Le vrai sens du développement

EMMANUEL GAUDIN

Résumé : Le développement orienté modèle a suscité beaucoup d'intérêt cette dernière décennie. Un des principes sous-jacents à ce type d'approche est de passer le plus de temps possible en haut de la branche gauche du cycle en V, c'est-à-dire à comprendre ce que l'on cherche à faire avant de plonger dans les détails d'implémentation. Comme chaque domaine concerne des problématiques différentes et que chaque étape du processus de développement a des besoins spécifiques, de nombreuses technologies ont fait leur apparition, voire leur réapparition, sur le marché. Quel que soit le niveau de détail des modèles, se pose la question du test et de la vérification. Or le test d'un modèle sous-entend une vérification dynamique et impose la possibilité de pouvoir exécuter ce dernier. Pour cela le langage de modélisation doit inclure une sémantique d'exécution et un langage d'action. Pour peu que l'on puisse avoir les mêmes caractéristiques dans un langage de test, il sera alors possible de retrouver le vrai sens du cycle en V : du haut vers le bas.

Mots clés : AADL, TASTE, SDL-RT, SDL, ASN.1, TTCN-3, IF, *model-checking*, test guidé par les modèles, ingénierie guidée par les modèles.

1. Quel degré de précision dans les modèles

Un des premiers objectifs d'un modèle est de faciliter la communication et la réflexion entre parties prenantes sur un sujet défini. On imagine aisément, par exemple, le synoptique d'une maison individuelle en quelques coups de crayon. Cela donnera une base de réflexion pour un architecte qui va détailler plus avant le projet. Dans un deuxième temps, l'architecte va détailler l'organisation du bâtiment pour préciser et valider un certain nombre de grandes idées avec son client. Les plans deviendront de plus en plus précis. Enfin, une fois cette deuxième phase validée, l'architecte va détailler les plans afin qu'un bureau d'étude vérifie la cohérence technique de l'ensemble et que ces derniers servent de référence contractuelle avec un entrepreneur. On voit dans cette démarche différents types de modèles avec différents niveaux de formalisation. Le modèle le plus précis a pour objectif de permettre d'effectuer un certain nombre de vérifications et s'appuie sur des concepts dédiés au métier. Une fois tous ces modèles vérifiés on passe à la construction.

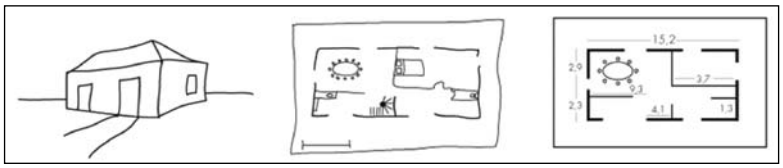

Figure 1 : Des modèles plus ou moins formels

Dans une démarche de développement logiciel orientée modèle, on va aussi retrouver différents niveaux de modélisation. Un modèle très informel permettra de documenter ou de discuter les grands principes d'une application, un modèle semi-formel permettra de rentrer dans les détails de la réalisation et le modèle formel sera exécutable et vérifiable. Pour cela il faut un langage de modélisation avec un fort degré de formalisme et lié au domaine d'application. Il existe aujourd'hui de nombreux langages de modélisation, et il est intéressant de les positionner les uns par rapport aux autres suivant leur niveau de précision et leur domaine d'application.

On se rend compte qu'il n'y a pas de langage universel qui permettrait de décrire n'importe quel type de modèle, mais des langages adaptés à chaque étape du développement ou domaine d'application. On notera au passage dans ce graphique la ressemblance avec la branche gauche du cycle en V.

Le modèle de l'application le plus poussé ayant pour objectif d'être vérifié, il est impératif que le langage de modélisation permette non seulement de définir les interfaces entre les éléments du système, mais surtout permette l'exécution du modèle. Ceci permettra de simuler le modèle afin de vérifier que les scénarios nominaux sont corrects, mais aussi de vérifier des propriétés quelles que soient les entrées du système (techniques de model-chec-

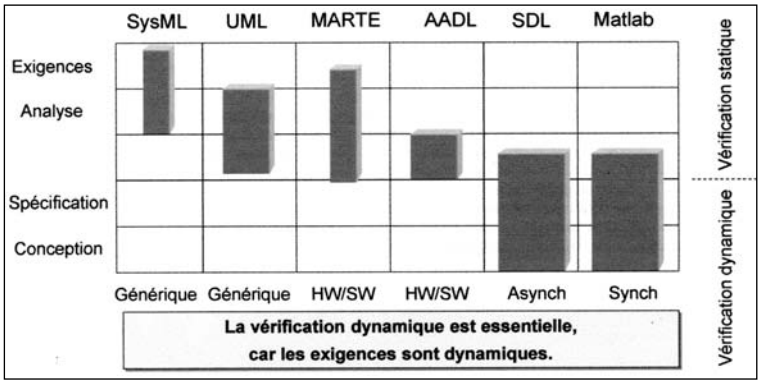

Figure 2 : Positionnement des langages de modélisation

king). Dans la pratique les techniques de *model-checking* [1] essaient toutes les combinaisons d'entrées possibles dans le système et vérifient à chaque unité d'exécution si un certain nombre de propriétés sont bien respectées. On s'assure donc que certaines propriétés sont toujours respectées quelles que soient les conditions, ce qui caractérise la robustesse du système. Les mêmes techniques peuvent être utilisées pour générer des cas de test. Dans ce cas les propriétés sont des objectifs de test qui doivent être atteignables. On peut alors trouver tous les scénarios qui permettent d'atteindre chaque objectif de test. Un outillage adéquat permettra alors de sélectionner les scénarios les plus courts et/ou d'éliminer les redondances. On retiendra, par exemple, SDL [2], standard de l'ITU-T, comme langage de modélisation pour les systèmes asynchrones qui offre toutes les caractéristiques nécessaire pour décrire un modèle exécutable et vérifiable.

2. L'INTÉGRATION CONTINUE

L'utilisation de langages formels permet l'exécution du modèle afin de le vérifier. Il est alors d'ores et déjà possible de tester ce dernier. Pour cela on pourra s'appuyer sur un langage de test qui propose le même niveau d'abstraction que le langage de modélisation. Contrairement aux langages de modélisation, les langages de test sont étonnement rares. Il paraît alors important de citer un des standards dans ce domaine : le TTCN-3 [3]. Ce standard de l'ITU, initialement de l'ISO, est un langage textuel qui s'appuie sur un certain nombre de services, comme les messages ou les temporisateurs, et une sémantique d'action comme, par exemple, l'exécution parallèle. Il peut être considéré comme formel, car il possède ses propres types de données et son propre langage d'action. Les notions d'alternatives et de gabarits permettent de gérer les différentes réponses possibles du système. Enfin, le langage TTCN-3 permet de tester une implémentation synchrone ou asynchrone.

TTCN-3 non seulement permet de tester des modèles formels de haut niveau,mais aussi propose un mapping d'implémentation, ce qui permet alors de tester une implémentation. Ce langage permet donc de décrire des tests sur toute la branche droite du cycle en V quelle que soit la sémantique du modèle en vis-à-vis.

Une fois de plus cette démarche n'a de sens que si les modèles dans la branche gauche du cycle en V sont exécutables, c'est-à-dire formels.

3. LE VRAI SENS DE LA PROGRESSION DU DÉVELOPPEMENT

Cette approche permet de démarrer les tests en haut du cycle en V sans descendre dans les branches. On pourra alors valider un modèle de haut niveau et les tests associés avant de passer à l'étape suivante. L'utilisation d'un langage de test permettra d'ailleurs d'automatiser les tests afin de facilement les ré-exécuter lors d'éventuelles évolutions incrémentales. On peut donc attaquer les deux branches du cycle simultanément et enfin le parcourir dans le bon sens, c'est-à-dire de haut en bas, pas de gauche à droite.

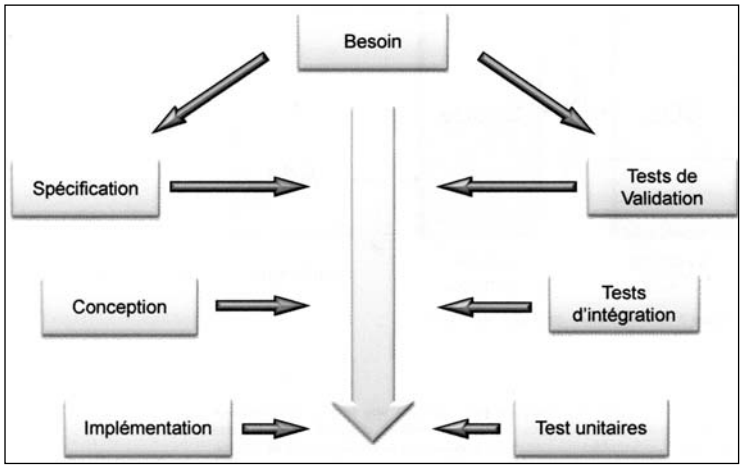

Figure 4 : Le véritable sens du développement

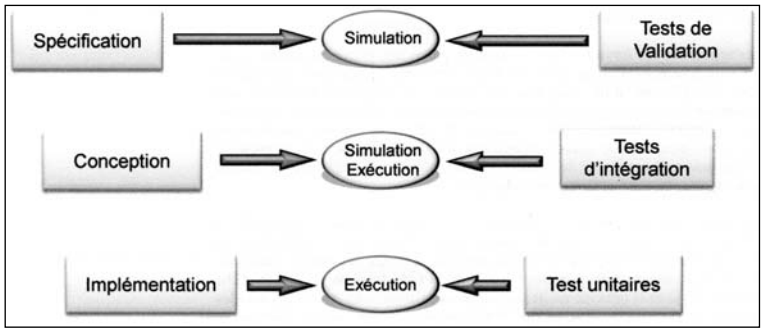

Figure 3 : Vérification le long du cycle en V

4. LES VRAIS SYSTÈMES SONT HÉTÉROGÈNES

Un bon langage de modélisation s'appuie sur des services et une sémantique précis afin d'être exécutable. Cette démarche implique qu'un langage unique ne permettra pas de décrire tous les modèles. Typiquement, si on décrit une loi de contrôle comme, par exemple, celle d'un régulateur de vitesse, on s'appuiera plutôt sur des technologies dites synchrones basées sur l'arrivée régulière d'échantillons de mesure. Un modèle de ce type de système comportera des fonctions de calcul dif-

férentiel afin de calculer la réponse à apporter aux mesures effectuées. Par contre, si on décrit un protocole de télécommunication, on s'appuiera sur une technologie asynchrone dans laquelle les informations sont échangées de manière non déterministe. Il s'agit alors de décrire le comportement du système suivant la séquence des informations échangées.

Un système complexe est globalement plutôt asynchrone et comporte des parties synchrones (GALS) [4] ; il comportera donc ces deux aspects. C'est pour cette raison que des travaux ont été entrepris par l'Agence Spatiale Européenne ces dernières années pour proposer une représentation qui permette de réunir des modèles basés sur des sémantiques différentes au sein d'un même super-modèle. C'est l'objectif du framework TASTE (The ASSERT Set of Tools for Engineering) [5] dont le principe est de décrire un super-modèle en AADL [6] dont les interfaces entre les blocs sont décrits en ASN.1 [7]. Chaque bloc AADLest alors raffiné avec le langage de modélisation le mieux adapté comme, par exemple,

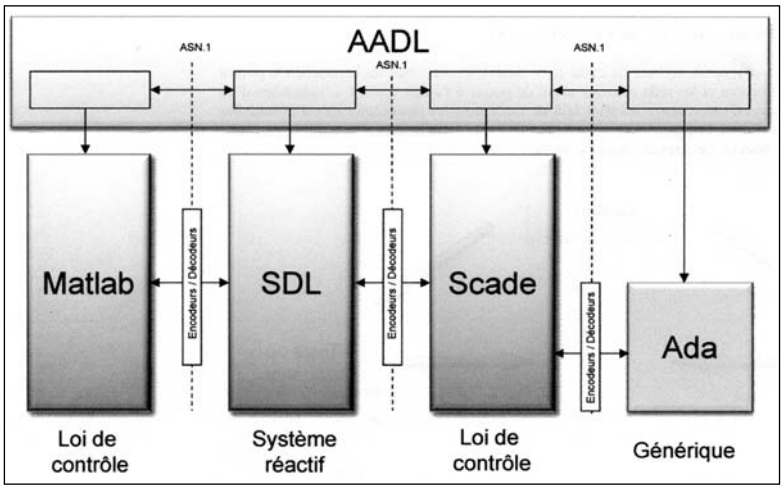

Figure 5 : L'architecture du framework TASTE

SCADE [8] pour un bloc synchrone et SDL pour un bloc asynchrone. Les modèles sont simulés et validés indépendamment puis peuvent être exécutés ensemble automatiquement avec les encodeurs/décodeurs ASN.1 fournis dans le framework adaptés aux générateurs de code des différents outils.

5. Conclusion

Le terme de modélisation peut recouvrir des aspects très variés. Il peut recouvrir une simple documentation comme il peut recouvrir des modèles extrêmement détaillés. Dans les systèmes critiques il est souvent recommandé d'utiliser des modèles formels exécutables permettant la simulation et la vérification de propriétés. Il est alors d'autant plus intéressant de couvrir les aspects tests très tôt qui permettront une intégration continue qui pourra s'adapter aussi bien à un cycle classique en cascade qu'à un cycle agile. Le développement retrouvera alors son véritable sens. Enfin, l'intégration de différentes technologies de modélisation dans des environnements industriels tels que TASTE permettent de couvrir tous les aspects du logiciel et même du matériel.

6. Références

[1] Edmund M. Clarke, Allen Emerson et Joseph Sifakis : *Model-checking: algorithmic verification and debugging* ; Communications of the ACM, vol. 52, n° 11, pp. 74-84, novembre 2009

[2] SDL: Specification and Description Language – norme Z.100 ITU-T.

[3] TTCN-3: Testing and Test Control Notation – norme Z.140 ITU-T.

[4] J. Muttersbach, T. Villiger et W. Fichtner : *Practical design of globally-asynchronous locally-synchronous systems (GALS)* ; actes de la sixième conférence ASYNC, pp. 52-59, 2000.

[5] M. Perrotin, E. Conquet, P. Dissaux, T. Tsiodras et J. Hugues :*The TASTE toolset: turning human-designed heterogeneous systems into computer-built homogeneous software* ; ERTS 2010 Conference.

[6] AADL (Architecture Analysis & Design Language) : un langage spécifié par SAE (the Society of AutomotiveEngineers).

[7] ASN.1 (Abstract Syntax Notation One) : une norme conjointe de ISO/IEC et ITU-T.

[8] SCADE (Safety Critical Application Development Environment) : un environnement de développement intégré développé par Esterel Technologies (voir *Génie Logiciel*, n° 92, pp. 7-15, mars 2012).