THÈSE

présentée par

Jérôme REVILLARD

pour obtenir le diplôme de
DOCTEUR DE L'UNIVERSITÉ DE SAVOIE
(Arrêté ministériel du 30 mars 1992)

Spécialité : INFORMATIQUE

Approche centrée architecture pour la conception logicielle des instruments intelligents

---

Soutenue publiquement le 15 décembre 2005 devant le jury composé de :

Jean-Pierre ELLOY	Président du jury	Professeur à l'Ecole Centrale de Nantes
Mireille BAYART	Rapporteur	Professeur à l'Université des Sciences et Technologies de Lille
Mourad Chabane OUSSALAH	Rapporteur	Professeur à l'Université de Nantes
Flavio OQUENDO	Directeur de thèse	Professeur à l'Université de Bretagne Sud
Eric BENOIT	Co-encadrant	Maître de Conférences à l'Université de Savoie
Sorana CIMPAN	Co-encadrant	Maître de Conférences à l'Université de Savoie

Préparée au sein du LISTIC
Laboratoire d'Informatique, Systèmes, Traitement de l'Information et de la Connaissance

Conclusion

Un concepteur d'instruments connaît, à l'heure actuelle, de nombreuses difficultés pour concevoir un instrument intelligent :

- la conception d'un instrument intelligent demande des compétences en informatique, en électronique, en physique et en mécanique. Or peu de personnes possèdent de telles compétences,
- les instruments intelligents sont fréquemment utilisés dans des environnements critiques. Le logiciel conçu doit alors répondre à des contraintes de sûreté de fonctionnement sévères.
- la configuration économique actuelle oblige les industriels à être innovant, à produire vite et à des coûts réduits. Or, dans le cas des instruments intelligents, ces différents objectifs sont difficiles à atteindre.

Comme il a été présenté dans le chapitre 1, ces difficultés peuvent être fortement réduites en effectuant des améliorations à différents niveaux de la conception des instruments intelligents :

1. Créer d'un modèle générique pour la partie logiciel des instruments intelligents. Cela signifie que ce modèle doit être valable pour n'importe quel type d'instrument intelligent.
2. Vérifier la conformité du logiciel créé par rapport au modèle le plus tôt possible dans le processus de développement. C'est-à-dire que l'on doit être capable d'effectuer, en phase de conception, toutes les vérifications nécessaires au respect du modèle de conception.
3. Avoir un modèle générique qui puisse évoluer facilement en fonction de l'apparition de nouveaux besoins dans le domaine ou de nouvelles technologies.
4. Avoir un processus de conception qui permette un passage aisé entre le modèle générique et le code source du logiciel à créer. L'idéal étant une génération automatique du code source depuis le modèle.
5. Le processus de conception ne doit pas subir de modifications importantes en cas d'évolution du modèle. Ceci concerne le processus en lui-même mais aussi les différents outils qui le compose.

La thèse [Tailland 00] a été réalisée avec pour objectif de créer un modèle et des outils qui facilitent le travail des concepteurs d'instruments intelligents. Ce modèle devait permettre de modéliser n'importe quel type d'instruments intelligents et les outils proposés, regroupés sous le nom de CAPTool, devaient garantir la sûreté et la fiabilité tant au niveau

de la conception que du fonctionnement. Néanmoins, ce type d'outils reste limité sur plusieurs aspects et notamment sur leur faculté d'évolution et leur facilité de mise en œuvre. De plus, on a pu rapidement s'apercevoir que l'on ne pouvait pas concevoir tous les types d'instruments intelligents avec le modèle créé.

Par conséquent, [Tailland 00] répond partiellement au point 1 précédent puisque le modèle n'est pas complet. Il répond complètement au point 2 puisque toutes les propriétés de son modèle sont vérifiées en phase de conception. Il répond au point 3 par l'utilisation d'un modèle en couche ou chacune est basée sur la théorie des graphes. Il répond au point 4 car une génération automatique du code est possible dans son approche. Par contre le point 5 n'est pas du tout pris en compte. En effet, si le modèle évolue, toute l'implémentation des outils qui se trouvent dans le processus doit être refaite.

Les travaux présentés dans ce manuscrit se positionnent dans la suite de la thèse de [Tailland 00] et ont pour but de repenser à la fois le modèle des instruments intelligents et les outils existants car il est apparu que les limitations auxquelles nous étions confrontées étaient à la fois liées au modèle de conception qui avait été proposé et à la façon dont avait été implémenté CAPTool.

I. Bilan et remarques

Le travail réalisé pour cette thèse a donc porté sur deux points :

- Amélioration du modèle de conception des instruments intelligents afin de pouvoir modéliser tous les types d'instruments.
- Amélioration du processus de conception et des outils qui le compose afin que le tout soit évolutif.

Modification du modèle existant :

En ce qui concerne la modification du modèle existant, une nouvelle couche répondant aux manques repérés précédemment a été rajoutée : la couche graphe des services internes. Cette couche fait que les services externes ne sont plus traités comme des sous ensembles de l'ensemble des services internes mais comme des sous graphes du graphe des services internes. Grâce à cette nouvelle modélisation, une certaine catégorie d'instruments intelligents qui ne pouvait pas être conçue jusqu'alors l'est désormais (voir chapitre 4 section I). Un autre avantage de cette couche est que l'on peut dès lors énoncer des propriétés sur la structure et sur le comportement global des instruments intelligents.

Le processus de conception et les outils qui le compose :

La partie qui concerne l'élaboration d'un nouveau processus de conception et des outils qui le compose a été pensée en utilisant les principes et outils du domaine de la conception centrée architecture. Ce domaine est en plein essor dans l'ingénierie des logiciels. En fait, l'architecture décrit la structure et le comportement du système dans son ensemble, et repose généralement sur la description de composants (unités de calcul localisées et indépendantes), de connecteurs (unités d'interaction entre les composants) et de configuration (ensembles de composants combinés via des connecteurs). D'autre part, ce développement nous offre la possibilité de créer nos propres langages, formels, spécifiques au domaine. Le fait que ces langages soient formels autorise des vérifications accrues en phase de conception. Pour mettre en œuvre concrètement cette approche, le langage de description d'architecture nommé ArchWare ADL a été choisi car il est apparu comme le plus approprié à notre problématique.

La couche graphe des services internes est modélisée grâce au langage spécifique à la notion de style architectural : ASL. Les services externes et les modes sont eux construis par raffinement du graphe des services internes avec le langage ARL. Toutes les propriétés du modèle ont aussi été implémentées dans le langage ASL afin de pouvoir les vérifier en phase de conception. L'apport de cette nouvelle méthode de conception est immense car les langages utilisés, bien qu'ils utilisent une syntaxe appartenant au domaine, sont formels. Ceci permet de faire des vérifications poussées dès la phase de conception. De plus, le langage ASL est construit afin de pouvoir modifier le langage spécifique sans avoir à effectuer aucune autre modification. Il est, en plus, possible d'ajouter ou de supprimer des propriétés très facilement en cas d'évolution du modèle. La partie qui est plus difficile à modifier est celle qui gère le langage spécifique placé au dessus du langage ARL car cette fonctionnalité ne fait pas partie de l'ADL lui même.

Un nouveau processus de conception basé en majorité sur les outils issus du projet ArchWare a donc été conçu. Il comporte quelques outils spécifiques et notamment un chargé de la génération automatique du code source Java qui sera directement embarqué dans les instruments intelligents. Le fait d'utiliser en majorité des outils ArchWare diminue grandement (voir annule) les différents besoins de modification du processus qui pourraient survenir en cas d'évolution du modèle.

Les outils ArchWare, à la base, n'ont pas été créés pour être utilisés dans la conception des instruments intelligents. Ce qui peut apparaître comme un désavantage au début s'est très vite transformé en un atout. En effet, du point de vue des instruments intelligents, ArchWare a apporté le formalisme qu'il manquait jusqu'alors pour que la conception soit sûre. Mais les apports se sont faits dans les deux sens. Les langages ArchWare ont bénéficié de nos travaux (surtout le langage ASL) pour s'étoffer, s'améliorer. Par exemple, on s'est aperçu que le langage ASL était trop rigide au départ pour pouvoir concevoir des éléments génériques qui étaient structurés de façon identique mais qui transportaient des données de types différents. Le langage ASL a donc été amélioré en ajoutant la possibilité de passer des types en paramètre à un style. Le seul désavantage de ces améliorations est que l'implémentation des outils n'a pas suivi par manque de temps. Par conséquent, le processus qui a été créé pour la conception des instruments intelligents ne peut être utilisé tel quel actuellement.

Remarque :
Il est intéressant de remarquer que depuis quelques années, le fait de vouloir vérifier des logiciels en phase de conception est une approche qui commence à se démocratiser et nos travaux en sont la preuve. Le fait d'utiliser une telle approche n'est pas quelque chose de nouveau, cependant, jusqu'à présent, ce type de conception était réservé à des entreprises ou à des domaines qui avaient de gros moyens financiers et pour qui la moindre erreur dans la conception d'un logiciel entraînait immédiatement une perte financière énorme (ex : le domaine de l'aérospatiale). De plus, ce type d'approche demandait jusqu'alors beaucoup plus de temps qu'une conception classique et n'était pas à la portée de tous car les langages utilisés (par exemple le langage B) sont des langages complexes.

Les langages de description d'architectures commencent à mettre à portée de tous ce type d'approche car les efforts qui doivent être fournis en phase de conception ont fortement diminués. En effet, grâce à l'intégration de mécanismes tel que la création de langages

formels spécifiques, les notions d'héritage et d'agrégation, plusieurs points ont été nettement améliorés :

- les langages ne sont plus réservés à des spécialistes,
- les possibilités de réutilisation ont été accrues,
- le temps de conception a nettement diminué.

Tout ceci va sans doute nous emmener à une démocratisation de ce type de conception.

II. Perspectives

Les perspectives de ces travaux sont diverses et peuvent être explorées en trois temps :

- court terme,
- moyen terme,
- long terme.

Court terme :

Une des perspectives à court terme de ces travaux est de pouvoir réaliser une validation complète de l'approche. Pour ce faire, la plus grande partie du travail va résider dans le fait de terminer les outils ArchWare afin qu'ils implémentent toutes les fonctionnalités prévues dans leurs spécifications.

Une deuxième perspective de recherche serait l'intégration de nos travaux dans l'application gMDEnv [Manset et al. 05]. Nous avons vu que cette application vise à implémenter les concepts relatifs à l'ingénierie des models (IDM) dans le cadre de la génération automatique d'applications Grid. Il est envisageable de l'étendre pour faire en sorte qu'elle puisse générer le code source des instruments intelligents. En effet, cette application, basée à l'origine sur le paradigme architecture orientée services, délivre un ensemble de fonctionnalités, dédiées aux systèmes Grid, similaires aux besoins rencontrés dans le domaine des instruments intelligents. De part sa conception, il est possible d'étendre le champ applicatif de cet environnement de développement par l'ajout de modèles propres à un domaine spécifique. Ainsi, dans le cadre de la conception d'instruments intelligents, il serait possible d'intégrer le modèle présenté dans nos travaux. L'application fournirait alors au concepteur d'instruments intelligents une interface graphique lui permettant de concevoir visuellement ses instruments et de générer automatiquement le code source. Il ne faut pas perdre de vue que le type de conception itilisé alors serait toujours une conception centrée architecture.

Moyen terme :

La perspective que l'on peut envisager à moyen terme est l'extension de notre approche à la conception d'une application complète, qui comporterait plusieurs instruments intelligents. Sachant que ces derniers peuvent interagir, c'est à dire qu'un instrument peut demander l'exécution d'un service à un autre, on peut imaginer pouvoir construire le graphe des services internes de l'application complète, qui serait en fait l'association de tous les graphes des services internes qui composent l'application. A partir de ce moment, d'autres questions de recherchent pourrais faire leur apparitions :

- Est ce qu'il y a des propriétés spécifiques aux applications composées d'instruments intelligents ?

- Les propriétés qui sont vraies pour le graphe des service internes des instruments intelligents sont elles toujours vraies pour le graphe d'une application totale ? (par exemple, le rebouclage au sein du graphe complet de l'application)

Long terme :

A plus long terme, une perspective importante sera la prise en compte d'une donnée importante dans le monde de l'instrumentation intelligente et qui manque actuellement au niveau du modèle : la notion de temps d'exécution des services. En effet, dans certaines applications, le temps d'exécution d'un service est une donnée importante car elle peut déterminer par exemple l'exactitude d'une mesure ou encore si un matériel est en panne ou non (un service interne qui doit durer en moyenne cents millisecondes et qui n'est toujours pas terminé au bout d'une seconde présente selon toute vraisemblance une anomalie). Il serait donc intéressant de pouvoir vérifier, en phase de conception, si des contraintes temporelles peuvent être vérifiées. Il pourrait y avoir deux méthodes possibles pour envisager ces vérifications. La première serait l'intégration dans ArchWare de cette notion temporelle et la deuxième serait de construire notre modèle pour faire en sorte de la simuler.