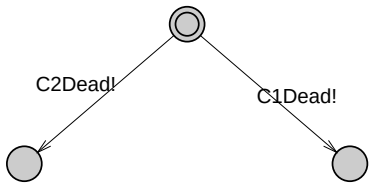


FIG. 6.17 – Serveur de panne

Application à l'étude de cas

Les automates de l'étude de cas soumis à défaillances sont présentés à l'annexe C.2.2 page 146.

La figure 6.17 décrit le serveur de panne utilisé pour vérifier l'étude de cas. Nous avons pris l'hypothèse que les deux calculateurs ne sont en aucun cas défaillants simultanément. Sous cette hypothèse, le model-checking du système couvrira tous les comportements de panne possibles.
Les propriétés que nous souhaitons vérifier sont les suivantes :
– P1 et P4 : les processus terminaux (Commande et Mission) et leur secours ne sont jamais en mode nominal simultanément ;
– P3 : le système est asservi par une commande calculée à partir de données viables au moins une fois par seconde ;
– P6 : l'état du système est stocké au moins une fois par seconde ;
– P2 et P5 : en cas de panne du calculateur principal le processus secours de commande passe en mode nominal en moins de 100ms, celui de Mission en moins de 500ms.

La figure 6.18 page suivante illustre le layer chargé de vérifier ces propriétés. Pour des raisons de clarté les `LayerLocalData` représentant les automates Uppaal du système ainsi que les observateurs et le serveur de pannes sont abstraits par le package `ATUppaal`. La contrainte `IntegriteDuModele` vérifie (de préférence préalablement) que la modélisation Uppaal n'a pas introduit de deadlock. Il s'agit là d'une propriété du modèle, non du système, mais il est fondamental de la garantir sans quoi les résultats sur les propriétés de sûreté n'auraient aucune valeur. En effet il est possible qu'un état redouté soit accessible dans le système mais pas dans le modèle Uppaal du fait d'un deadlock introduit par le modèle synchrone.

Les figures 6.19 à 6.22 fournissent les observateurs utilisés par le layer de la figure 6.18 page suivante.

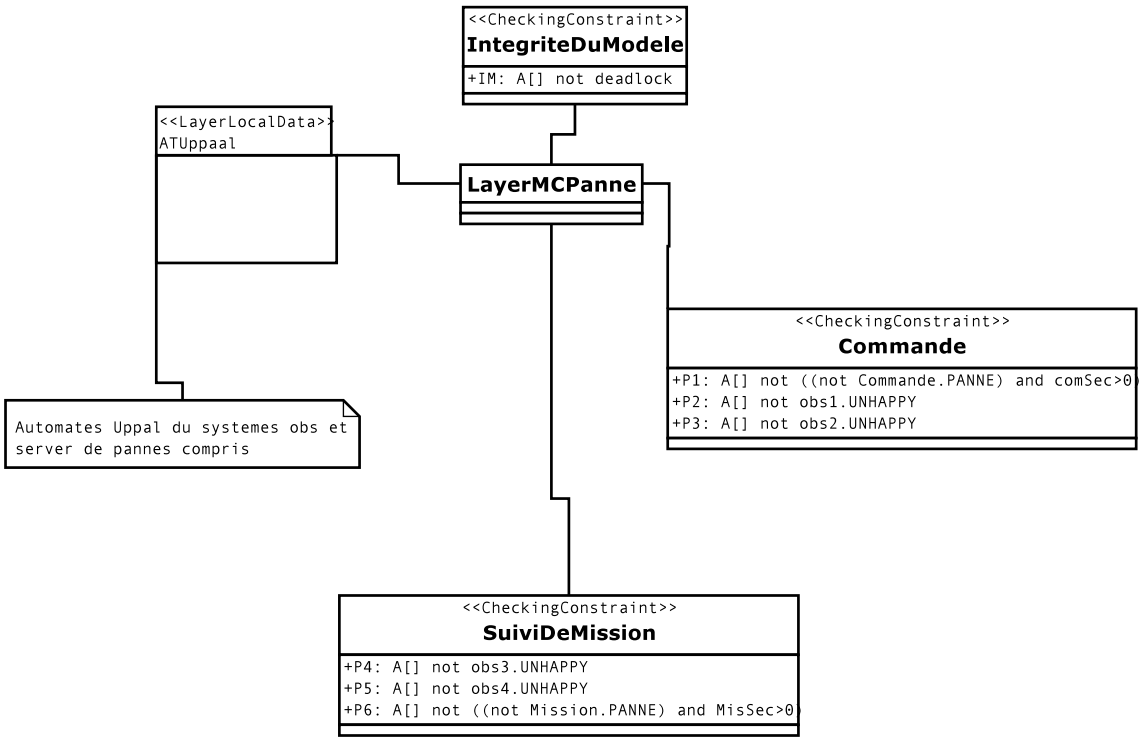


FIG. 6.18 – Layer model checking avec défaillance

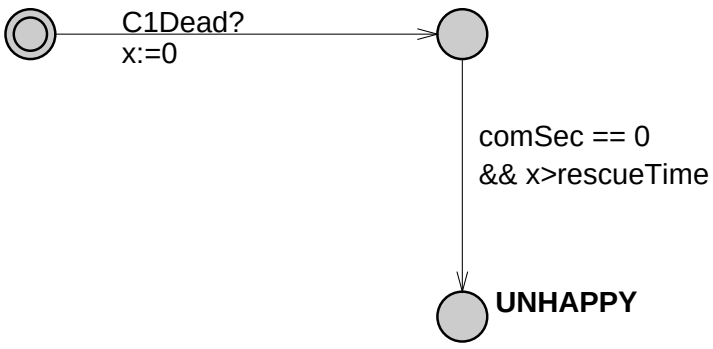


FIG. 6.19 – obs1 (P2 avec const rescueTime 100)

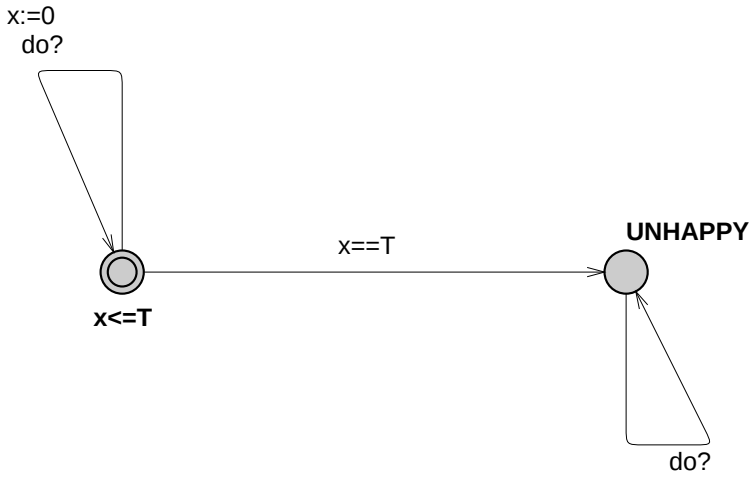


FIG. 6.20 – obs2 (P3 avec const T 500)

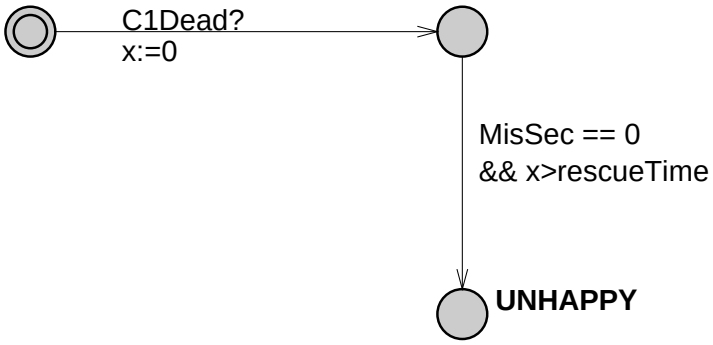


FIG. 6.21 – obs3 (P5 avec const rescueTime 500)

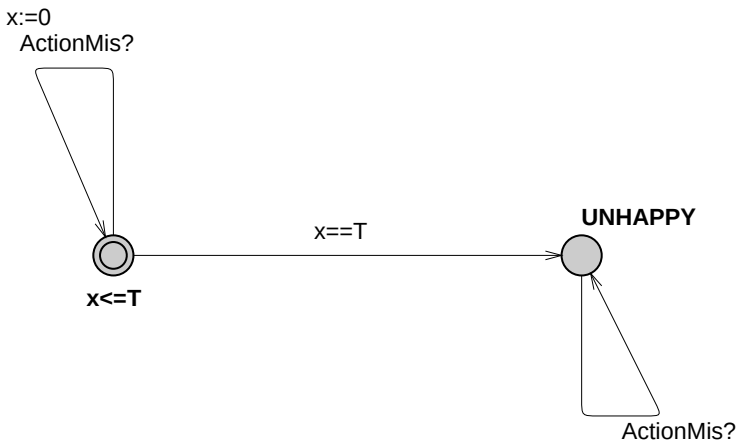


FIG. 6.22 – obs4 (P6 avec const T 1000)

Résultats

Toutes les propriétés soumises à vérification par UPPAAL sont satisfaites
Les résultats de la vérification des propriétés des contraintes du layer 6.18 page 94 par le model-checker Uppaal sont présentés ci-dessous :

```
A[] not ((not Commande.PANNE) and comSec>0)
property is satisfied


A[] not ((not Mission.PANNE) and MisSec>0)
property is satisfied

/*
const rescueTime 100
*/
A[] not obs1.UNHAPPY
property is satisfied


/*
const T 500
*/
A[] not obs2.UNHAPPY
property is satisfied


/*
const rescueTime 500
*/
A[] not obs3.UNHAPPY
property is satisfied


/*
const T 1000
*/
A[] not obs4.UNHAPPY
property is satisfied
```

6.7 Layers et complexité des systèmes : notes spéculatives

La notion de layer telle que présentée ici est fortement liée à celle d'exigence système. L'analyse de ces exigences pour évaluer *a priori* la complexité du système et de sa réalisation est un enjeu majeur

de l'ingénierie système. Cette analyse se heurte actuellement au manque de formalisme permettant d'exprimer les exigences. Les layers peuvent se présenter comme un bon candidat de formalisme supportant une analyse de complexité des systèmes. Cette courte partie présente une étude prospective de la notion de complexité des exigences en s'appuyant sur le formalisme des layers dont le but n'est pas de proposer une formule mais de suggérer une approche de recherche. Nous pouvons proposer deux axes de définition de la complexité des layers. La première basée sur la complexité de calcul des contraintes associées et la deuxième liée à la structure même du layer.

6.7.1 Complexité Algorithmique

Les contraintes associées à un layer sont exprimées en langage exécutable et doivent être évaluées algorithmiquement. Si nous associons à une contrainte la complexité du problème de son évaluation, la complexité algorithmique d'un layer peut être définie comme la somme des complexités de ses contraintes. Bien qu'importante dans le cadre initial d'utilisation des layers, cette complexité algorithmique ne nous semble *a priori* pas porter de réelles informations quant à la complexité intrinsèque des exigences système. Pour cela il semble plus prometteur de se pencher sur la complexité structurelle des layers.

6.7.2 Complexité structurelle

En dehors de ses contraintes, un layer est composé de plusieurs types d'objets :
– Des objets offerts par les facettes ;
– Des objets du pivot ;
– Éventuellement, des objets requis de facettes.
Il semble naturel d'évaluer la complexité propre d'un layer à partir du nombre d'objets offerts par les facettes car les autres ne sont qu'un espace de stockage et n'apportent pas réellement d'informations. De plus, il est possible de pondérer l'influence d'un objet particulier sur un layer avec le nombre d'attributs de cet objet dont se sert la contrainte du layer. À cette complexité propre, il faut bien sûr ajouter celle des layers dont il dépend. Nous aboutissons ainsi à un processus récursif d'évaluation de la complexité. Plusieurs questions demeurent néanmoins ouvertes : tout d'abord, puisque cette méthode se base sur la dépendance entre les layers (donc les exigences), est-on en mesure de résoudre une éventuelle circularité qui surviendrait dans les dépendances ? Ensuite, la complexité ainsi évaluée est-elle caractéristique du système où bien est-elle fortement sensible au découpage en layers choisi ? Dans ce cas, existe-t-il un découpage canonique des exigences permettant d'évaluer la complexité intrinsèque du système ? Ces questions sont évoquées ici à titre de pistes de recherche et nous ne tenterons pas d'y répondre dans le cadre de cette thèse.

6.8 Conclusion de la deuxième partie

L'objectif de cette partie était de présenter les outils théoriques et conceptuels sur lesquels s'appuie notre approche.
La figure 6.23 page suivante rappelle les relations liant ces différents concepts.
Il ressort de cette partie tout d'abord une formalisation de la vision du système qu'ont les intervenants et le MOE respectivement par les notions de facette et de pivot puis l'introduction d'un mécanisme assurant la cohérence des facettes et de leur bonne intégration au sein du système global :

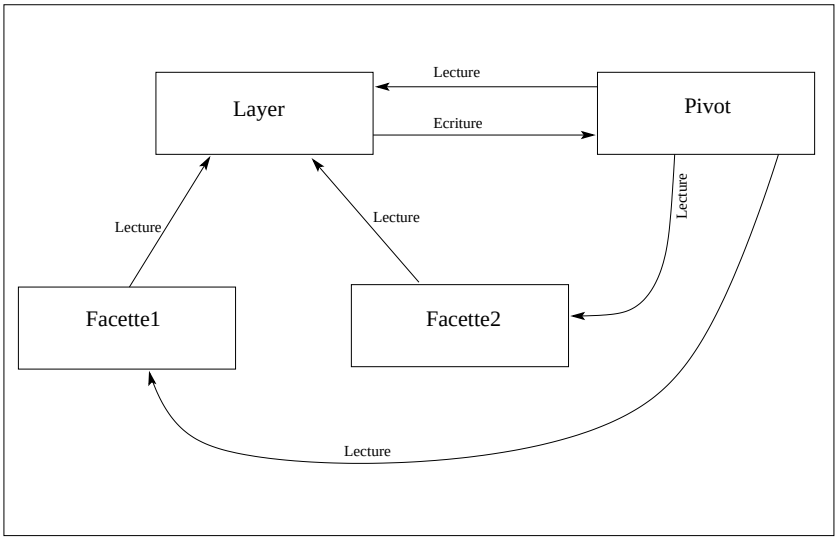


FIG. 6.23 – Synthèse des relations entre facettes, pivot et layers

les layers. Ces layers constituent le point central de nos travaux. En offrant une réponse à la problématique de la gestion des propriétés transverses, ils justifient la définition des autres concepts de modélisation que nous avons choisis. En effet, définir un ensemble de modèles séparés décrivant un système n'a de sens que si l'on s'assure de leur cohérence au regard des exigences transverses. Cette partie décrit notre approche dans un contexte statique hors de tout processus de développement. Il nous faut maintenant montrer comment les outils précédemment introduits peuvent s'intégrer dans une dynamique de développement. C'est le but de la partie suivante qui décrit deux processus de développement envisageables tirant partie des spécificités de nos modèles lors de deux cas d'utilisation, et donne des exemples de problématiques diverses pouvant se présenter lors du développement d'un système ainsi que la réponse apportée par notre approche.

Troisième partie

Méthodologie : utilisation des modèles et structures

Introduction

Les structures introduites précédemment permettent de décrire un système à un instant donné de son développement. Les exemples tirés de l'étude de cas comportent des données issues de choix de conception qui n'ont lieu qu'à un stade avancé du développement. Il nous faut maintenant montrer comment l'ensemble des concepts que nous avons introduits s'intègre au sein d'un processus de développement et quelle plus-value il apporte au MOE en terme d'aide à la conception. Cette partie présente deux cas d'utilisation possibles de notre méthode pour différents cas de figure pouvant se présenter lors du développement de systèmes embarqués. Dans un premier temps, nous étudions le cas du développement d'un système complet, c'est à dire sans réutilisation de partie existante. Nous fournissons un processus de développement tirant partie des structures définies précédemment. Dans un deuxième chapitre, nous montrons comment notre approche permet la réutilisation d'une partie existante au travers de l'étude du dimensionnement d'une plate-forme matérielle devant s'adapter à un ensemble de logiciels applicatifs.

Chapitre 7

Processus de développement “ex nihilo”

“Mes clients sont libres de choisir la couleur de leur voiture à condition qu’ils la veuillent noire.”
Henry Ford

Ce chapitre présente notre méthode dans le cadre d’un système développé à 100% à partir des exigences système. Autrement dit, sans réutilisation de parties provenant de systèmes préexistants. Dans ce cas, le processus induit par l’utilisation d’une telle structure de gestion des modèles et de leurs relations peut se découper en deux phases principales : une phase de négociation et une phase “à modèles stables”. Ce comportement se rapproche assez de ce qui peut être observé dans les développements industriels, où le choix des partenaires, puis les choix technologiques et la distribution des tâches donnent lieu à négociation. La première phase comprend l’établissement d’un consensus sur le choix des méta-modèles employés par les différents intervenants, puis sur le méta-modèle du pivot. Suit une phase de définition de la structure du pivot en lui-même, en analysant les éléments de solutions préliminaires apportés par les facettes. La seconde phase correspond à un enrichissement de cette structure, qui consiste en la précision des attributs du pivot, et en la convergence vers le modèle-solution. Nous allons maintenant détailler les différentes étapes de ce processus, schématisé par la figure 7.1 page suivante.

7.1 La phase d’initialisation

Cette phase, illustrée figure 7.2 page 105 a pour but de définir des modèles-solution pour chacune des facettes ainsi que le modèle pivot destiné au Maître d’œuvre. La première étape consiste pour le Maître d’Oeuvre en l’analyse et la compréhension des exigences systèmes émises par le Maître d’ouvrage. Cette étape est fondamentale, car d’une part elle conditionne la bonne réalisation du système, et d’autre part parce que beaucoup de projets échouent du fait d’une mauvaise interprétation des exigences (cf. [Int99]). L’interprétation des exigences par le Maître d’Œuvre conduit à

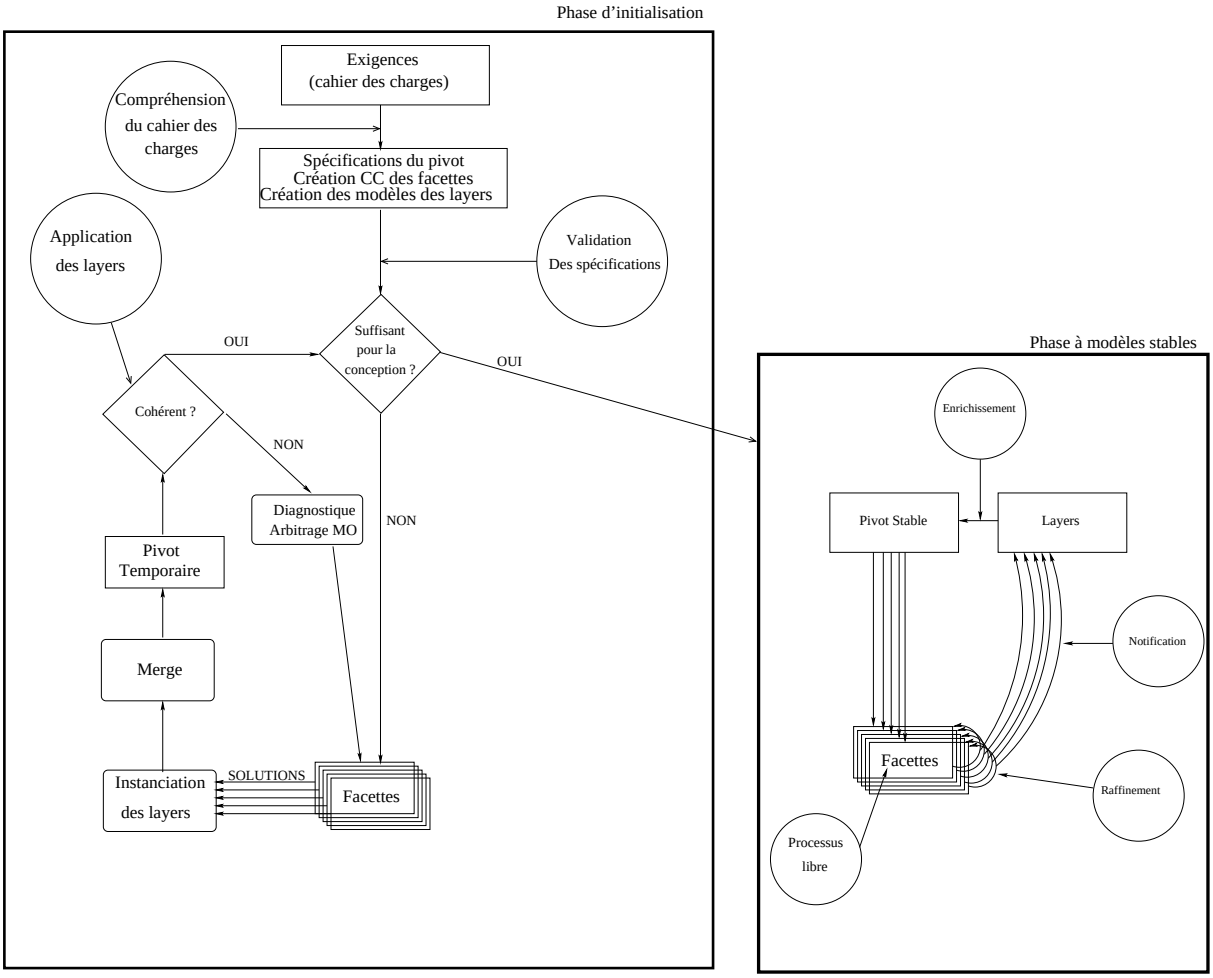


FIG. 7.1 – Processus de développement

l'élaboration d'un ensemble de spécifications techniques plus formelles (sous forme de modèles). Les activités d'analyse des exigences consistent en :
– comprendre les missions du système,
– construire une base de données des exigences (besoin et système),
– définir une architecture fonctionnelle préliminaire explicitant les capacités du système (et du système de soutien)

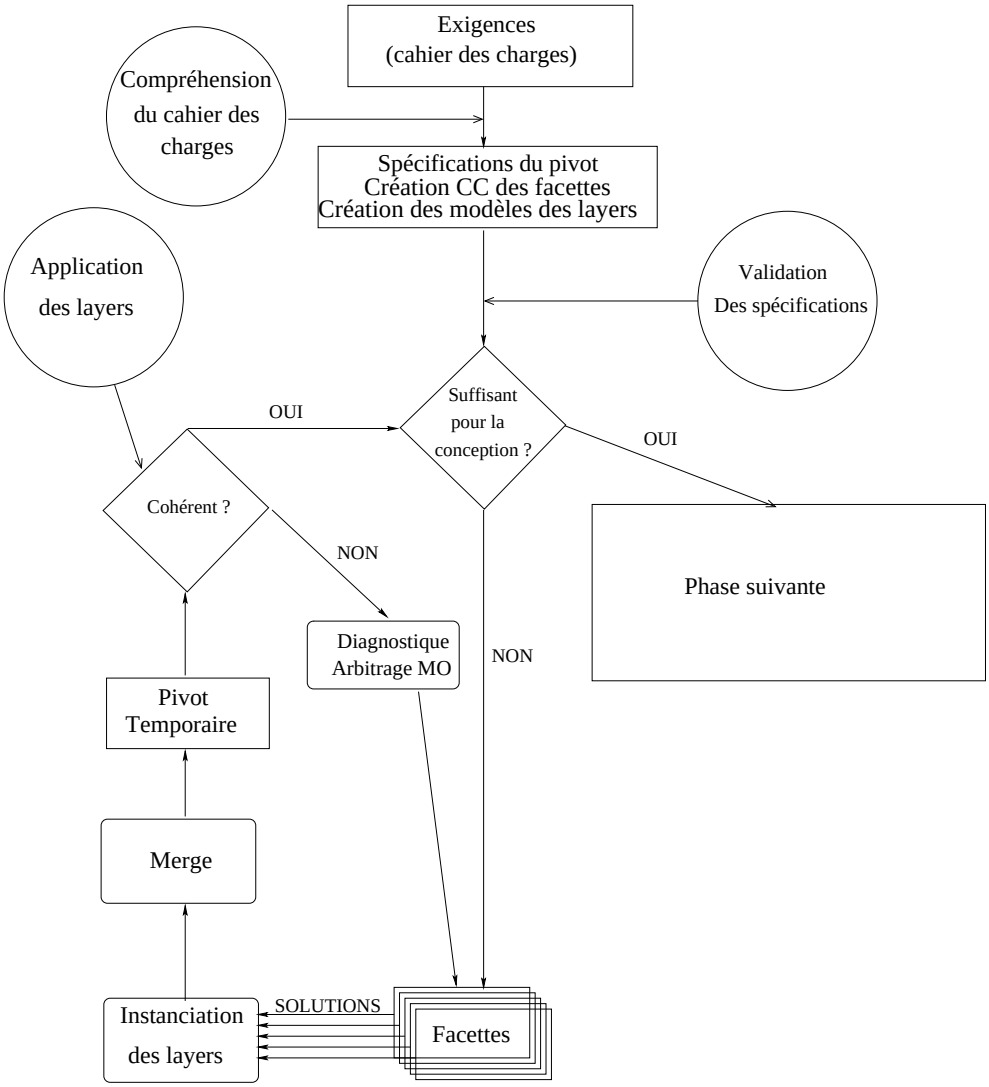


FIG. 7.2 – Phase d'initialisation

Dans cette phase d'analyse les outils de gestion des exigences et modélisation d'architecture se complètent, l'un apporte la maîtrise de la base des exigences et l'autre la représentation des missions et scénarios opérationnels et de soutien qui permettront de préciser le besoin et les exigences. En particulier, les diagrammes de séquences représentant les scénarios opérationnels du système constituent un outil essentiel pour comprendre le besoin. La modélisation de missions et services du système s'appuie sur les "uses cases" UML pour la partie statique. Les scénarios opérationnels, qui décrivent dynamiquement des utilisations du système, sont modélisés en utilisant les diagrammes de

séquences UML. Le comportement général du système décrit par des changements d'états est modélisé à l'aide des diagrammes d'état-transition de UML. Une fois les exigences interprétées, le Maître d'Œuvre dispose de spécifications du système très abstraites qui sont considérées comme le premier pivot. Elles se présentent sous la forme d'un pivot précisant les valeurs des attributs des objets du système sous forme d'intervalles de valeurs admissibles en regard des exigences initiales. L'interprétation des exigences permet d'établir des modèles de layers qui seront instanciés par la suite. Ce pivot, bien qu'abstrait, permet une première vérification de la cohérence des spécifications établies à la suite de l'analyse des exigences. En vérifiant la compatibilité numérique des encadrements de valeurs des attributs du système entre eux et avec les relations qui les lient, il est possible de détecter (numériquement) des erreurs dans la compréhension des exigences ou dans les exigences elles-même. Malheureusement, du fait de la souplesse des exigences initiales, cette analyse ne garantit pas la validité des spécifications ; plus les exigences sont strictes, meilleure est la qualité du diagnostic effectué à cette étape. À ce stade, le modèle du pivot ne reflète que les spécifications et ne fournit pas d'éléments de solution, il est donc insuffisant pour entamer la conception. En revanche il permet d'identifier les différentes facettes métier nécessaires et de déterminer les spécifications de chacune d'elles à partir de celles du pivot. La réalisation de ces spécifications sera soumise à appel d'offre pour déterminer les sous-traitants qui seront choisis pour être en charge des différentes facettes.
L'étape suivante consiste pour les facettes à proposer un modèle solution abstrait de leur partie du système (un Modèle Public). Pour clarifier notre propos nous considérerons qu'à une facette est affecté un et un seul sous-traitant dès le début, mais il nous faut préciser la marche à suivre en cas d'appel à concurrence ; cette concurrence ne joue que lors de la phase d'initialisation : lors de la conception effective il est naturel de considérer les sous-traitants définitivement choisis. La concurrence entre sous-traitants pour une facette donnée revient à considérer que cette facette propose plusieurs modèles solution (un par candidat) qu'il faudra traiter un à un.
Un fois les MP des facettes connus, les layers précédemment choisis sont instanciés, en commençant par le layer mapping. Celui-ci permet de fusionner les modèles proposés par les facettes pour obtenir le pivot correspondant (boîte « Merge »). Ce pivot doit être validé : les solutions des différentes facettes doivent être compatibles entre elles (en plus de respecter leur cahier des charges). Les layers de type validation (dont les contraintes sont des `CheckingConstaints`) doivent être vérifiés. C'est à dire que les intervalles de valeurs possibles pour les attributs proposés (lorsqu'ils existent) par les MP permettent une satisfaction de ces layers.
Si le pivot n'est pas cohérent, une facette au moins doit changer son modèle solution. La question de savoir laquelle, n'a en général, pas de réponse unique et évidente. En effet, il n'est pas question ici d'erreur d'un sous-traitant, mais d'une incompatibilité entre les choix techniques de plusieurs intervenants qui respectent individuellement leur cahier des charges (celui-ci laissant une assez grande latitude pour ne pas figer les technologies *a priori*). Le maître d'œuvre doit alors donner son arbitrage et désigner les facettes qui doivent corriger leur modèle solution et préciser leurs spécifications afin d'introduire les contraintes induites par les modèles solution retenus. Par exemple, dans le cadre de notre développement de flotteur, si l'une des exigences est que le coût de développement doit être inférieur à 100 000 euros, il est difficile de spécifier *a priori* le coût maximum respectif de chacune des facettes. À l'issue des premières propositions, si la facette logiciel propose 20 000 euros, la facette électronique 30 000, la mécanique 50 000 et l'automatique 15 000, on aboutit à un total de 115 000 euros. Le maître d'œuvre doit négocier avec chacune des facettes pour rediscuter de leur cahier des charges. Il peut dans le cas qui nous préoccupe décider de diminuer les contraintes temps réel du logiciel afin de diminuer la puissance de l'électronique, et ainsi faire diminuer le coût de celle-ci. Ceci est possible car le pivot comporte les formules liant les

contraintes temps réel du logiciel à la puissance des processeurs. Ce processus est réitéré jusqu'à l'obtention d'un pivot cohérent suffisamment détaillé pour permettre d'entrer en phase de réalisation proprement dite.

7.2 La phase à modèles stables

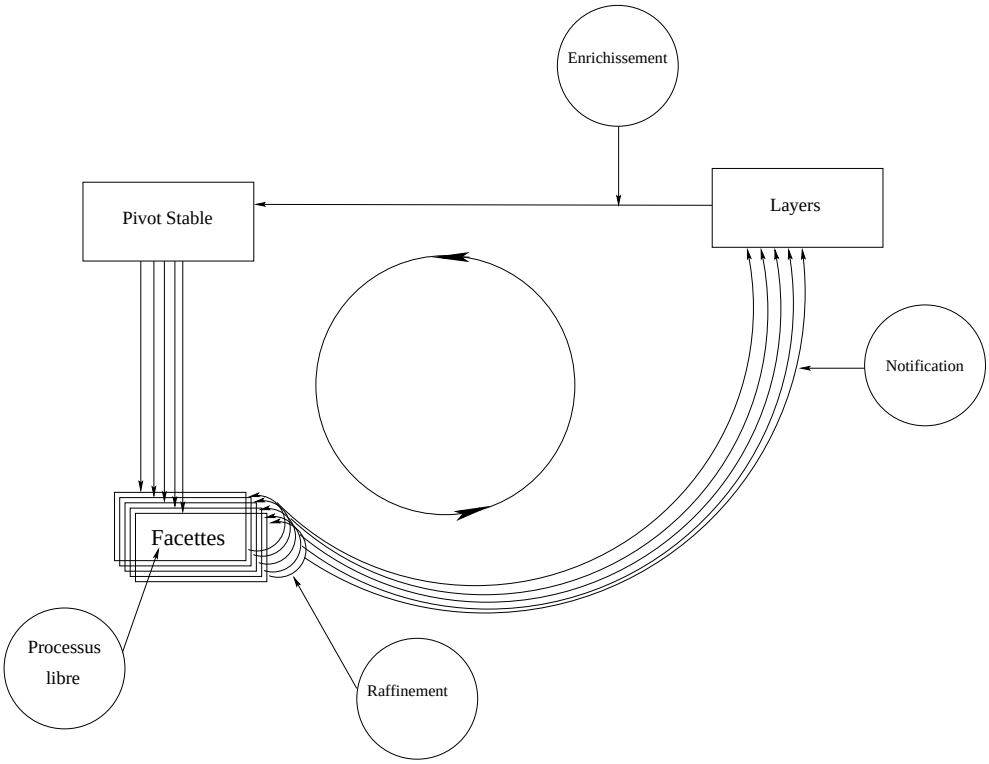


FIG. 7.3 – Phase à modèles stables

Lors de la phase de conception (figure 7.3), appelée phase à modèles stables, chaque facette réalise sa partie du système en employant le processus de son choix. *In fine* leur partie doit être conforme au Modèle Public défini lors de la phase d'initialisation. Au cours du développement la structure des Modèles Publics ne doit pas changer, en revanche les valeurs des attributs laissées sous forme d'intervalles acceptables à l'issue de la phase d'initialisation sont précisées par les facettes dès qu'elles effectuent un choix de conception. Les layers sont alors notifiés et mettent à jour toutes les valeurs des attributs du pivot qui dépendent de ceux qui ont été précisés et vérifient que cette précision n'entraîne pas de violation des exigences. Les autres facettes sont ainsi informées. Ce processus permet au MO d'avoir une connaissance en temps réel de l'avancement du développement de son système. De plus, il est le garant de la cohérence sémantique et syntaxique des informations échangées par les sous-traitants dans la mesure où ils ne peuvent communiquer directement et que les layers traduisent les informations provenant d'une facette dans les langages compréhensibles par les autres (celui du pivot). Les risques d'erreurs dûs à l'incompréhension entre sous-traitants sont donc réduits. Ce processus permet en outre, grâce à une séparation claire des exigences par domaine de diminuer la taille des modèles à vérifier et le niveau d'abstraction auquel ces vérifications ont lieu

(celui du pivot). Il rend donc possible l'application des méthodes formelles augmentant ainsi la confiance que l'on peut accorder au système *a priori* (c'est à dire au plus tôt dans le cours du processus de développement). Une fois le système réalisé, les Modèles Publics des facettes et le pivot final permettent de faciliter la maintenance et le maintien en condition opérationnelle du système. En effet, cet ensemble de modèles donne une vision claire des dépendances entre les parties du système ainsi que les responsabilités de chacun.

Chapitre 8

Réutilisation d'une facette

"A complex system that works is invariably found to have evolved from a simple system that works."
Linux fortune (anonyme)

De nombreux systèmes se construisent autour d'une partie préexistante. Soit en réutilisant une composante d'un ancien système, soit en utilisant des macro-composants sur étagère. Dans ce cas, le processus de développement doit s'adapter à la fois pour éviter les écueils de la perte de souplesse induite par la présence d'un composant "boîte noire" et en tirer partie pour accélérer le développement du reste du système en limitant l'éventail de choix de conception et en préparant au plus tôt l'intégration finale du système. Au cours de ce chapitre nous allons montrer comment nos structures, et plus particulièrement le concept de layer permettent de répondre à cette problématique. Pour ce faire nous considérerons le cas de la conception d'une plate-forme matérielle en vue de la réutilisation de logiciels existants. Un des problèmes majeurs qui se posent dans ce cas est le dimensionnement de la plate-forme. Nous nous proposons de dimensionner la facette matériel du flotteur connaissant les logiciels qu'elle devra exécuter. Nous prendrons l'hypothèse simplificatrice que la seule contrainte temps réel sur les fonctions est qu'elles aient terminé leurs traitements (émis toutes leurs données) dans un temps inférieur à leur période. La prise en compte des gigues ferait appel à des techniques d'une complexité dépassant largement le cadre de cette thèse.

8.1 Étapes du processus

La première étape du processus consiste en la réalisation d'un Modèle Public complet à partir de l'analyse du logiciel (rétro-ingénierie des modèles, analyse statique du code, etc). Dans notre exemple nous obtenons le modèle de la figure 4.5 page 55. Parallèlement, un pivot primaire est créé à partir du cahier des charges. Les valeurs des attributs de ses objets sont pour la plupart encore indéterminées à ce stade.

La facette matériel propose plusieurs structures de plate-forme (MP) (en terme de nombre de ressources de calcul, de liens de communication, etc.). Ces propositions sont guidées d'une part par l'expérience des intervenants de la facette et d'autre part, peut-être par certaines exigences via un processus proche de celui que nous proposons ici (par exemple des contraintes de tolérance aux fautes conduiront à la multiplication des ressources de calcul, tandis que des contraintes d'encombrement ou de consommation électrique tendront à en limiter le nombre).
Pour chacune des architectures proposées est établi un layer mapping ainsi que tous les layers liés à la facette matériel, en particulier le layer performances qui nous sert ici de facteur dimensionnant. La facette matériel peut alors, pour chaque configuration, considérer les contraintes des layers comme des (in)équations à résoudre où les variables correspondant à la facette logiciel sont des constantes. Pour chaque architecture, elle obtient un ensemble de valeurs acceptables (solution des (in)équations) qui lui permettront de dimensionner ses objets (le cas échéant de choisir ses composants sur étagère). Notre approche en la matière est de considérer que réaliser un (sous-)système revient à éliminer le plus de configurations possibles avant d'opérer *in fine* un choix arbitraire entre celles qui sont équivalentes en regard des critères de choix initiaux (liés aux exigences).

8.2 Dimensionnement de la plate-forme : calcul effectif

Les étapes précédentes permettent de déterminer un ensemble de types d'architectures et de mapping plausibles sur ces architectures. Nous allons déterminer le dimensionnement de deux des plate-formes possibles.
Pour ce faire nous nous fonderons sur quelques choix technologiques *a priori* :
– Les politiques d'ordonnancement implantées sur les ressources de calcul sont de type *Time Slicing* à tranche de temps fixe sans priorité ;
– Les fonctions sont affectées à une ressource de calcul et une seule. Une fonction n'utilise donc pas deux ressources de calcul pour s'exécuter, et ne change pas de ressource au cours du temps.

Les figures 8.1 et 8.2 page suivante présentent les deux architectures proposées par la facette matériel. Dans cet exemple nous ne cherchons qu'à dimensionner les ressources de calcul de ce fait, les figures ne représentent que celles-ci et leur environnement immédiat (voir fig 4.11 page 62 pour le reste de l'architecture). La première est basée sur deux ressources de calcul redondantes exécutant chacune l'intégralité des logiciels, l'autre propose de répartir les traitements sur deux paires de calculateurs redondants. Étant donnée l'extrême symétrie induite par la redondance des ressources de calcul (imposée par des contraintes de tolérance aux pannes) nous ne traiterons, dans chaque cas qu'une seule des ressources d'une paire redondante.
Il faut maintenant déterminer le mapping associé à chaque architecture. Il peut, bien entendu en exister plusieurs, dans ce cas il faut procéder à une étude de dimensionnement pour chacun d'entre eux.

Pour l'architecture de la figure 8.1 page suivante les mappings possibles sont équivalents, puisque les deux calculateurs exécutent l'ensemble des logiciels et sont redondants. Nous utiliserons donc celui de la figure C.2 page 137.

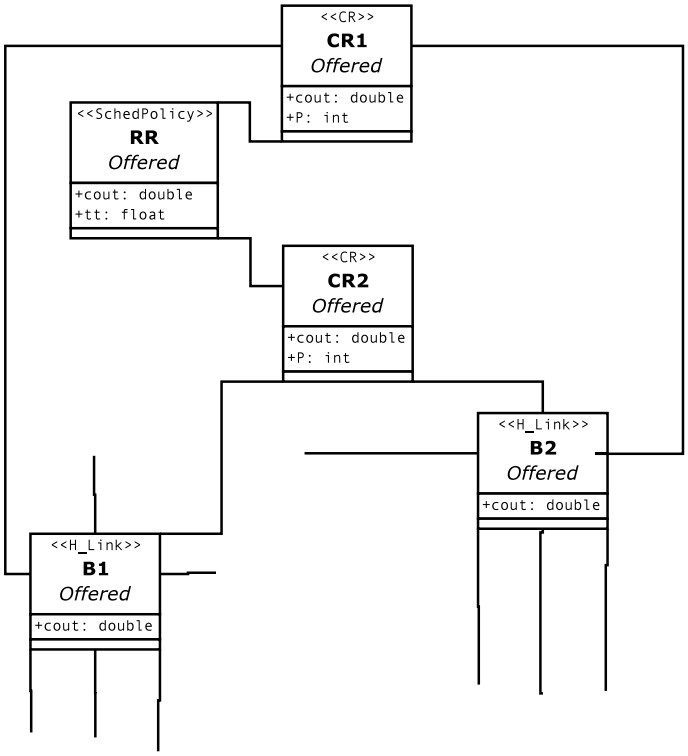


FIG. 8.1 – Proposition de Modèle Public à 2 ressources de calcul

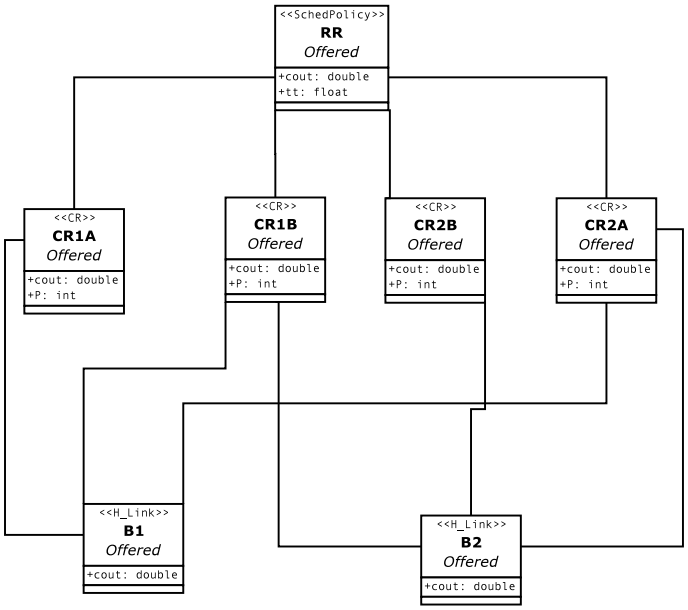


FIG. 8.2 – Proposition de Modèle Public à 4 ressources de calcul

En revanche, dans le cas de la figure 8.2 page précédente nous établirons le mapping en équilibrant la taille des traitements exécutés par chaque ressource.

8.2.1 Notations

Définissons les notations utilisées par la suite :

Définition 2 *Nous noterons c.F l'ensemble des fonctions s'exécutant sur la ressource de calcul c. Cette notation est un abus d'écriture puisqu'il n'existe pas d'association directe entre une ressource de calcul et les fonctions qu'elle exécute. En revanche c.F peut être construit en naviguant dans le layer mapping :*

$$c.F = c.mapFun.fonctions$$

Définition 3 *Le nombre de fonctions s'exécutant en parallèle sur une ressource de calcul c est noté c.#F (* NB : $c.\#F = Card(c.F)$*). Ce nombre est une donnée du mapping, par conséquent elle est disponible pour tous les layers.*

Définition 4 *Soit F une fonction, nous noterons T_F sa période.*

Définition 5 *Soit F une fonction, le nombre d'instructions maximal pour produire toutes ses données de sortie est noté $I_{max}(F)$*

Définition 6 *La durée des tranches de temps d'une ressource de calcul C_i est notée tt_i*

8.2.2 Méthodologie générale

La première étape consiste à déterminer la durée maximale de la tranche de temps pour chaque calculateur (il n'est pas question ici de garantir l'optimalité). Il faut que la tranche de temps soit suffisamment courte pour permettre à toutes les tâches de satisfaire leur périodicité :

Définition 7 *Soit $D_{MAX}(c.tt)$ la durée maximale de la tranche de temps acceptable pour la ressource de calcul c,*

$$D_{MAX}(c.tt) = \frac{\min\limits_{F_i \in \{c.F\}} (F_i.T)}{c.\#F}$$

La figure 6.5 page 82 donne le layer calculant le pire temps de réponse (R_{MAX}) des fonctions du pivot. Il nous faut dimensionner la plateforme de telle sorte que R_{MAX} soit inférieur à la période, soit en notant :

- $s \stackrel{\Delta}{=} this.layer.process$ le processus logiciel du layer ;
- $cr \stackrel{\Delta}{=} this.layer.cr$ la ressource de calcul impliquée ;
- $\#F \stackrel{\Delta}{=} this.layer.cr.mapFun.size()$ le nombre de fonctions mappées sur la ressource.

$$\left\lfloor \frac{F.WCET}{cr.schedPol.tt} \right\rfloor \times (cr.schedPol.HP - cr.schedPol.tt) + F.WCET < F.periode$$

En supposant que le WCET peut être calculé comme le nombre maximum d'instructions du processus divisé par la puissance de la ressource, l'équation à satisfaire devient :

$$\left\lfloor \frac{s.nbInstrMax}{cr.schedPol.tt * cr.P} \right\rfloor \times (cr.schedPol.HP - cr.schedPol.tt) + \frac{s.nbInstrMax}{cr.P} < F.periode$$

en s'appuyant sur le fait que $\lfloor x \rfloor <= x$ et en développant, l'inéquation devient :

$$cr.P > \frac{s.nbInstrMax \times cr.schedPol.HP}{F.periode \times cr.schedPol.tt}$$

La tranche de temps *tt* est une donnée de la ressource de calcul (plus exactement de sa politique d'ordonnancement).

8.2.3 Architecture à deux ressources de calcul

Calcul de la tranche de temps, en secondes :

$$D_{MAX}(tt_i) = \frac{\min\limits_{F \in \{F_i\}} (T_F)}{\#F_i}$$

$$D_{MAX}(tt_i) = \frac{25.10^{-3}}{4} = 6,25.10^{-3}$$

Nous choisissons $tt = 5.10^{-3}$s.
Soit P_1 la puissance de $C1$. P_1 doit satisfaire le système suivant :

$$\left\{ \begin{array}{l|lll} (SuiviMission) & c1.P & > & \frac{10^6 \times 25.10^{-3}}{100.10^{-3} \times 5.10^{-3}} = 50.10^6 \\ (TraitementCapteur) & c1.P & > & \frac{5.10^5 \times 25.10^{-3}}{50.10^{-3} \times 5.10^{-3}} = 50.10^6 \\ (Navigation) & c1.P & > & \frac{4.10^5 \times 25.10^{-3}}{50.10^{-3} \times 5.10^{-3}} = 40.10^6 \\ (LoiCommande) & c1.P & > & \frac{2,2.10^5 \times 25.10^{-3}}{25.10^{-3} \times 5.10^{-3}} = 44.10^6 \end{array} \right.$$

Il faut donc choisir un processeur d'une puissance supérieure à 50 MIPS (Millions d'Instructions Par Seconde). Le calcul désigne aussi les processus qui reviennent le plus cher, il s'agit ici de `SuiviMission` et `TraitementCapteur`.

8.2.4 Architecture à quatre ressources de calcul

Nous allons maintenant dimensionner les ressources de calcul primaires (CR1A et CR2A) de l'architecture décrite figure 8.2 page 111.
Le calcul de la tranche de temps, en secondes utilise la formule :

$$D_{MAX}(tt_i) = \frac{\min\limits_{F \in \{F_i\}} (T_F)}{\#F_i}$$

$$D_{MAX}(tt_1) = \frac{25.10^{-3}}{2} = 12,5.10^{-3}$$

Nous choisissons $tt = 12,5.10^{-3}$s.

$$D_{MAX}(tt_2) = \frac{50.10^{-3}}{2} = 25.10^{-3}$$

Nous choisissons $tt = 25.10^{-3}$s.
Soit P_1 la puissance de $CR1A$. P_1 doit satisfaire le système suivant :

$$\left\{ \begin{array}{l|lll} (SuiviMission) & P_1 & > & \frac{10^6 \times 25.10^{-3}}{100.10^{-3} \times 12,5.10^{-3}} = 20.10^6 \\ (LoiCommande) & P_1 & > & \frac{2,2.10^5 \times 25.10^{-3}}{25.10^{-3} \times 12,5.10^{-3}} = 17,6.10^6 \end{array} \right.$$

Il faudra choisir pour $CR1A$ une ressource de calcul d'une puissance d'au moins 20MIPS. Le processus dominant étant *SuiviMission*.

Soit P_2 la puissance de $CR2A$. P_2 doit satisfaire le système suivant :

$$\left\{ \begin{array}{l|lll} (TraitementCapteur) & P_2 & > & \frac{5.10^5 \times 50.10^{-3}}{50.10^{-3} \times 25.10^{-3}} = 20.10^6 \\ (Navigation) & P_2 & > & \frac{4.10^5 \times 50.10^{-3}}{50.10^{-3} \times 25.10^{-3}} = 16.10^6 \end{array} \right.$$

Il faudra choisir pour $CR2A$ une ressource de calcul d'une puissance d'au moins 20MIPS. Le processus dominant étant $TraitementCapteur$.

8.3 Conclusion de la troisième partie

Cette partie avait pour but de placer nos structures dans un contexte de développement. Le processus de développement "ex nihilo" fournit les grandes lignes de leur utilisation en vue d'un développement complet d'un système, et peut (doit) être adapté aux processus existant dans les entreprises. Nous avons pu présenter ce processus à des spécialistes de l'ingénierie système lors du *Congrès Francophone du Management de Projet 2004* [1] (cf. [TMR+04]). Nous avons ensuite montré une propriété intéressante du concept de layer. L'utilisation des layers telle qu'évoquée jusqu'ici consiste à renseigner le pivot grâce aux informations extraites des MPs de facettes déjà construites (ou en cours de construction, mais dont la structure est figée). En revanche, si nous cherchons à créer une facette qui, associée avec une autre, préexistante, doit former un système conforme à un cahier des charges donné, il faut alors créer un pivot "final" cohérent avec la facette existante et parfaitement conforme au cahier des charges. Les layers peuvent alors être vus comme des équations à résoudre dont les solutions seront des modèles possibles du MP de la facette recherchée...

[1] consacré à l'intégration des projets

Quatrième partie

Conclusion

Chapitre 9

Synthèse et conclusion

9.1 Synthèse

Au cours de cette thèse, nous nous sommes intéressés à la problématique engendrée par la coopération de multiples intervenants dans la modélisation de systèmes embarqués. Notre démarche a été d'étudier ce thème du point de vue le plus large possible, afin d'identifier les problèmes qui se posent et de fournir une méthodologie générale permettant d'y répondre partiellement. Nous avons cherché à montrer que la multiplication des intervenants spécialistes de différents métiers, si elle peut être, de prime abord, ressentie comme une contrainte supplémentaire au problème déjà épineux de la modélisation des systèmes, peut au contraire lui apporter des éléments de solution naturels. En effet, elle fournit, non seulement, un axe de séparation strict entre les divers éléments de modèle, mais aussi elle permet de définir un ensemble restreint d'informations partagées entre les différents modèles.
Nous avons donc dans un premier temps cherché à établir les modèles permettant aux différents intervenants de communiquer entre eux. Ceci impliquait d'établir d'une part les informations à partager et celles que chacun pouvait et devait cacher, et d'autre part la structure des modèles adaptés à chaque intervenant. De cette réflexion, il ressort deux grands types de méta-modèles :

1. Le pivot est un modèle spécifiquement destiné au MOE. Il fournit une vision "gros grain" mais complète du système. Il permet en outre d'exprimer les exigences globales ;
2. Les MPs sont les interfaces des facettes métier. Bien qu'ils soient spécifiques à chaque facette d'un même système, ils partagent la même philosophie. Ils expriment les informations portées et mises à disposition par la facette. Ce sont eux qui fournissent les données brutes permettant de renseigner le pivot sur les choix de conception des facettes.

Une fois identifiés les éléments à la charge des différentes facettes, il était fondamental d'exprimer les exigences transverses (non attribuables à telle ou telle facette) pour pouvoir, d'une part assurer la correction du système au regard de son cahier des charges, mais aussi d'assurer la cohérence des choix d'implantations des facettes entre elles. Nous avons dans ce but introduit la notion de layer comme une famille de modèle portant les informations transverses.
La description statique de ces modèles fait l'objet de la première partie de cette thèse.
La seconde partie est elle, dédiée à leur exploitation dans le cadre d'un processus de développement. Elle représente une part importante de nos travaux dans la mesure où nous nous plaçons dans le domaine de l'aide à la maîtrise d'œuvre. Dans ce cadre il est impératif de s'intégrer dans un processus de développement. De plus la notion de layer est prévue pour permettre, au plus tôt la

détection d'incohérences entre les facettes ou avec les exigences transverses. Nous avons donc présenté dans cette deuxième partie deux cas d'utilisation différents de nos structures :

1. Premièrement, nous nous plaçons dans le cadre idéal du développement *complet* d'un système à partir de ses spécifications. Nous introduisons un processus en double spirale tirant partie des spécificités du découpage des modèles que nous proposons ;
2. Nous montrons ensuite comment nos structures permettent la réutilisation de l'existant sur un exemple (naïf) de dimensionnement de plate-forme.

9.2 Perspectives

9.2.1 Pivots récursifs

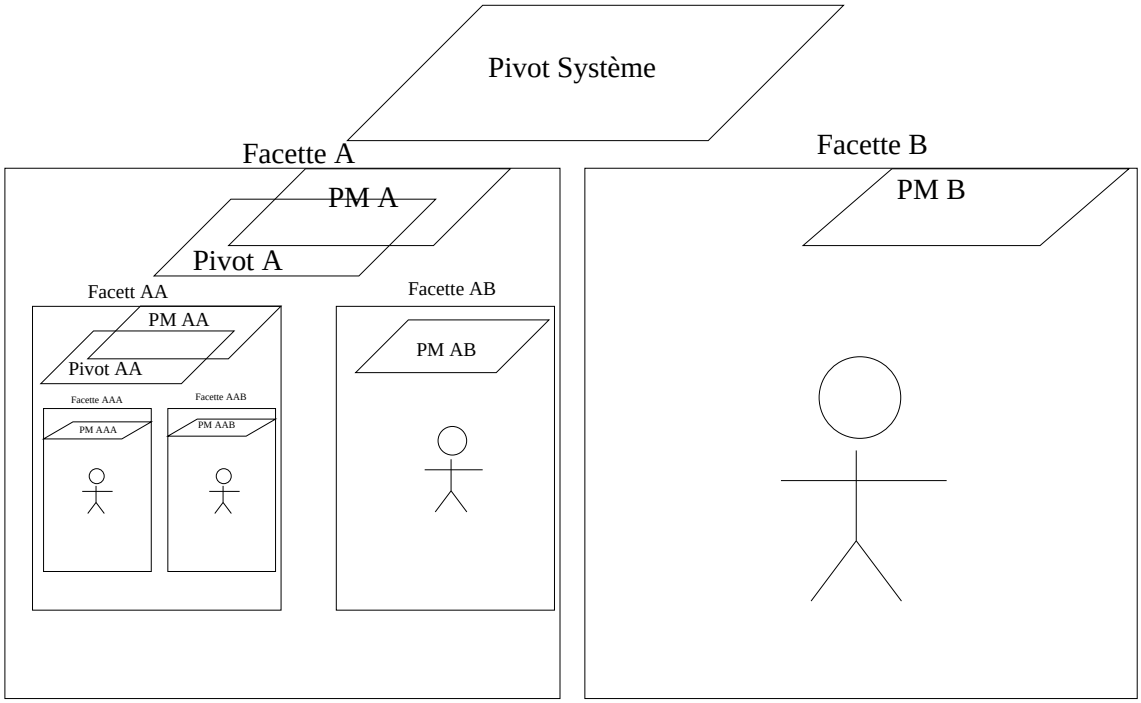
Dans le cadre de cette thèse nous avons pris comme hypothèse implicite que chaque facette métier est monolithique. Bien entendu, lors du développement de systèmes complexes, chacune d'elles peut constituer un système à part entière et faire appel à plusieurs (sous-)facettes. Dans ce cas, nous proposons d'itérer la méthode jusqu'à ce que les facettes puissent être considérées comme étant atomiques. À chaque itération le pivot du sous-système constitué par une facette devient le Modèle Public de la facette pour le système d'ordre supérieur, au prix de transformations de modèles vraisemblablement mineures. La figure 9.1 page suivante illustre ce fonctionnement.

9.2.2 Élargissement du domaine d'étude

Nous nous sommes restreints à l'étude des systèmes embarqués et, dans ce cadre, à deux uniques facettes aux propriétés d'orthogonalité intéressantes. Nous pouvons élargir ce domaine d'étude de deux façons. Premièrement, par l'ajout de nouvelles facettes tel qu'évoqué chapitre 4.3.4 page 63. Le prix à payer sera probablement une relative expansion de la taille du pivot et du nombre de layers. Nous pensons néanmoins que ce phénomène sera limité et que la complexité finalement atteinte reflètera la complexité inhérente au système considéré.
Nous pourrions aussi élargir cette étude au cas de classes de systèmes plus larges que celles étudiées ici. Bien que nous ne nous soyions pas réellement penchés sur le sujet, nous pensons que la suppression de contraintes spécifiques au domaine de l'embarqué (particulièrement celles liées à des questions de criticité) risque de complexifier les interactions entre facettes et d'augmenter la taille des différents modèles d'un ordre de grandeur exponentiel des degrés de liberté. L'application au dimensionnement de plate-forme présenté au chapitre 8 page 109 donne l'intuition d'un tel comportement. En effet, seules de fortes contraintes préalables permettent de limiter le nombre d'architectures et mappings envisageables. De telles contraintes (redondance, encombrement, masse, etc), se feront de plus en plus rares en tentant de généraliser la méthode.

9.2.3 Poursuites éventuelles des travaux

Le choix d'effectuer une étude "en largeur" du sujet implique que les solutions que nous proposons restent au stade de guides méthodologiques, ou d'indices. La plupart des hypothèses simplificatrices sur lesquelles nous nous sommes basés le sont drastiquement. Les suites immédiates à donner à ce travail sont de poursuivre l'étude des nombreuses parties non approfondies. Par exemple, il nous



Intervenant atomique (monolithique)

FIG. 9.1 – Pivots récursifs

semble important de détailler le layer performance temps réel, en y intégrant le calcul des gigues afin de prouver (ou d'invalider) cette approche de modélisation des propriétés temps réel. Une autre part importante restant à effectuer est l'étude de l'automatisation de nos méthodes par des techniques de transformations de modèles. Ce travail fait actuellement l'objet d'une thèse au laboratoire DTN de l'ENSIETA.
Nos travaux ont aussi pour vocation de poser les bases d'une chaîne d'outils logiciels d'aide à la maîtrise d'œuvre dans le domaine des systèmes embarqués. L'implantation de ces outils reste à réaliser et sort du cadre de cette thèse.

Annexe A

Comportements des processus de la facette logiciel

La présente annexe fournit les automates des processus du MP de la facette logiciel.

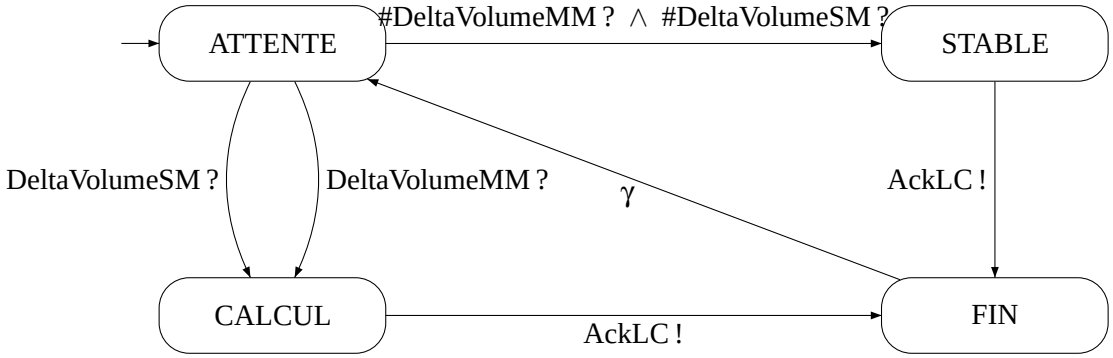


FIG. A.1 – ALC_SEC : Automate du processus LoiCommande

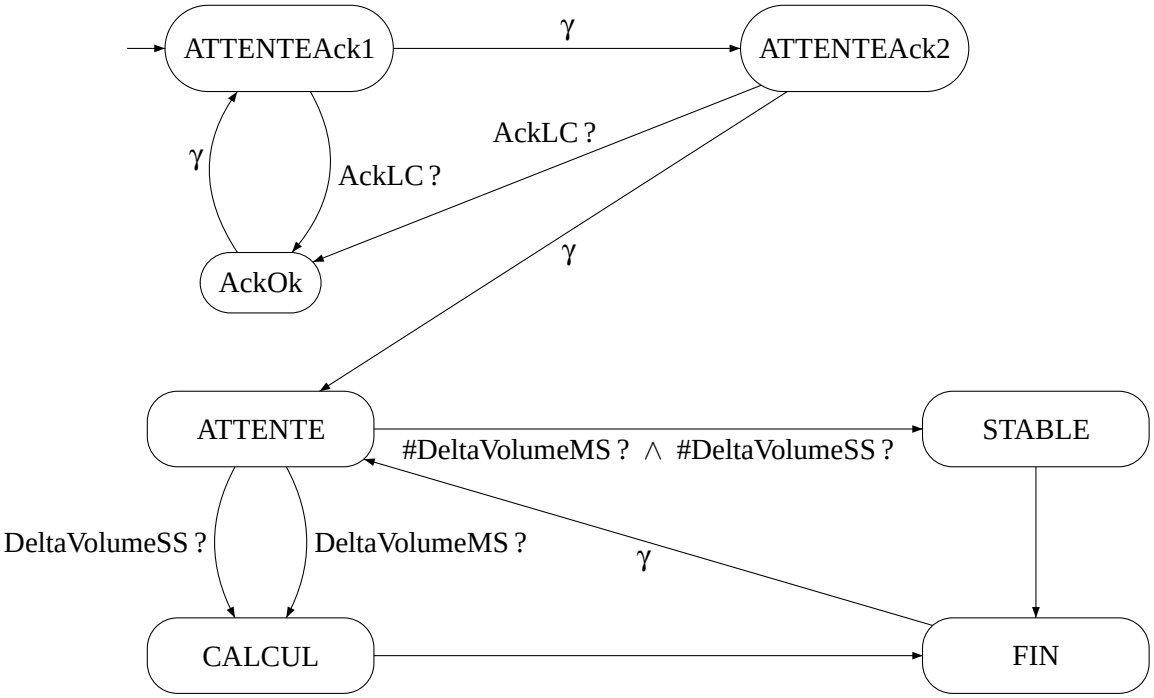


FIG. A.2 – ALC_SEC : Automate du processus LoiCommandeSec

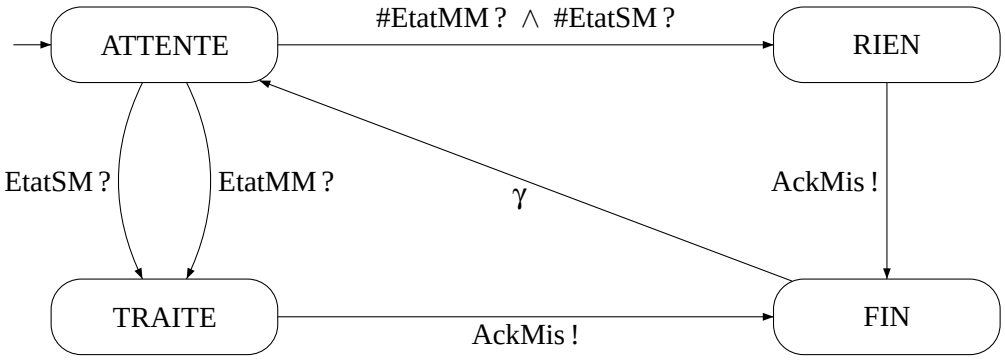


FIG. A.3 – ASM : Automate du processus SuiviMission

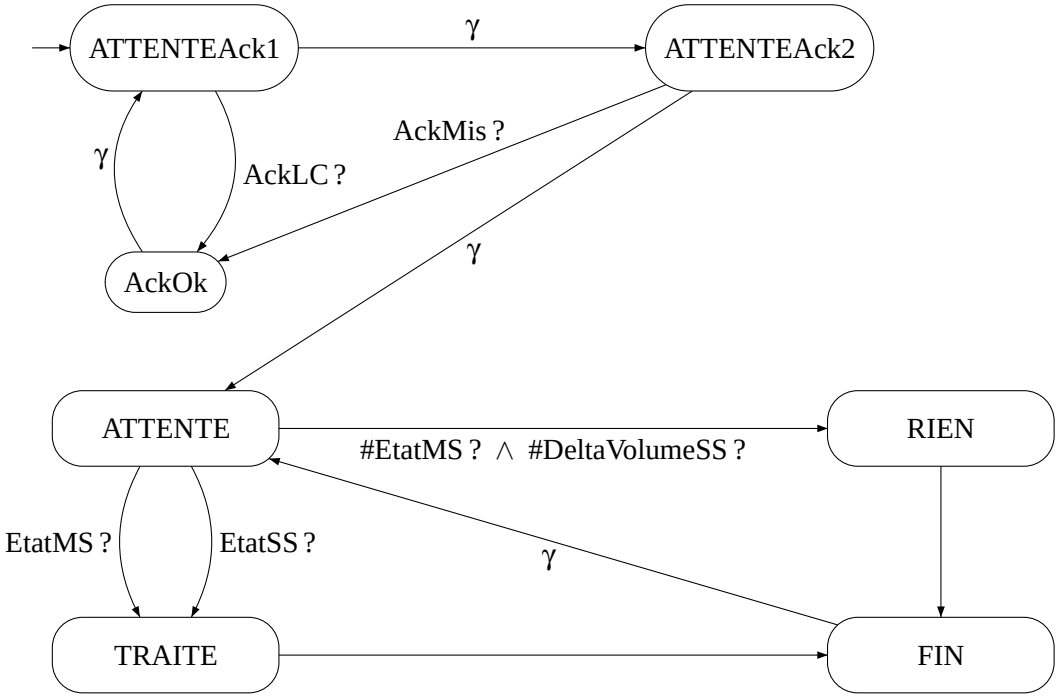


FIG. A.4 – ASM_SEC : Automate du processus SuiviMission de secours

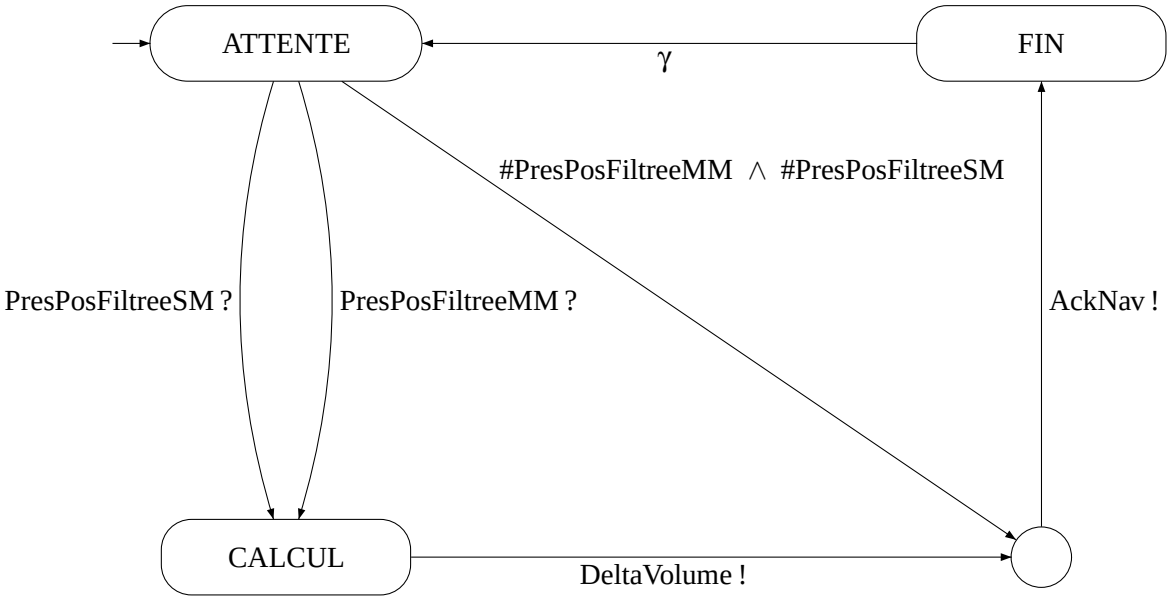


FIG. A.5 – ANAV_SEC : Automate du processus Navigation

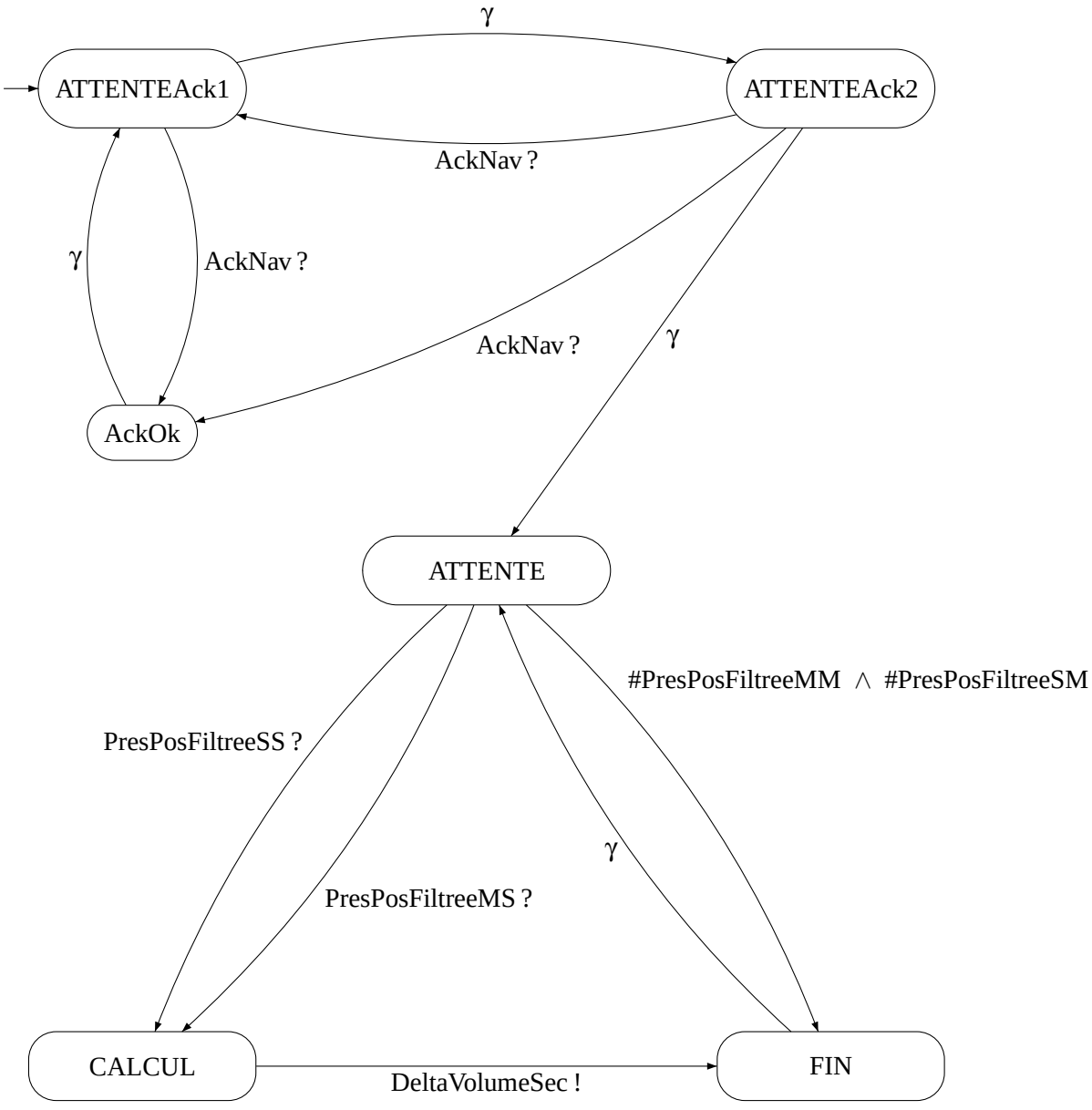


FIG. A.6 – ANAV_SEC : Automate du processus Navigation de secours

Annexe B

Pivot

Nous présentons ici les différents aspects du pivot que nous n'avons pas souhaité fournir dans le corps du document pour en alleger la lecture.

B.1 Sémantique opérationnelle

Dans cette section, nous définissons la sémantique opérationnelle du pivot sous la forme d'un système de transitions étiqueté et temporisé. La relation de transition est définie par induction structurelle au moyen de règles de la forme :

$$\frac{\text{prémisses}}{\text{conclusion}}$$

pour lesquelles la conclusion est déduite des prémisses.

Notations

On note X l'horloge globale dont le zéro correspond au lancement du système. Cette horloge n'a pas à exister physiquement dans le système.
Pour toute `S_Data` a fournie par le MP de la facette logiciel nous noterons a^p la `P_Data` qui lui est associée par le mapping (*cf* 6.3 page 78).

Fonctions

On note :

$$TF := (X - \delta) mod\ T$$

TF est le temps passé par la fonction considérée dans l'activation en cours.
Soient $I_1 = [a,b]$ et $I_2 = [c,d]$ on note :

$$I_1 + I_2 := [a+c, b+d]$$

$$Sup(I_1) = b$$

$$Inf(I_1) = a$$

FR1 Si l'automate de la fonction peut émettre ses sorties $(a_1, a_2, \ldots, a_n)$ et que la fonction est dans sa phase d'emmission alors elle émet instantanéement les données pivot correspondant aux sorties de l'automate $(a_1^p, a_2^p, \ldots, a_n^p)$

$$\frac{RMin \leq TF \leq F.RMax \wedge A \xrightarrow{\{a_1!,a_2!,\ldots,a_n!\}} A'}{F \xrightarrow{\{a_1^p!,a_2^p!,\ldots,a_n^p!\}} F_{[A/A']}}$$

FR2 Une fonction peut laisser s'écouler le temps tant qu'il n'atteint pas la fin de la phase d'émission S.

$$\frac{\forall t < RMax - TF \mid RMax - TF < t \leq T}{F \xrightarrow{t} F}$$

FR3 Si lors de la lecture des entrées de A la donnée a est présente, F reçoit a^P

$$\frac{TF = 0 \wedge A \xrightarrow{a?} A'}{F \xrightarrow{a_p?} F_{[A/A']}}$$

FR4 Si lors de la lecture des entrées de A la donnée a est présente et que ce cas est spécifié par l'automate, F tire une transision de défaut de a ($\#a?$)

$$\frac{TF \in E \wedge A \xrightarrow{\#a?} A'}{F \xrightarrow{\#a?} F_{[A/A']}}$$

FR5 Au début de chaque période la fonction tire une transition blanche si son automate peut tirer une transition de début de période (γ).

$$\frac{TF = 0 \wedge A \xrightarrow{\gamma} A'}{F \xrightarrow{\varepsilon} F_{[A/A']}}$$

Liens de communication

CR1 Lors de la réception d'une donnée, un délais aléatoire pris dans la gigue de sortie de la donnée lui est affecté.

$$\frac{a \in I \wedge a.LCMind \leq a.LCMax}{C \xrightarrow{a?} C[B/B \cup \{(a,d)\}]}$$

CR2 passage du temps : un lien peut laisser passer un temps inférieur au minimum des délais strictement positifs (les messages déjà disponibles voient leur délais (bien que déjà négatifs) diminuer mais n'influent pas sur le temps pouvant s'écouler)

$$\frac{\forall (a,d) \in B | d > 0 \wedge \forall t \leq d}{C \xrightarrow{t} C_{[B/(m,d-t)|(m,d)\in B]}}$$

CR3 émission : si un message en attente a un délais nul il peut être émis (rendu disponible).

$$\frac{\exists (a,0) \in B}{C \xrightarrow{a!} C_{[B/B-\{(a,0)\}]}}$$

Exécution parallèle de deux fonctions : sémantique de l'opérateur $\|_F$

$F_1 \|_F F_2$:

FFR1 Si F_1 attend une donnée a et devient F_1' alors $F_1 \|_F F_2$ attend a et devient $F_1' \|_F F_2$.

$$\frac{F_1 \xrightarrow{a?} F_1'}{F_1 \|_F F_2 \xrightarrow{a?} F_1' \|_F F_2}$$

FFR2 Si F_2 attend une donnée a et devient F_2' alors $F_1 \|_F F_2$ attend a et devient $F_1 \|_F F_2'$.

$$\frac{F_2 \xrightarrow{a?} F_2'}{F_1 \|_F F_2 \xrightarrow{a?} F_1 \|_F F_2'}$$

FFR4 Si F_1 émet une donnée a et devient F_1' alors $F_1 \|_F F_2$ émet a et devient $F_1' \|_F F_2$.

$$\frac{F_1 \xrightarrow{a!} F_1'}{F_1 \|_F F_2 \xrightarrow{a!} F_1' \|_F F_2}$$

FFR5 Si F_2 émet une donnée a et devient F_2' alors $F_1 \|_F F_2$ émet a et devient $F_1 \|_F F_2'$.

$$\frac{F_2 \xrightarrow{a!} F_2'}{F_1 \|_F F_2 \xrightarrow{a!} F_1 \|_F F_2'}$$

FFR6 si les deux fonctions laissent passer un même temps t alors leur exécution parallèle laisse aussi passer t unités de temps.

$$\frac{F_1 \xrightarrow{t} F_1' \wedge F_2 \xrightarrow{t} F_2'}{F_1 \|_F F_2 \xrightarrow{t} F_1' \|_F F_2'}$$

Remarque : les règles **FFR1** à **FFR5** décrivent l'entrelacement.

Fonctionnement parallèle de deux liens de communication : sémantique de l'opérateur $\|_C$

$C_1 \|_C C_2$:

CCR1 Si C_1 attend une donnée a et devient C_1' alors $C_1 \|_C C_2$ attend a et devient $C_1' \|_C C_2$.

$$\frac{C_1 \xrightarrow{a?} C_1'}{C_1 \|_C C_2 \xrightarrow{a?} C_1' \|_C C_2}$$

CCR2 Si C_2 attend une donnée a et devient C_2' alors $C_1 \|_C C_2$ attend a et devient $C_1 \|_C C_2'$.

$$\frac{C_2 \xrightarrow{a?} C_2'}{C_1 \|_C C_2 \xrightarrow{a?} C_1 \|_C C_2'}$$

CCR4 Si C_1 émet une donnée a et devient C_1' alors $C_1 \|_C C_2$ émet a et devient $C_1' \|_C C_2$.

$$\frac{C_1 \xrightarrow{a!} C_1'}{C_1 \|_C C_2 \xrightarrow{a!} C_1' \|_C C_2}$$

CCR5 Si C_1 émet une donnée a et devient C_1' alors $C_1 \|_C C_2$ émet a et devient $C_1' \|_C C_2$.

$$\frac{C_2 \xrightarrow{a!} C_2'}{C_1 \|_C C_2 \xrightarrow{a!} C_1 \|_C C_2'}$$

CCR6 si les deux liens laissent passer un même temps t alors leur exécution parallèle laisse aussi passer t unités de temps.

$$\frac{C_1 \xrightarrow{t} C_1' \wedge C_2 \xrightarrow{t} C_2'}{C_1 \|_C C_2 \xrightarrow{t} C_1' \|_C C_2'}$$

Exécution parallèle d'une fonction et d'un lien : sémantique de l'opérateur $\|_S$

SR1 si la fonction émet une donnée et que le lien peut la recevoir, alors l'exécution parallèle se transforme conformément au comportement du lien et de la fonction.

$$\frac{F \xrightarrow{a!} F' \wedge C \xrightarrow{a?} C'}{F \|_S C \xrightarrow{\varepsilon} F' \|_S C'}$$

SR2 si le lien émet une donnée et que la fonction peut la recevoir, alors l'exécution parallèle se transforme conformément au comportement du lien et de la fonction.

$$\frac{F \xrightarrow{a?} F' \wedge C \xrightarrow{a!} C'}{F \|_S C \xrightarrow{\varepsilon} F' \|_S C'}$$

SR3 si le lien n'émet pas la donnée a et que la fonction spécifie son comportement en cas de non reception de a, alors l'exécution parallèle se transforme conformément au comportement du lien et de la fonction.

$$\frac{F \xrightarrow{\#a?} F' \wedge C \xrightarrow{a!} \not\rightarrow C'}{F \|_S C \xrightarrow{\varepsilon} F' \|_S C'}$$

SR4 si la fonction et le lien laissent passer un même temps t alors leur exécution parallèle laisse aussi passer t unités de temps.

$$\frac{F \xrightarrow{t} F' \wedge C \xrightarrow{t} C'}{F \|_S C \xrightarrow{t} F' \|_S C'}$$

B.2 Etude de cas

Les figures suivantes représentent les parties du pivot de l'étude de cas que nous avons ommises lors de la présentation de celui-ci.

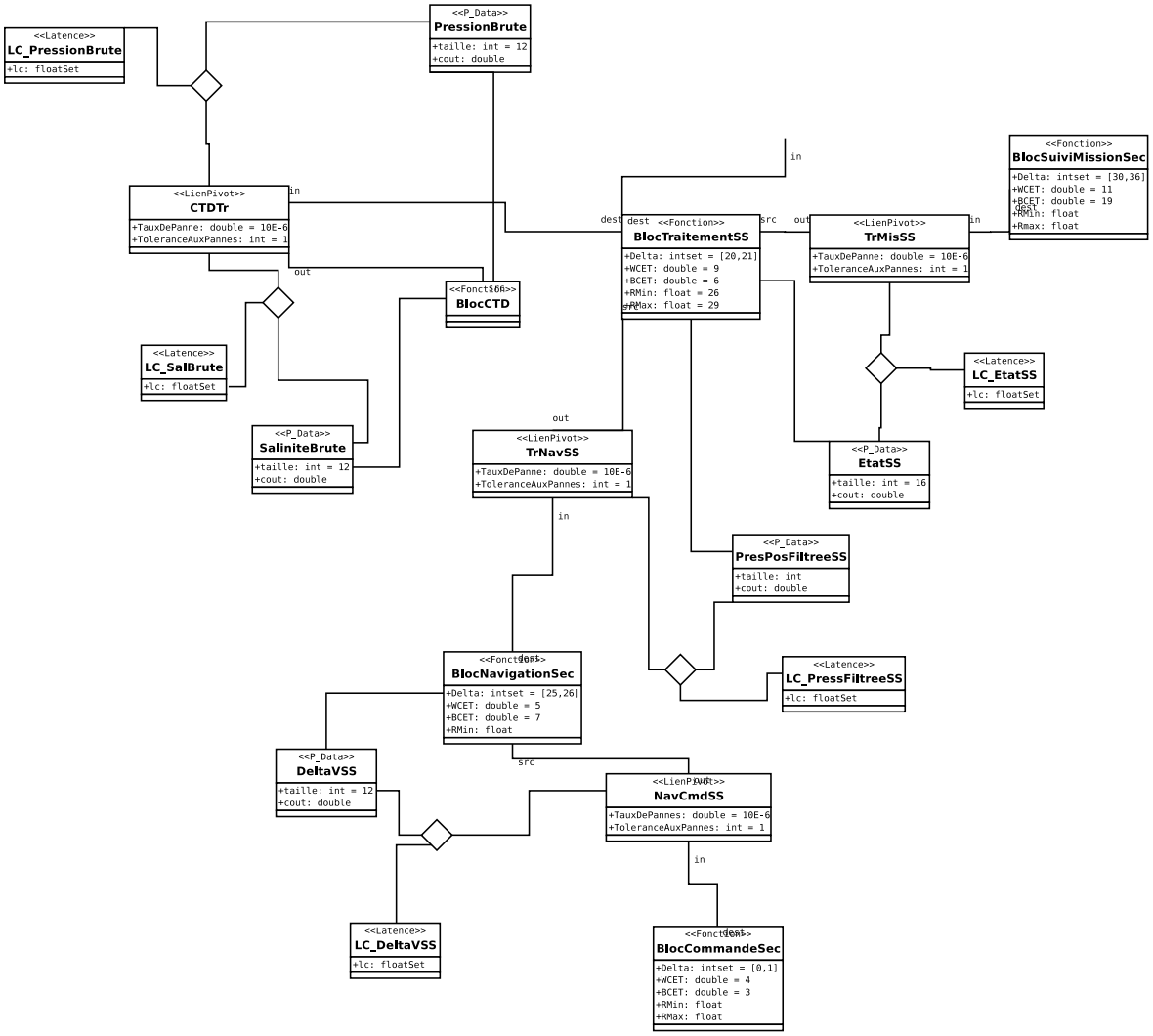


FIG. B.1 – Pivot de l'étude de cas secours vers secours

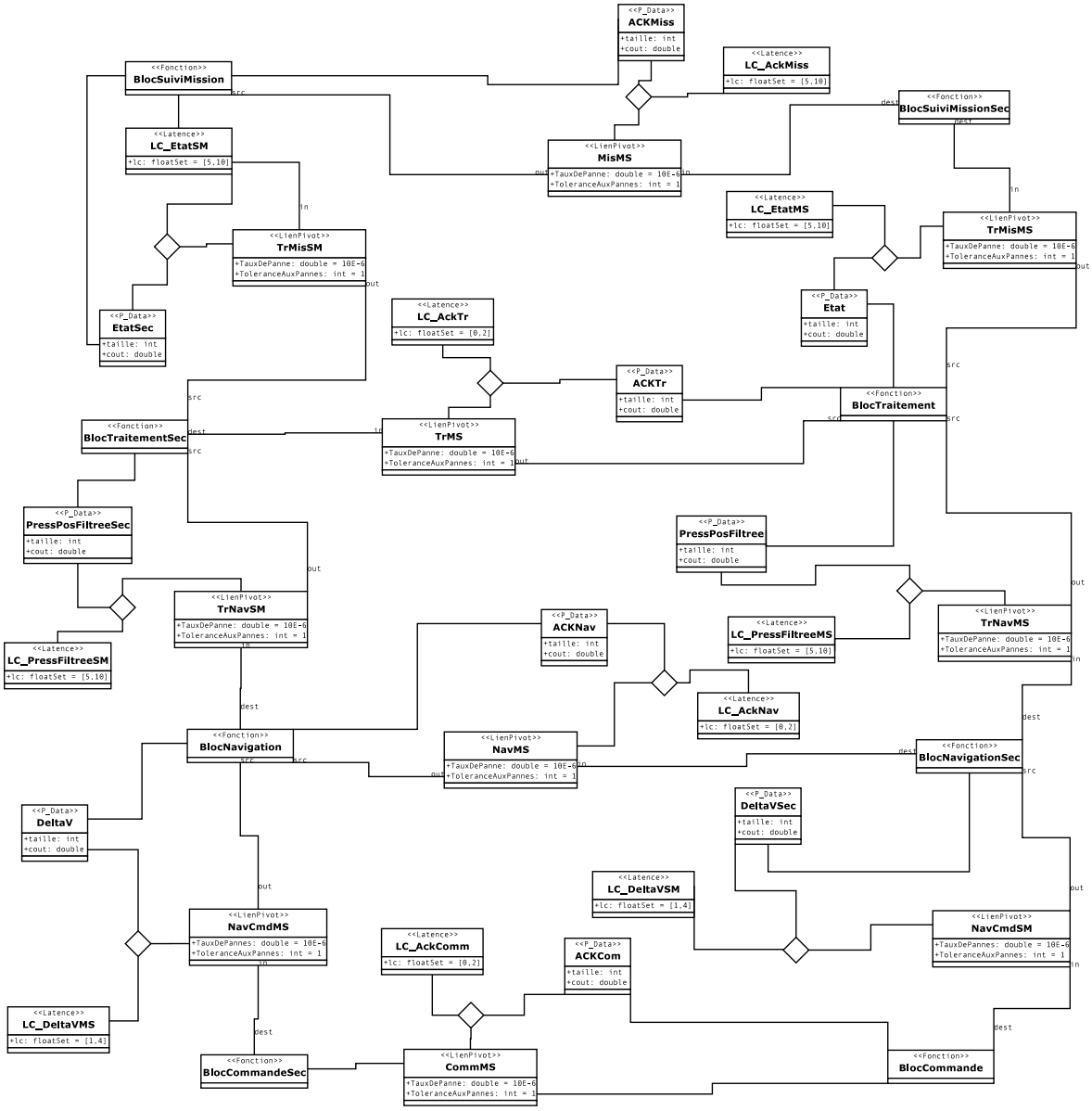


FIG. B.2 – Pivot de l'étude de cas Secours/Maitre, Maitre/Secours

Annexe C

Layers

C.1 Mapping de l’étude de cas

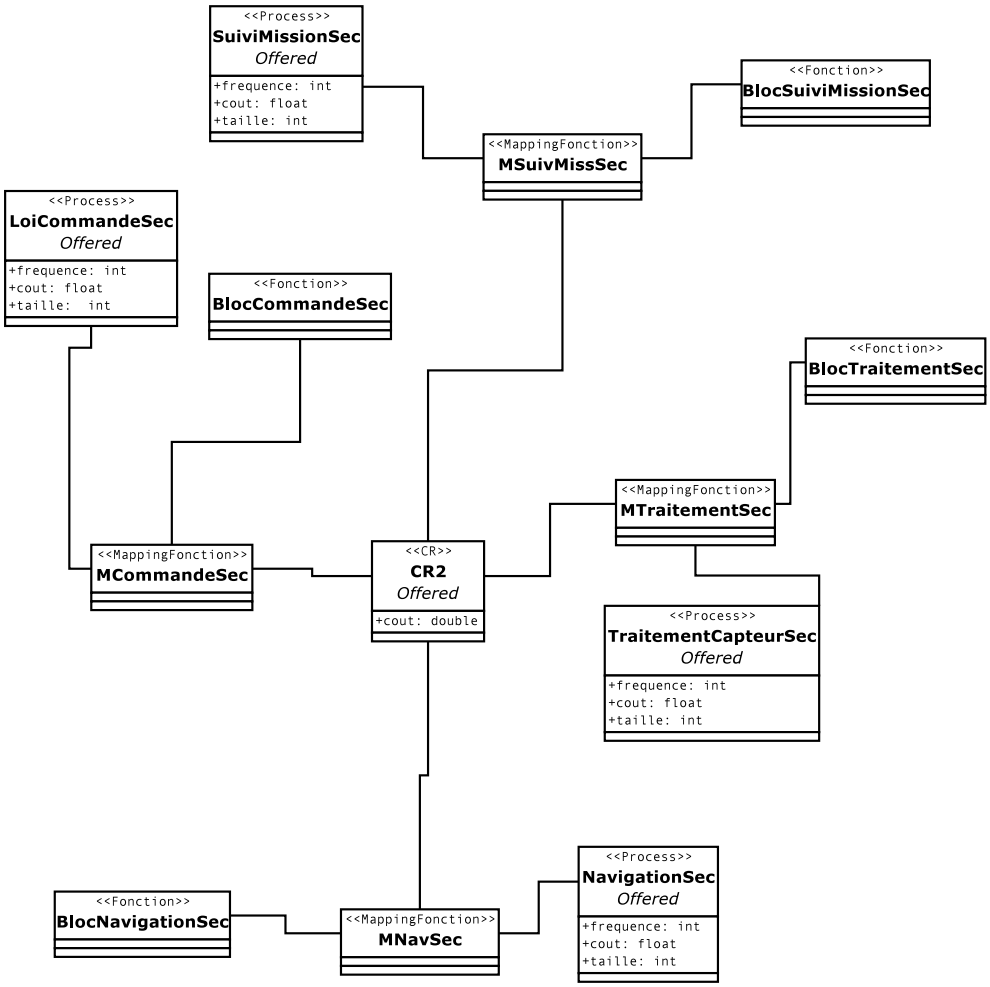


FIG. C.1 – Mapping du calculateur de secours

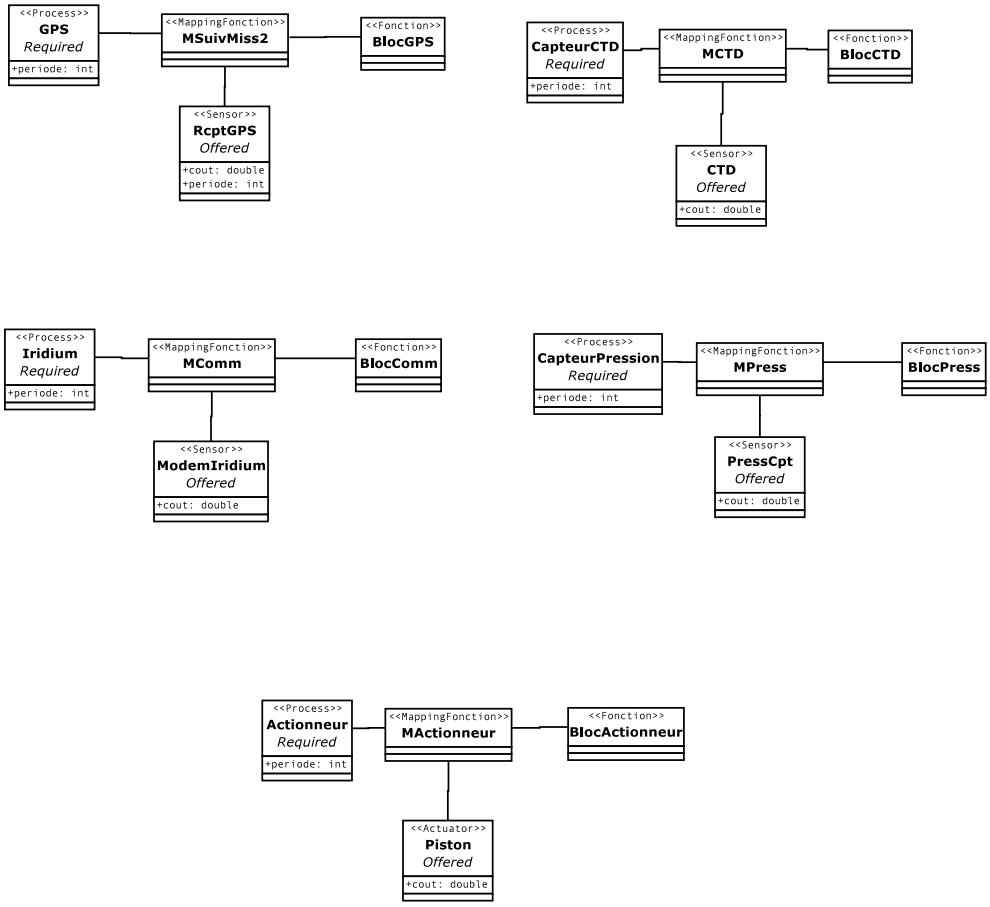


FIG. C.2 – Mapping des capteurs/actionneurs

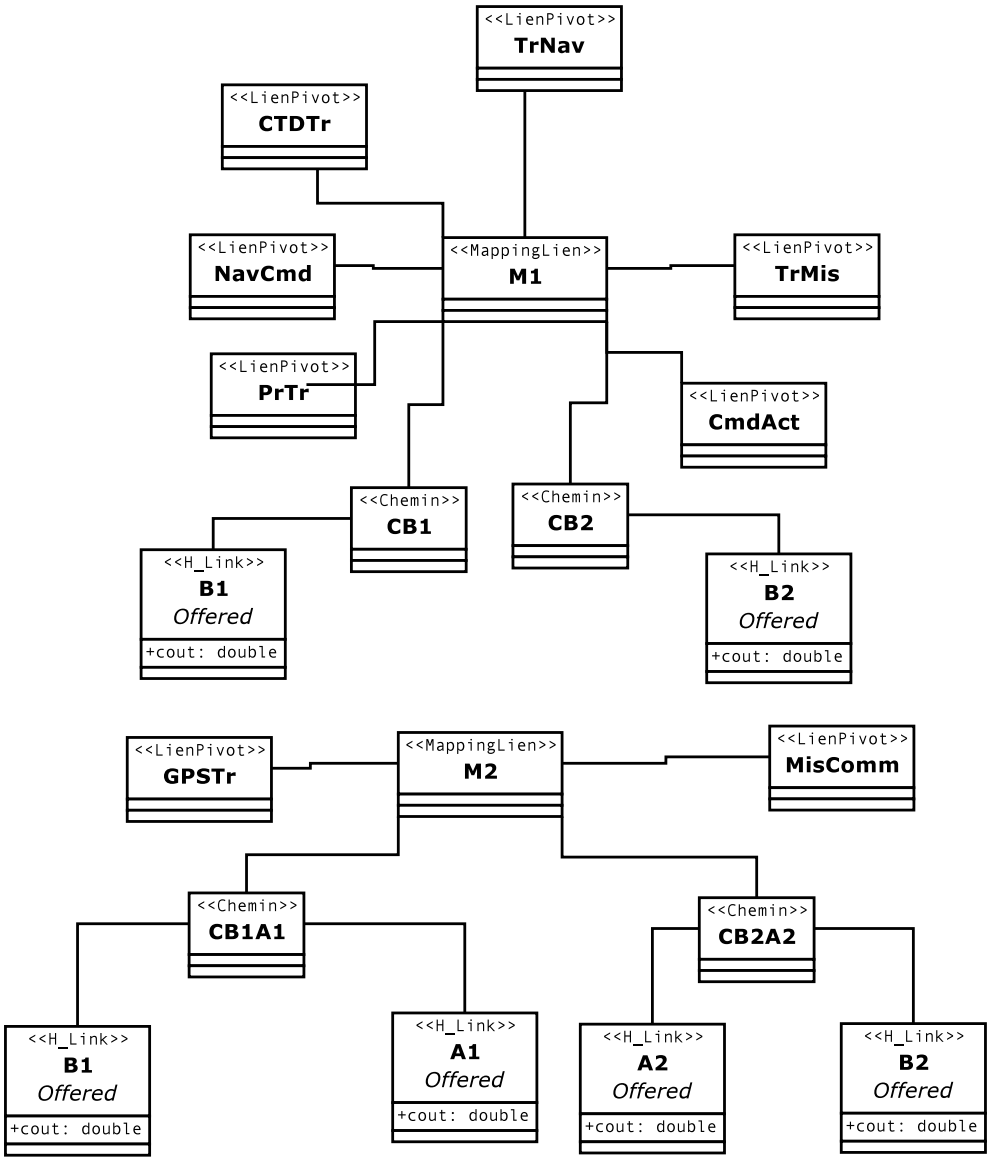


FIG. C.3 – Mapping des liens de communication

C.2 Automates Uppaal de l'étude de cas

C.2.1 Traduction des fonctions en mode nominal

Les figures suivantes fournissent les automates Uppaal des fonctions du pivot de l'étude de cas en l'absence de défaillance des ressources de calculs.

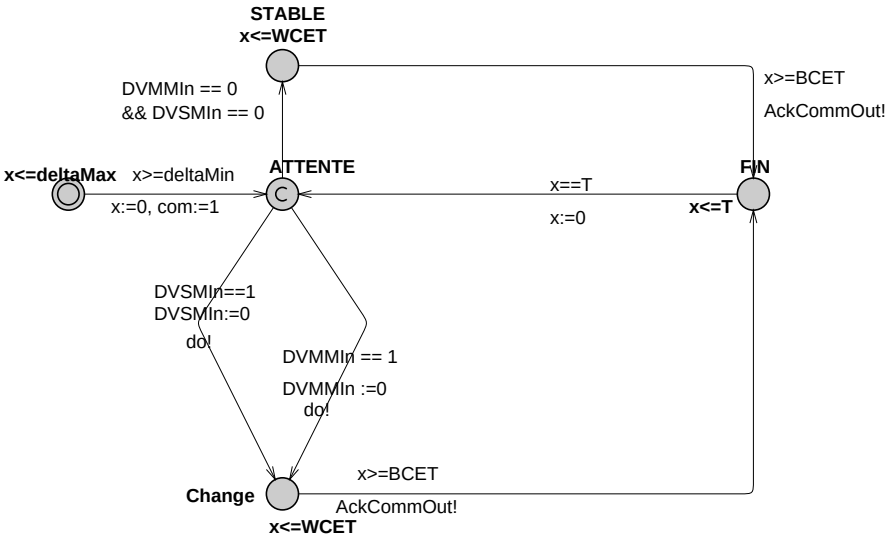


FIG. C.4 – Traduction de BlocCommande

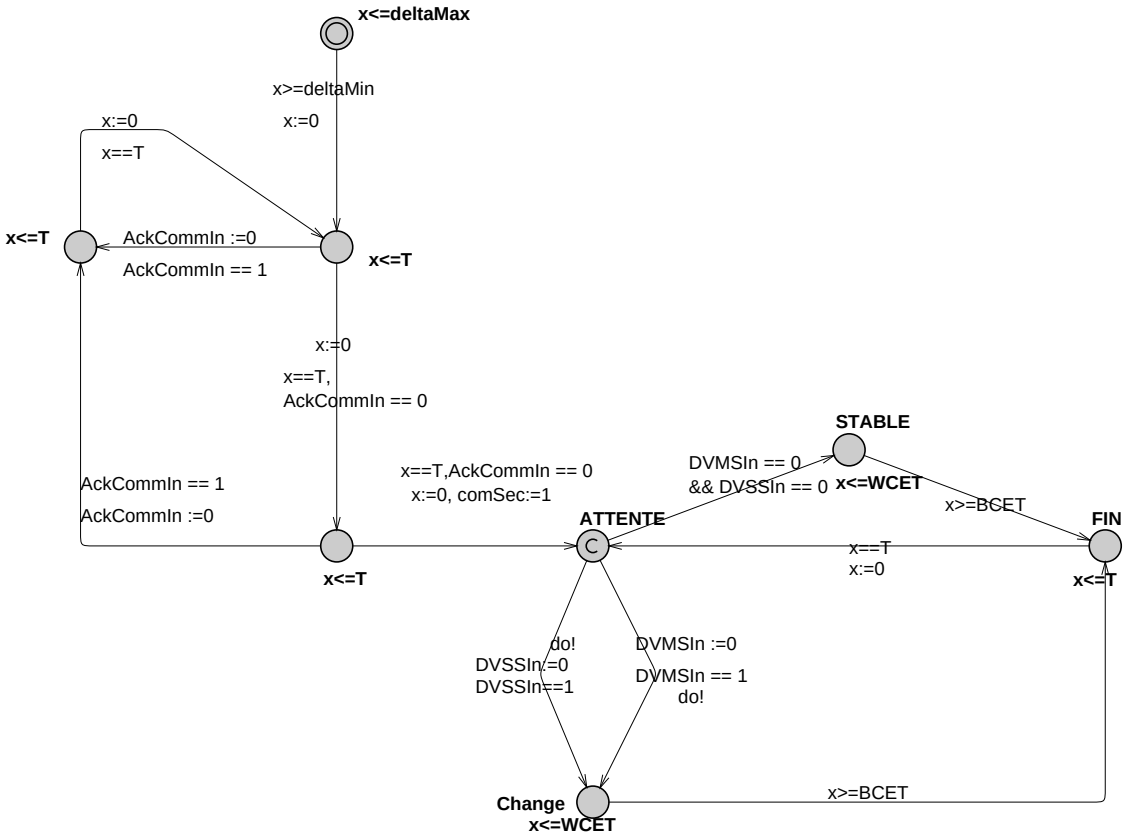


FIG. C.5 – Traduction de BlocCommandeSec

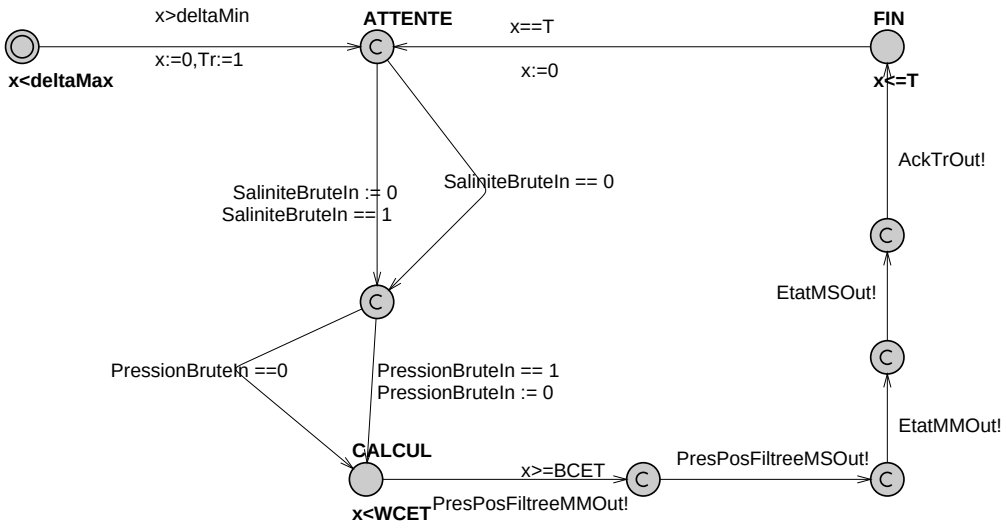


FIG. C.6 – Traduction de BlocTraitement

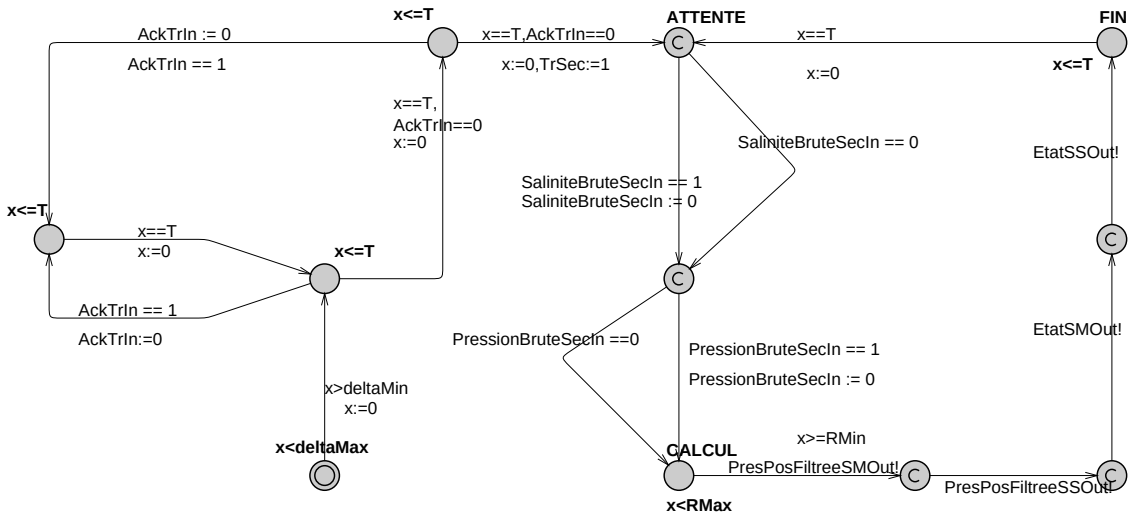


FIG. C.7 – Traduction de BlocTraitementSec

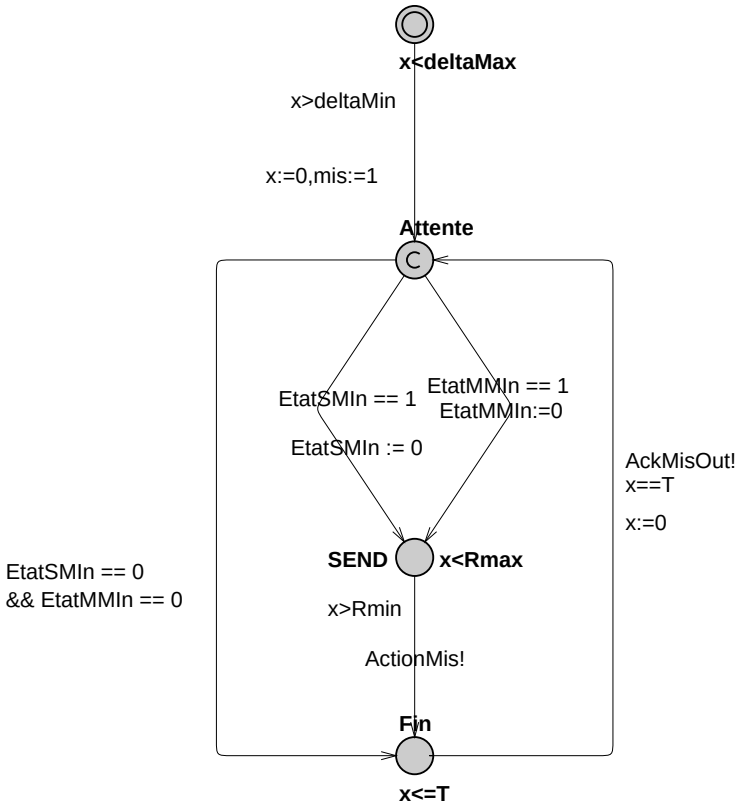


FIG. C.8 – Traduction de BlocMission

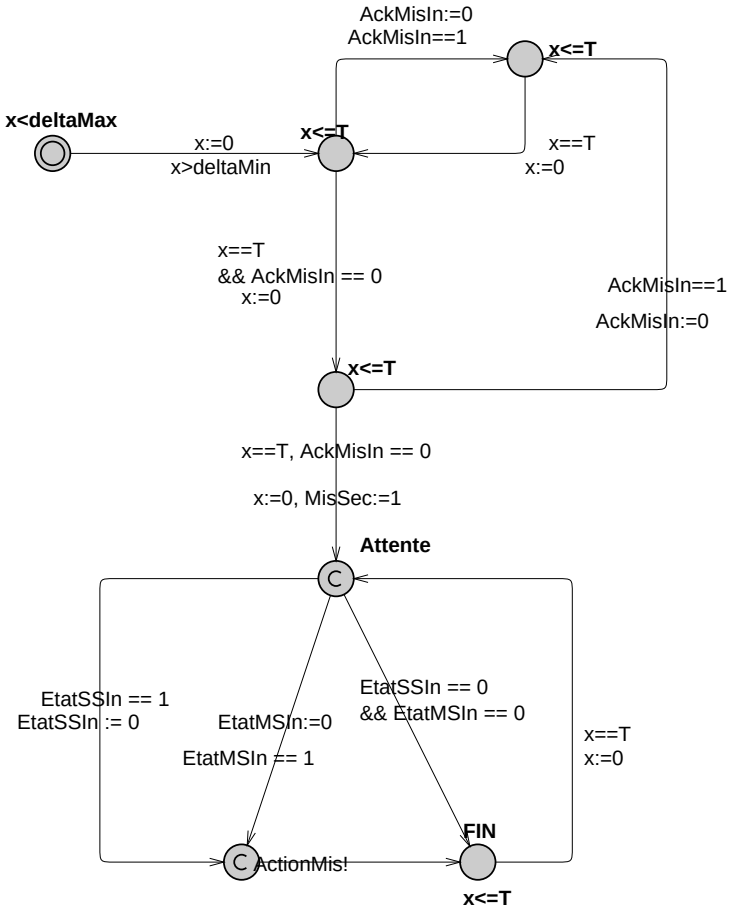


FIG. C.9 – Traduction de BlocMissionSec

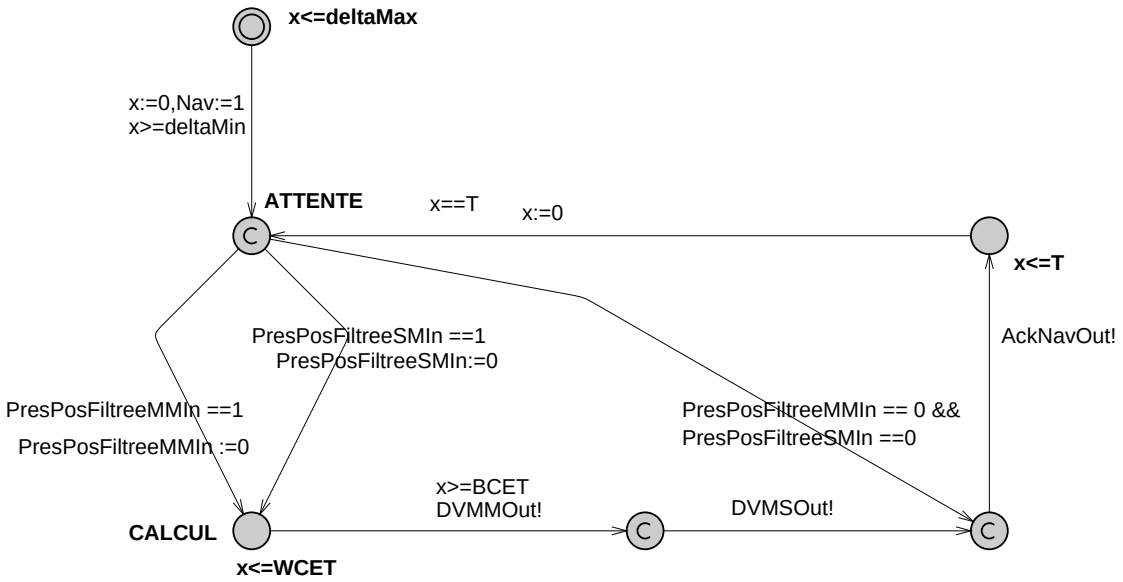


FIG. C.10 – Traduction de BlocNavigation

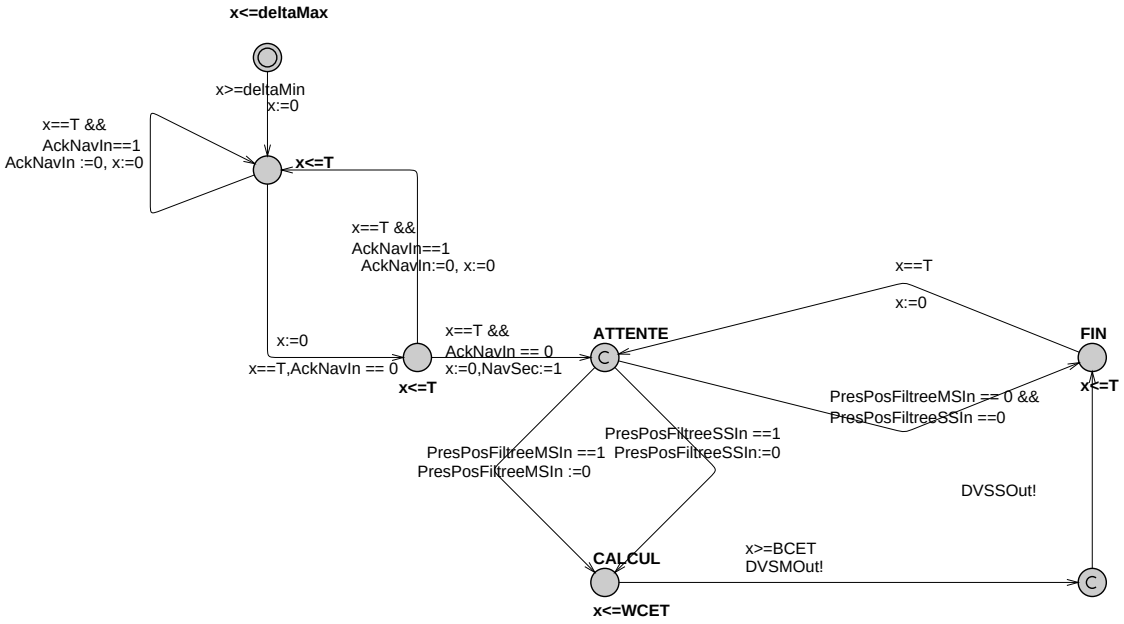


FIG. C.11 – Traduction de BlocNavigationSec

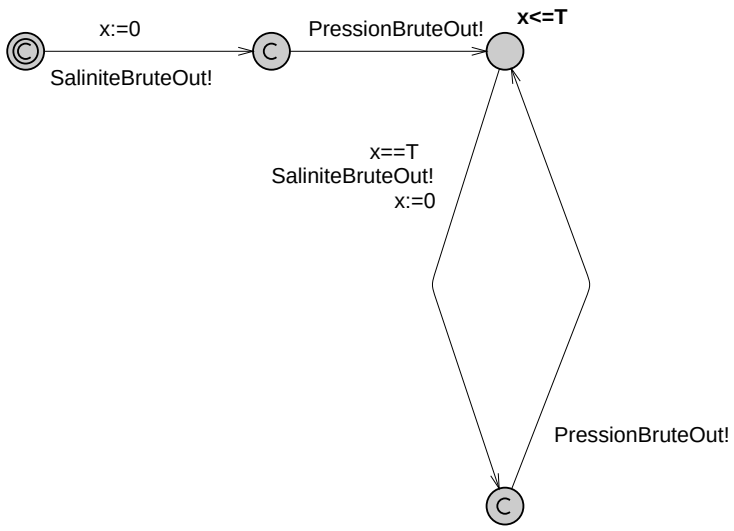


FIG. C.12 – Traduction du capteur

C.2.2 Traduction des fonctions avec modelisation des pannes

Les figures suivantes fournissent les automates Uppaal des fonctions du pivot de l'étude en tenant compte de la panne d'au plus une ressource de calcul.

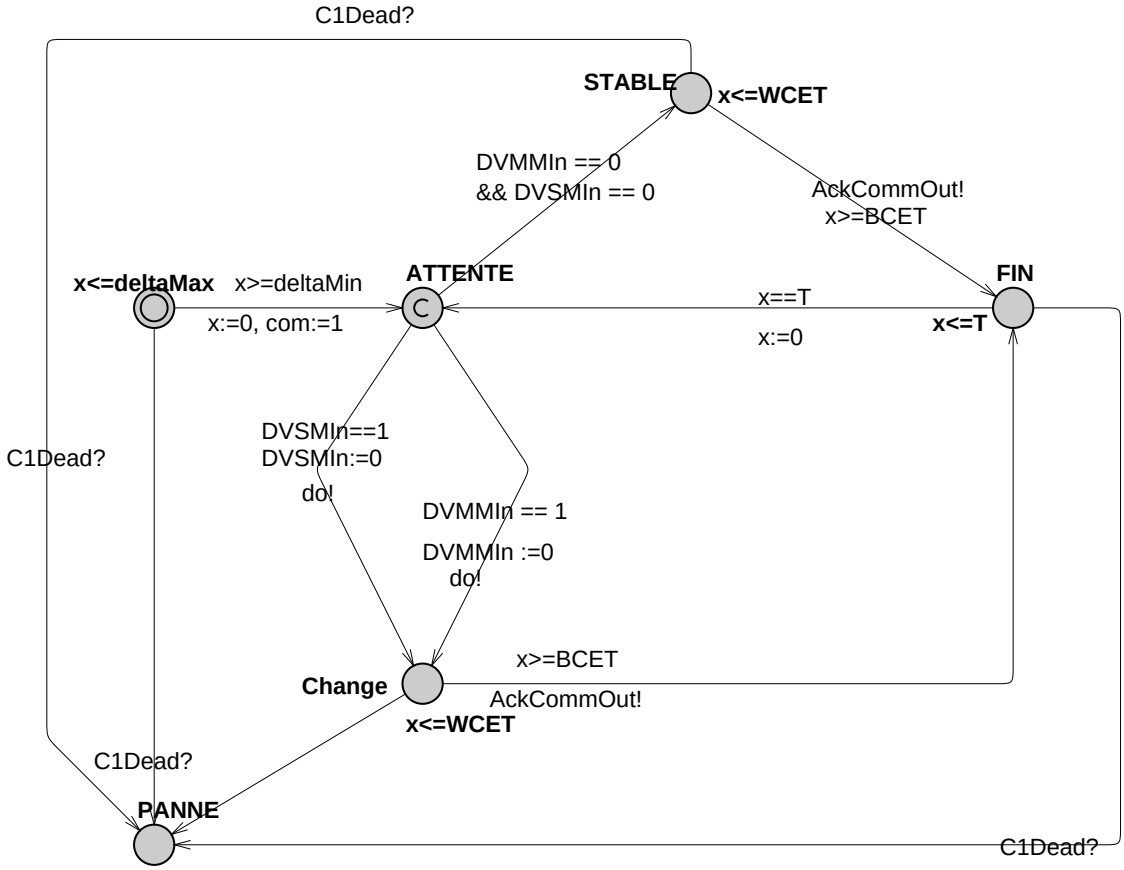


FIG. C.13 – Traduction de BlocCommande

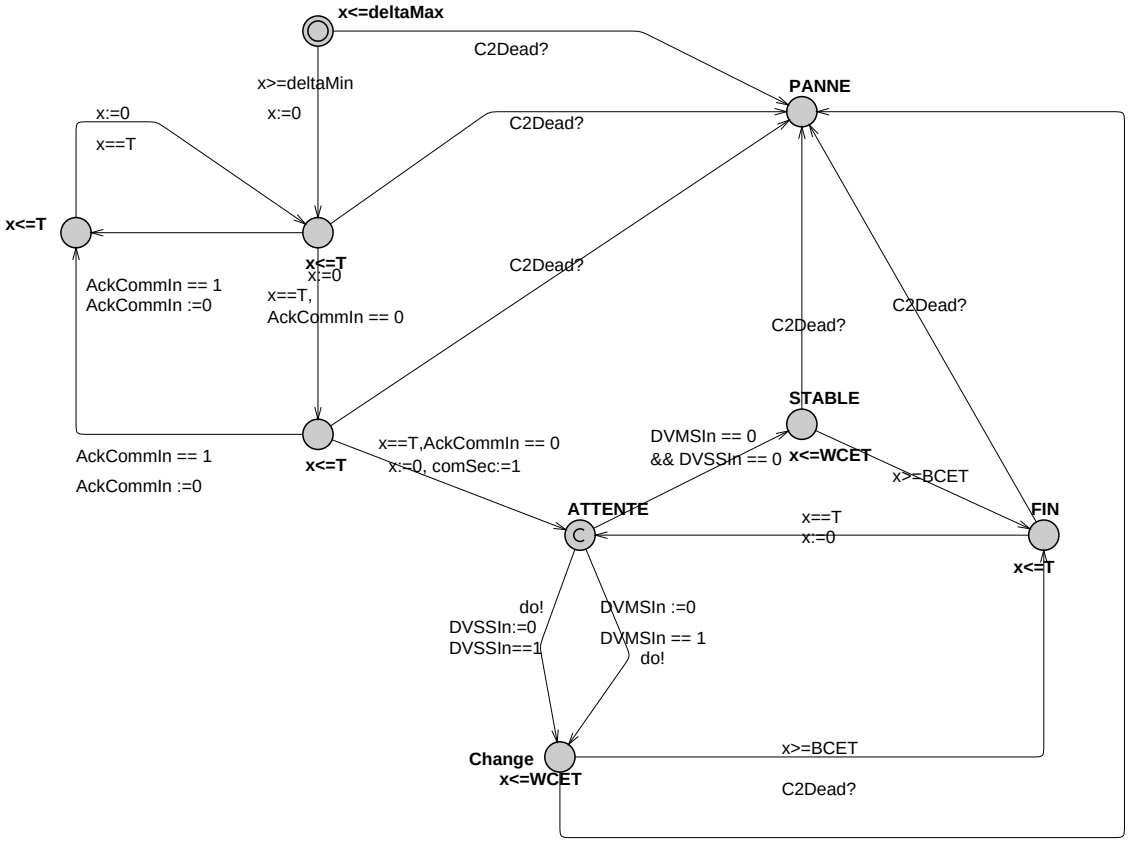


FIG. C.14 – Traduction de BlocCommandeSec

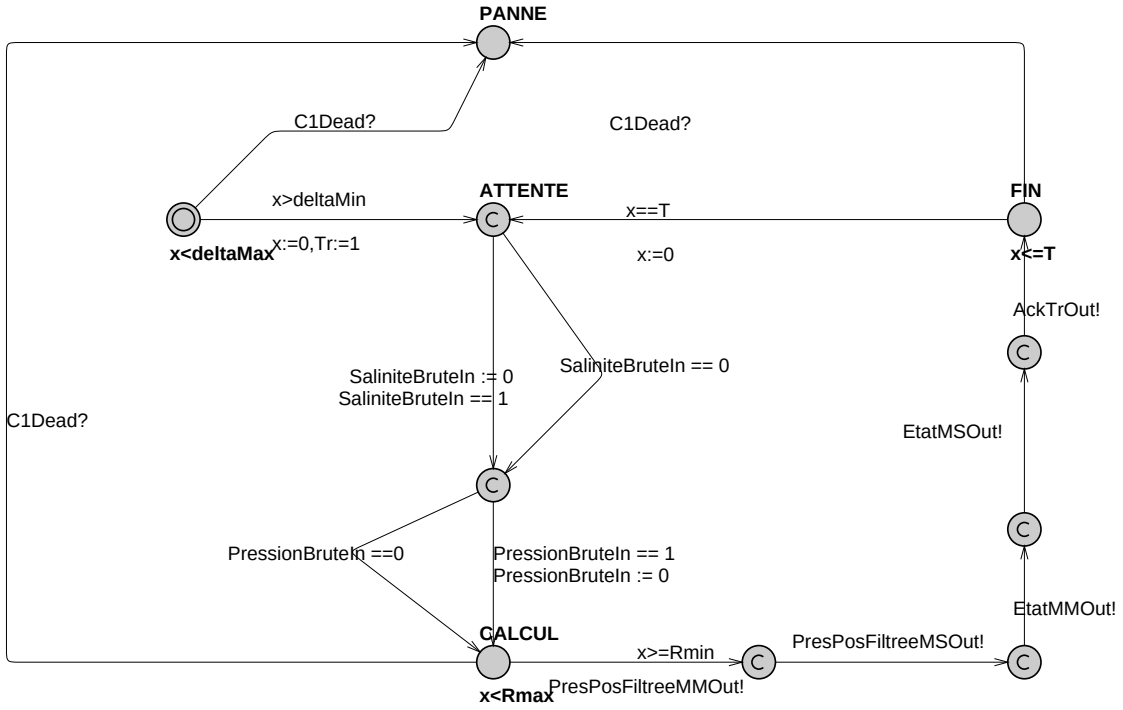


FIG. C.15 – Traduction de BlocTraitement

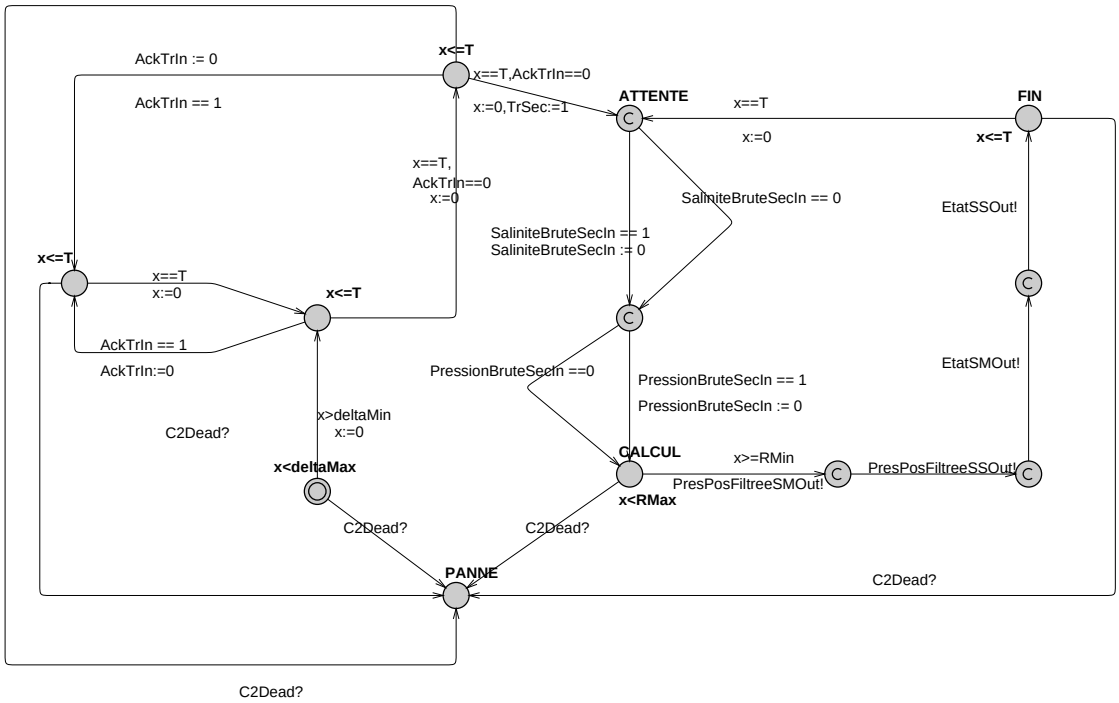


FIG. C.16 – Traduction de BlocTraitementSec

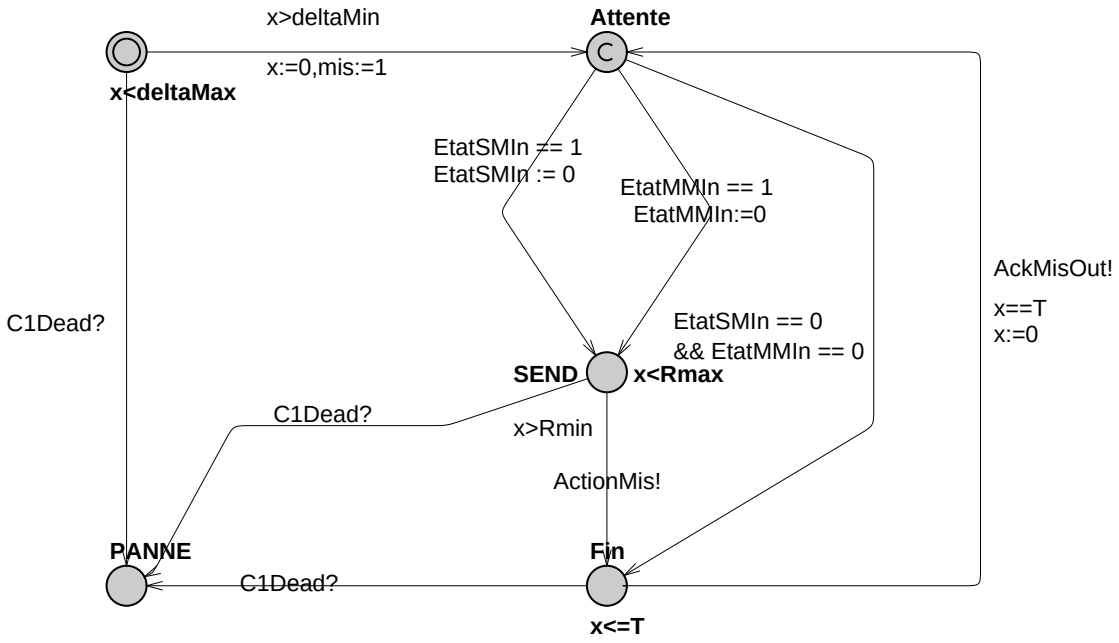


FIG. C.17 – Traduction de BlocMission

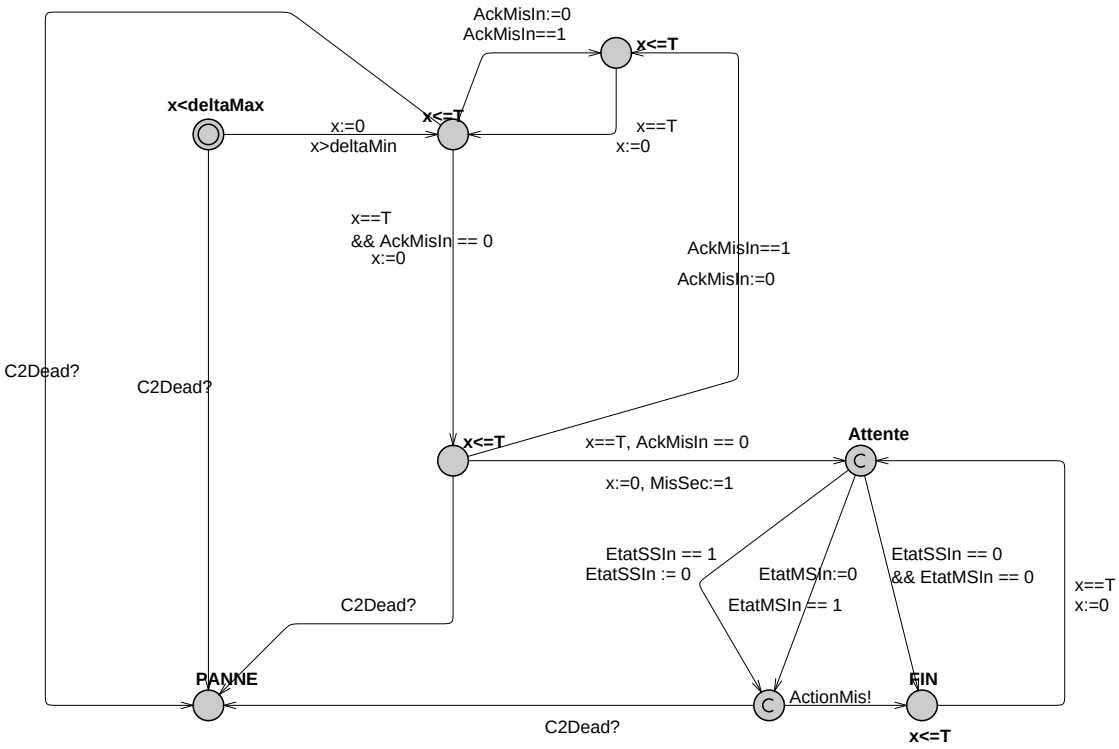


FIG. C.18 – Traduction de BlocMissionSec

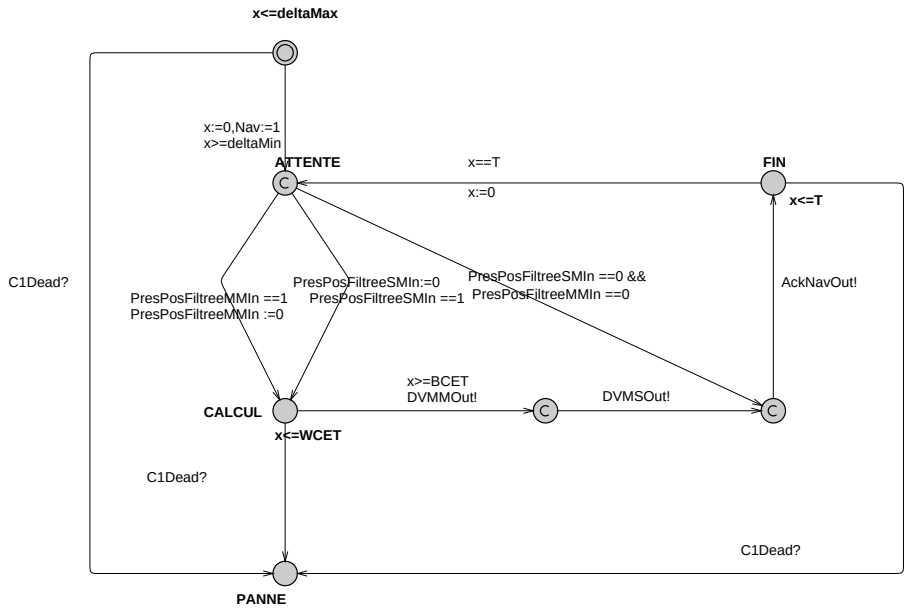


FIG. C.19 – Traduction de BlocNavigation

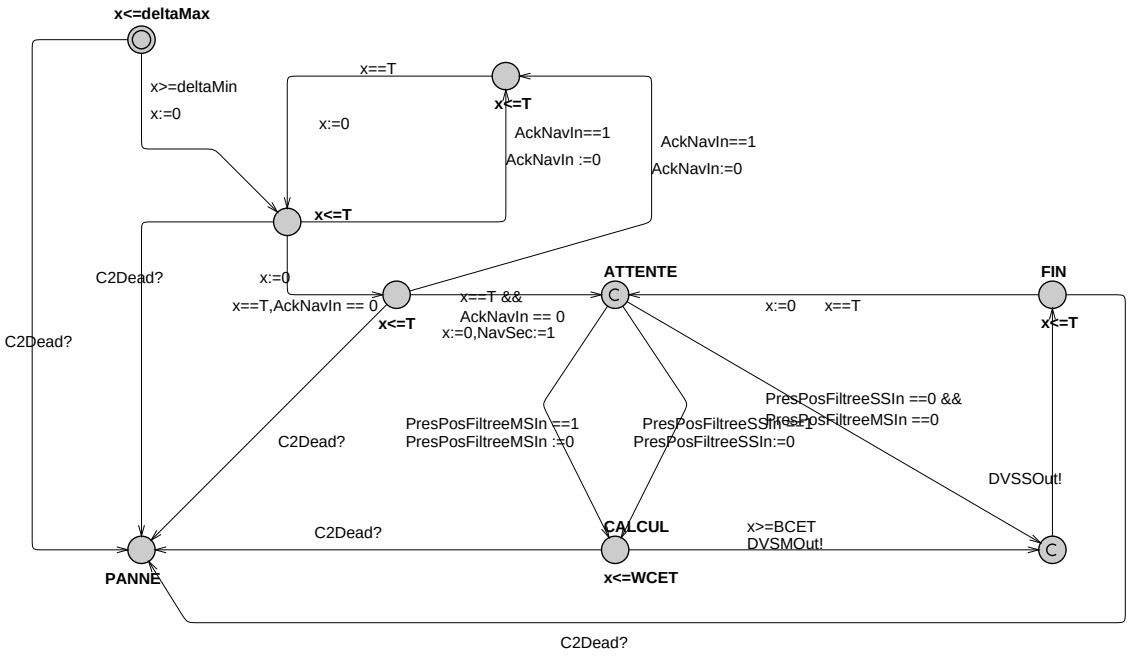


FIG. C.20 – Traduction de BlocNavigationSec

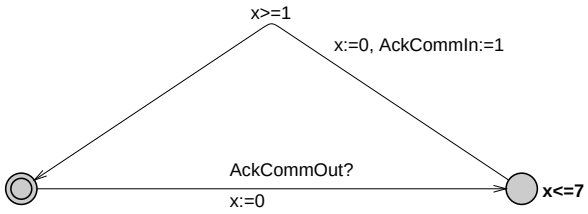


FIG. C.21 – Traduction du lien COMMS

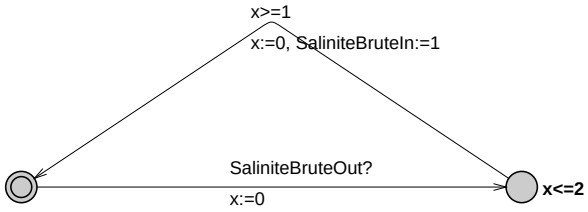


FIG. C.22 – Traduction du lien CTDTR

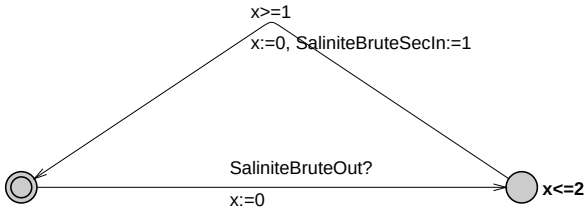


FIG. C.23 – Traduction du lien CTDTRSec

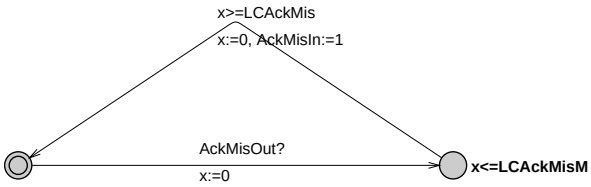


FIG. C.24 – Traduction du lien MisMS

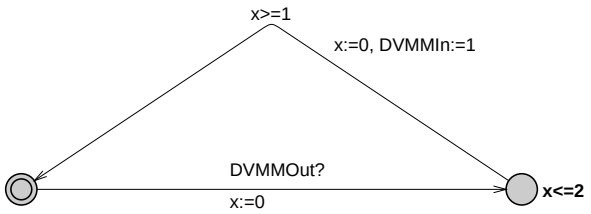


FIG. C.25 – Traduction du lien NAVCOM

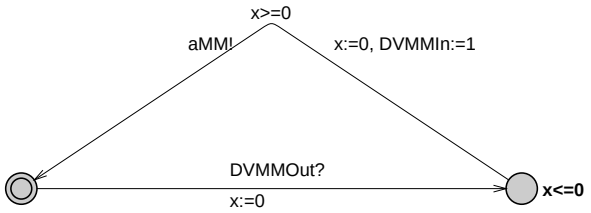


FIG. C.26 – Traduction du lien NAVCOMMM

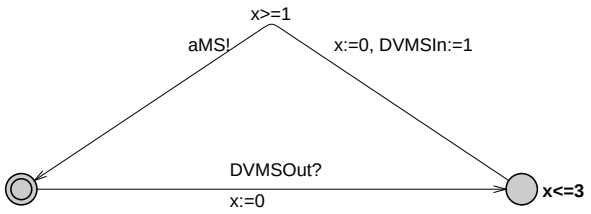


FIG. C.27 – Traduction du lien NAVCOMMS

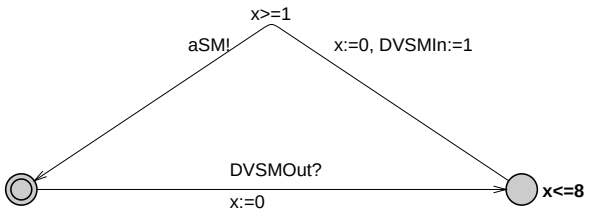


FIG. C.28 – Traduction du lien NAVCOMSM

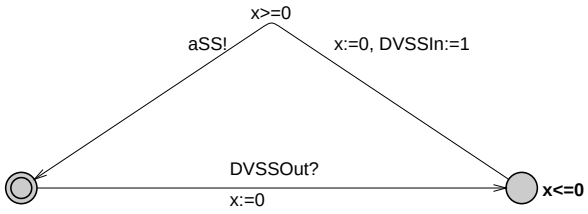


FIG. C.29 – Traduction du lien NAVCOMSS

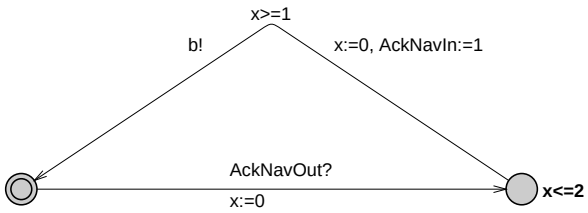


FIG. C.30 – Traduction du lien NAVMS

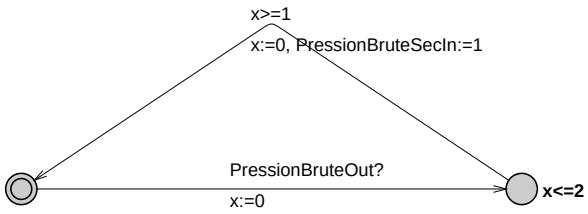


FIG. C.31 – Traduction du lien PRTRSec

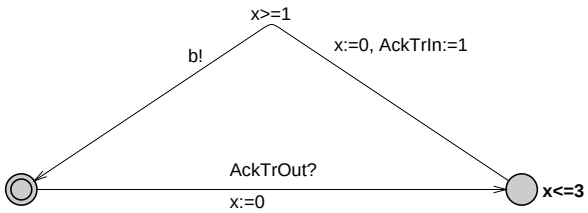


FIG. C.32 – Traduction du lien TRMS

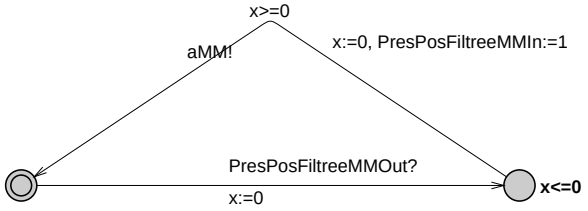


FIG. C.33 – Traduction du lien TRNAVMM

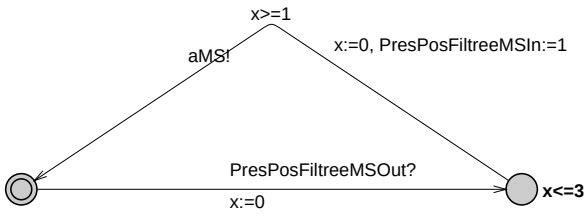


FIG. C.34 – Traduction du lien TRNAVMS

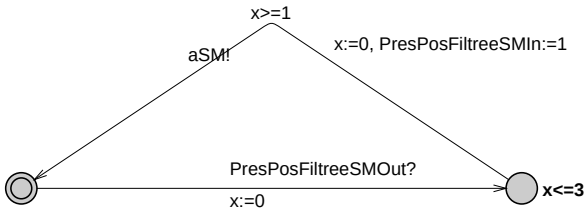


FIG. C.35 – Traduction du lien TRNAVSM

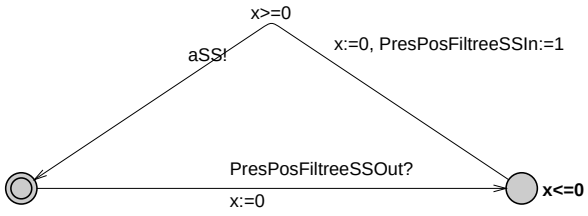


FIG. C.36 – Traduction du lien TRNAVSS

Index

Glossaire

BNF Backus-Naur Form : formalisme de description de syntaxe utilisant des symboles terminaux et non termiaux (voir les logiciels YACC et Bison)

CAN Controller Area Network

CORBA *Common Object Request Broker Architecture* http ://www.corba.org

COTS *Component Of The Shelf* Composant sur étagère

Facette Désigne à la fois un intervenant metier et le sous-système qu'il réalise (l'ensemble des modèles qu'il développe). Une facette est composée d'un modèle interface publique nommé Modèle Public et d'une partie réalisation privée cf. chapitre **??** page ??. En ce sens une facette peut être vue comme un composant de modélisation

Layer Modèle "exécutable" garant de la cohérences des PM des facettes entre eux et avec le pivot. Ils permettent via des contraintes soient de vérifier des invariant de cohérence, soit de calculer des informations transverses à partir de données extraites des PM des facettes cf. chapitre 6 page 75

MDE *Model Driven Engineering*

MIPS *Millions d'Instructions Par Seconde*

Phase à modèles stables La deuxième phase de notre processus de développement. Il s'agit de la phase de réalisation proprement dite du système voir chapitre 7 page 103

MOE *Maître d'œuvre*

MP *Modèle Public* Interface publique d'une facette

OMG *Object Management Group* http ://www.omg.org

Partie Privée Ensemble des modèles est des processus *internes* mis en œuvre par une facette pour réaliser sa tâche. Aucun des autre acteurs n'y a accés

Pivot Modèle analytique orienté fonctionnalités déstiné au maître d'œuvre. Il sert aussi à stocker les informations transverses de haut niveau calculées par les layers.

PRISME-AVN *Plateforme de Recherche pour l'Intégration la Simulation la Modélisation et l'Evaluation d'Architecture aVionique Nouvelle*

UML *Unified Modeling Language*

Table des figures

Bibliographie

[ABB+02] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jurgen Wust, and Jorg Zettel. *Component-Based Product Line Engineering with UML*. Addison-Wesley, 2002. ISBN : 0-201737-91-4.

[ACR+03] Biju Appukuttan, Tony Clark, Sreedhar Reddy, Laurence Tratt, and R. Venkatesh. A Model-Driven Approach to Building Implementable Model Transformations. UML'O3 - Workshop in Software Modeling Engineering, 2003.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[Aie97] Antoine Aiello. *Environnement oriente objet de modelisation et de simulation a evenements discrets de systemes complexes*. PhD thesis, Universite de Corse, 1997.

[ASF98] Lanusse A, Gerard S, and Terrier F. Real-time modeling with uml : the accord approach. UML'98 : Beyond the Notation, 1998.

[AT98] M. Aksit and B. Tekinerdogan. Solving the modeling problems of object-oriented languages by composing multiple aspects using composition filters, 1998.

[BBD+] G. Bel, F. Boniol, G. Durrieu, J. Foisseau, Ch. Fraboul, and V. Wiels. Modeles comportementaux pour l'avionique modulaire integree.

[BBE] Frederic Boniol, Gerard Bel, and Jerome Ermont. Modelisation et verification de systemes interes asynchrones : etude de cas et approche comparative.

[BBF00] Frederic Boniol, Gerard Bel, and Jack Foisseau. Modelisation et verification de systemes integres asynchrones dans le langage synchrone lustre : application aux systemes avioniques. *JFLA2000*, 2000.

[BDNC01] Gwendal Blorec, Veronique Delebarre, Stephane Natkin, and Eric Choveau. Compilation d'articles. In *Seminaire : Ingenierie de l'expression des exigences fonctionnelles et non fonctionnelles.* CMSL, octobre 2001.

[Bez02] Jean Bezivin. Aspect-oriented modeling : Oxymoron or pleonasm ? In *AOPDCS'02*, 2002.

[BF00] Kent Beck and Martin Fowler. *UML Distilled : A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2000. ISBN : 0-201710-91-9.

[BLP+01] F. Benaben, M. Larnac, J.P. Pignon, C.Antoine, and J. Magnier. Une methode d'aide à la conception fonctionnelle de système techniques multi-technologiques. *Génie Logiciel*, 57 :32–38, Juin 2001.

[BY] Johan Bengtsson and Wang Yi. Timed automata : Semantics, algorithms and tools.

[CAN] http ://www.interfacebus.com/Design_Connector_CAN.html.

[CFLM01] Claude Chevenier, Gauthier Fanmuy, Pascal Lamothe, and Gerard Morganti. Cours : Ingenierie des exigences, processus, demarches et outils. In *Seminaire : Ingenierie de l'expression des exigences fonctionnelles et non fonctionnelles.* CMSL, octobre 2001.

[CG03] Annie Choquet-Geniet. Panorama de l'ordonnancement temps réel monoprocesseur. In *École d'Été Temps Réel (ETR2003)*, pages 263–276, Toulouse, Septembre 2003.

[CM04] Joël Champeau and François Mekerke. Patterns, aspects and views in an mda process. In *Workshop in Software Model Engineering 2003, UML 2003*, 2004.

[CMR03] Joël Champeau, François Mekerke, and Emmanuel Rochefort. Towards a clear definition of patterns, aspects and views in mda. In *Engineering Methods to Support Information Systems Workshop, OOIS 2003*, 2003.

[Cou00] Patrick Cousot. Interprétation abstraite. *Techniques et science informatiques*, pages 1–9, 2000.

[CW96] P. Cornwell and A. Wellings. Transaction specification for object-oriented real-time systems in HRT-HOOD. 1031 :365– ? ?, 1996.

[Erm02] J. Ermont. *Une algèbre de processus pour la modélisation et la v érification des systèmes temps-réel avec préemption.* PhD thesis, ENSAE, 2002.

[FBB+99] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring : Improving the Design of Existing Code*. Addison-Wesley, 1999. ISBN : 0-201485-67-2.

[FKS02a] Robert France, Dae-Kyoo Kim, and Eujeen Song. Patterns as precise characterizations of designs. Technical report, 2002.

[FKS02b] Robert France, Dae-Kyoo Kim, and Eujeen Song. Role-based modeling language (rbml) specification v1.0. Technical report, 2002.

[Fow96] Martin Fowler. *Analysis Patterns : Reusable Object Models*. Addison-Wesley, 1996. ISBN : 0-201895-42-0.

[Fra03] David Frankel. *Model Driven Architecture : Applying MDA to Enterprise Computing*. John Wiley & Sons, 2003. ISBN : 0-471319-20-1.

[FRF+02] Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, and Randy Stafford. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002. ISBN : 0-321127-42-0.

[FS99] Martin Fowler and Kendall Scott. *UML Distilled : A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 1999. ISBN : 0-201657-83-X.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN : 0-201633-61-2.

[Gos01] Gregor Gossler. *Modelisation compositionnelle des systemes temps-reel, theorie et pratique*. PhD thesis, Universite Joseph Fournier, Grenoble, 2001.

[Gru00] J. Grundy. Multi-perspective specification, design and implementation of software components using aspects, 2000.

[Gue96] Rachid Guerraoui. Strategic research directions in object-oriented programming. *ACM Computing Surveys*, 28(4) :691–700, December 1996.

[Hil99] Rich Hilliard. Using the UML for Architectural Description. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723 of *LNCS*, pages 32–48. Springer, 1999.

[Hil01] Rich Hilliard. Viewpoint Modeling. In *Proceedings of ICSE'2001 - First Workshop on Describing Software Architectures with the UML*, 2001.

[HL95] Walter L. Hürsch and Cristina Videira Lopes. Separation of concerns. Technical Report NU-CCS-95-03, College of Computer Science, Northeastern University, Boston, MA, February 1995.

[Hol93] Ian M. Holland. *The Design and Representation of Object-Oriented Components*. PhD thesis, Northeastern University, 1993.

[Int99] Standish Group International. Chaos : A recipe for success. Technical report, http ://www.standishgroup.com/sample_research/chaos_1994_1.php, 1999.

[JBJ04] Mokrane Bouzhegoub Jacky Estublier Jean-Marie Favre Sébastien Gérard Jean Bézivin, Mireille Blay and Jean-Marc Jézéquel. Rapport de synthèse de l'AS CNRS sur le MDA. Technical report, CNRS, Nov 2004.

[JE03] F. Boniol J. Ermont. La vérification des systèmes temps réel soumis à la préemption de processus est indécidable. In *MSR2003 (Modélisation des Systèmes Réactifs)*, 2003.

[JVF] J.Foisseau, V.Weils, and F.Boniol. Un exemple de modèle conceptuel de référence pour le développement des systèmes avioniques. Communication AFIS.

[Ken02] S Kent. Model Driven Engineering. In *Proceedings of IFM 2002*, LNCS 2335, pages 286–298. Springer-Verlag, unknown 2002.

[KHH+01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. *Lecture Notes in Computer Science*, 2072 :327–355, 2001.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[KM00] Jean Pierre Krimm and Laurent Mounier. Compositional state space generation with partial order reductions for asynchronous communicating systems. 2000.

[Kri00] Jean Pierre Krimm. *Application des ordres partiels a la generation compositionnelle de systemes async hrones*. PhD thesis, Universite Joseph Fournier, 2000.

[Kru95] P. Kruchten. Architectural blueprints - the view model of software architecture. *IEEE software*, (12) :42–50, novembre 1995.

[KWB03] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained : The Model Driven Architecture : Practice and Promise*. Addison-Wesley, 2003. ISBN : 0-321194-42-X.

[Lar03] François Laroussini. Automates temporisés et hybrides. In *École d'Été Temps Réel (ETR2003)*, pages 155–166, Toulouse, Septembre 2003.

[LH89] Karl J. Lieberherr and Ian Holland. Assuring good style for object-oriented programs. *IEEE Software*, pages 38–48, September 1989.

[Lie96] Karl J. Lieberherr. *Adaptive Object-Oriented Software : the Demeter Method with Propagation Patterns*. PWS Publishing Company, Boston, 1996.

[LL] Cristina Videira Lopes and Karl J. Lieberherr. Abstracting process-to-function relations in concurrent object-oriented applications. pages 81–99.

[LLM99] Karl Lieberherr, David Lorenz, and Mira Mezini. Programming with Aspectual Components. Technical Report NU-CCS-99-01, College of Computer Science, Northeastern University, Boston, MA, March 1999.

[LO97] Karl J. Lieberherr and Doug Orleans. Preventive program maintenance in Demeter/Java (research demonstration). In *Int'l Conf. Software Engineering*, pages 604–605, Boston, MA, 1997. ACM Press.

[LP] Larsen and Pettersson. *Timed and Hybrid Systems in UPPAAL2k*. "http ://www.cs.auc.dk/ paupet/talks/MOVEP2k.html/".

[LX93] Karl J. Lieberherr and Cun Xiao. Object-Oriented Software Evolution. *IEEE Trans. Soft. Eng.*, 19(4) :313–343, April 1993.

[MB02] Stephen Mellor and Marc Balcer. *Executable UML : A Foundation for Model-Driven Architecture*. Addison-Wesley, 2002. ISBN : 0-201748-04-5.

[MDA02] Model-driven architecture. Technical report, 2002.

[Mey92] Bertrand Meyer. Applying design by contracts. *IEEE Computer*, 25(10) :40–51, 1992.

[MGFA02] François Mekerke, Geri Georg, Robert France, and Roger Alexander. Tool support for aspect-oriented design. In J.-M. Bruel and Z. Bellahséne, editors, *Advances in Object-Oriented Information Systems OOIS 2002 Workshops, LNCS 2426*, pages 280–289, September 2002.

[MTC04] Francois Mekerke, Wolfgang Theurer, and Joel Champeau. Non-functional aspects management for craft-oriented design. In *<<UML>>, Lisbon 2004 Workshop on Models for Non-functional Aspects of Component-Based Software*, 2004.

[OLL02] Johan Ovlinger, Karl Lieberherr, and David Lorenz. Aspects and modules combined. Technical Report NU-CCS-02-03, College of Computer Science, Northeastern University, Boston, MA, March 2002.

[OMG] *OMG*. http ://www.omg.org.

[OT00] H. Ossher and P. Tarr. Multi-dimensional separation of concerns and the hyperspace approach. In *Proceedings of the Symposium on Software Architectures and Component Technology : The State of the Art in Software Development*. Kluwer, 2000.

[Pal97] Jens Palsberg. Class-graph inference for adaptive programs. *Theory and Practice of Object Systems*, 3(2) :75–85, April 1997.

[PCDR02] Armelle Prigent, Franck Cassez, Philippe Dhaussy, and Olivier Roux. Extending the translation from SDL to Promela. In *9th International SPIN Workshop on Model Checking of Software (SPIN'02)*, volume 2318 of *Lecture Notes in Computer Science*, pages 400–414, Grenoble, France, March 2002. Springer-Verlag. Copyright Springer-Verlag <http ://www.springer.de>.

[Per01] Michael Perin. Coherence de specification multi-vues. *Techniques et science informatiques*, 20(7) :875–900, 2001.

[PPL96] Jens Palsberg, Boaz Patt-Shamir, and Karl Lieberherr. A new approach to compiling adaptive programs. In Hanne Riis Nielson, editor, *European Symposium on Programming*, pages 280–295, Linkoping, Sweden, April 1996. Springer Verlag Lecture Notes in Computer Science 1058.

[PPL97] Jens Palsberg, Boaz Patt-Shamir, and Karl Lieberherr. A new approach to compiling adaptive programs. *Science of Computer Programming*, 29(3) :303–326, 1997.

[Pua03] Isabelle Puaut. Méthodes de calcul de wcet (worst-case execution time) état de l'art. In *École d'Été Temps Réel (ETR2003)*, pages 263–276, Toulouse, Septembre 2003.

[PXL95] Jens Palsberg, Cun Xiao, and Karl Lieberherr. Efficient implementation of adaptive software. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 17(2) :264–292, 1995.

[Ray] Kerry Raymond. Reference model of open distributed processing (rm-odp) : Introduction.

[RFP] Uml profile for schedulability, performance and time specification, omg adopted specification, 02-03-02. Technical report.

[Ric03] Pascal Richard. Analyse du temps de réponse des systèmes temps réel. In *École d'Été Temps Réel (ETR2003)*, pages 263–276, Toulouse, Septembre 2003.

[SD99] M.W.A. Steen and J. Derrick. Applying the UML to the ODP Enterprise Viewpoint. Technical Report 8-99, Computing Laboratory, University of Kent at Canterbury, May 1999.

[SY99] Junichi Suzuki and Yoshikazu Yamamoto. Extending uml with aspects : Aspect support in the design phase. In *Proceedings of the third ECOOP Aspect-Oriented Programming Workshop*, 1999.

[SYS] *SysML*. http ://www.sysml.org.

[TBDP07] Wolfgang Theurer, Frederic Boniol, Philippe Dhaussy, and Claire Pagetti. Un cadre conceptuel pour la modélisation multi points de vue de systèmes embarqués. *Numéro spécial de la revue RTSI - L'OBJET*, 2007.

[TMC05] Wolfgang Theurer, Francois Mekerke, and Joel Champeau. "modélisation distribuée par métiers pour les systèmes embarqués". In *LMO, Berne 2005 Workshop OCM*, mars 2005.

[TMR+04] Wolfgang Theurer, Francois Mekerke, Emmanuel Rochefort, Joël Champeau, and Philippe Dhaussy. Vers la gestion de la cohérence dans les processus multi-modèles métier. In *Congrès Francophone du Management de Projet*. AFITEP, AFIS, AFAV, décembre 2004.

[YA01] Sherif Yacoub and Hany Ammar. Uml support for designing software systems as a composition of design patterns. In *UML 2001 - The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1-5, 2001, Proceedings*, 2001.

Résumé

Les systèmes embarqués sont des systèmes de plus en plus complexes, critiques, désormais à logiciels prépondérants, et souvent plongés dans un milieu physique perturbateur. La conception de tels systèmes nécessite de ce fait le concours de plusieurs équipes spécialisées dans des domaines différents : la sûreté de fonctionnement, la conception de plate-forme d'exécution et de communication temps réel.... Compte-tenu de la complexité globale induite, toutes ces équipes ne peuvent avoir chacune qu'un "point de vue" partiel du système qu'elles doivent pourtant concourir à spécifier, réaliser, et tester. Cet éclatement des compétences conduit au fait qu'il n'existe plus de définition globale et détaillée intégrant tous les aspects du système. La conséquence évidente en est une plus grande difficulté du processus de conception, et un risque accru d'erreurs ou d'incompréhensions entre équipes de conception.
Notre objectif est d'identifier les problèmes posés par la multiplication des intervenants dans la conception des systèmes embarqués, puis proposer un cadre conceptuel, reposant sur la notion de point de vue, et sur lequel sont définies les relations de cohérence entre points de vue.
Pour ce faire nous nous intéressons au point de vue de l'architecte système qui seul possède une vue globale du système appelée modèle "pivot". Nous introduisons ensuite des modèles de "facettes métiers" correspondant aux points de vue des équipes spécialisées réalisant les éléments du système. Nous montrons que ces "facettes métiers" sont reliées au "pivot" par des notions de "contrat".
Nous introduisons enfin un ensemble de modèles de "layers" décrivant chacun une préoccupation transverse (i.e., nécessitant la connaissance de plusieurs points de vue) et permettant la vérification de propriétés globales.
Enfin nous introduisons un processus de développement tirant partie des concepts précédemment introduits.
L'ensemble des notions que nous proposons sont illustrées au fil du texte par une étude de cas sur un flotteur océanographique.

Abstract

Designing embedded systems, (more and more complex, critical, software intensive and often interacting with a disturbing environment), involve several highly domain-specialized teams (safety, execution platform …) collaborating together. Each of those teams can only have a partial, restricted "point of view" on the system it contributes to build.
The goals of this thesis are first to identify what problems can be encountered when many stakeholders collaborate to develop an embedded system, and then to define a conceptual framework on which are defined the consistency relations between the "points of view". We first focus on the point of view of the system architect. It is the only stakeholder to have a global (although abstract) vision on the system which we name "pivot". We then introduce what we call "profession facets" models which describe the points of view of the specialized teams realizing the system. We show how those "profession facet" and the "pivot" are linked by a notion of "contract". Finally, we introduce a set of models called "layers" each of which describes a cross-cutting concern.
This concept of layer is the major contribution of this work. As a matter of fact, managing the constancy between the artifacts produced by the specialized teams is a major challenge since most of the most expensive errors come from the interstices (think about the impact on errors only detected during integration tests). The layers are designed to allow to avoid/detect such errors when they appear, all along the development process.
All the concepts and methods described herein are illustrated on the case study of an oceanographic autonomous buoy.