

CAO ÉLECTRONIQUE

Conception de systèmes : explorer les architectures sans a priori

Alors que la modélisation fonctionnelle des systèmes est pratiquée depuis longtemps, la modélisation au niveau architectural est plus récente. Pour mener à bien ces deux approches, il convient de dépasser la vision traditionnelle d'un système sous le seul aspect logiciel/matériel. Une nouvelle voie consiste à différencier l'application de la plate-forme pour mieux les assembler ensuite. CoFluent explique ici comment.

L'accroissement de la complexité en termes de fonctionnalité et d'architecture des systèmes électroniques multiprocesseurs, qu'ils soient sur une puce, une seule carte ou distribués sur plusieurs cartes, rendent la modélisation et simulation de ces systèmes indispensable. A un degré d'abstraction élevé, la conception au niveau système ou ESL (Electronic system level) devient indispensable. De plus en plus fréquemment pratiquée, elle est définie comme étant la conception conjointe du matériel (hardware) et du logiciel (software) dans un système électronique. Mais, dans la pratique, développer du matériel ou du logiciel sont des métiers différents pratiqués par des équipes séparées composées d'ingénieurs aux formations souvent distinctes utilisant des méthodes, des procédés et des outils propres à leur métier. Ce qui pose le problème du choix du moment et des bases sur lesquelles on décide de séparer les parties matérielle et logicielle

Par Vincent Perrier,

CoFluent Design

Cofondateur et directeur associé de la société française CoFluent Design, créée en 2003, Vincent Perrier est ingénieur diplômé en informatique industrielle. Il commença sa carrière comme développeur logiciel, puis rejoignit Telelogic en tant que consultant. Il passa ensuite cinq ans chez Wind



River Systems, dont trois ans aux Etats-Unis en tant qu'ingénieur d'application et responsable marketing.

d'un système et de définir les interconnexions entre les différents composants de ce système. Logiquement, cette étape est abordée lors de la phase de définition de l'architecture d'un système. Cependant, il importe de ne pas effectuer cette séparation de manière

prématurée sans étude préalable, car on s'expose alors à des difficultés dans le développement du système et de sa maintenance. Il est notamment essentiel de disposer d'une représentation détaillée et validée du comportement des fonctions du système et de ses performances, sous la forme d'un modèle abstrait, avant de déterminer lesquelles seront implantées en logiciel et matériel.

L'expérience et les statistiques des analystes (notamment VDC, Gartner Dataquest, Collett International Research) montrent que près de la moitié des projets de systèmes embarqués sont en retard, un peu moins de 20% sont annulés et plus de 60% des premières moutures de puces sont erronées. Les principales causes de ces difficultés sont identifiées : elles sont liées à des erreurs fonctionnelles, une mauvaise interprétation des spécifications et/ou une difficulté à gérer la complexité ou le changement.

Ainsi, il est probable qu'une dichotomie logicielle/matérielle intervenant trop tôt dans la conception des systèmes joue un rôle important dans les retards et annulations des projets. En effet, si on réalise cette opération trop en amont d'un projet, on réduit la vision globale du système à deux entités séparées et l'on contribue alors aux différents facteurs de retard et d'annulation dudit projet. Une fois le lien avec le système cassé, les spécifications et les fonctionnalités globales sont perdues de vue et toutes sortes de dérives peuvent apparaître. Les parties « hard » et « soft » évoluent alors chacune de leur côté et leur intégration finale peut donner un système qui ne respecte pas les spécifications de départ. Construites indépendamment, ces parties sont en outre de plus en plus difficiles à maintenir dans la vision d'ensemble du cycle de vie d'un système.

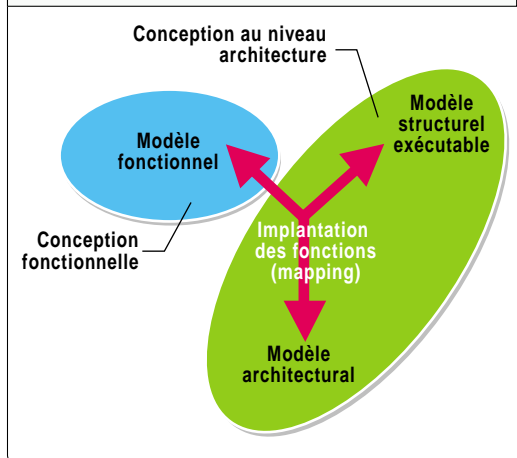
D'une vision « hard/soft » à une approche application/plate-forme

Une des difficultés actuelles dans ce procédé de partition entre le matériel et le logiciel est que la frontière entre les deux

Modèle de développement en « Y »

FIGURE 1

Dans un modèle de développement en « Y », la modélisation de l'application et celle de la plate-forme sont réalisées séparément, avant de les rassembler pour former l'architecture finale.



est devenue de plus en plus floue et variable. D'autant plus que les avancées technologiques actuelles indiquent que dans un futur proche il sera aisé de décider, à partir de la même description d'une fonction, de l'implanter indifféremment en logiciel ou en matériel. La seule différence résidera alors dans la nature du « moteur d'exécution ». Soit les bits issus de la traduction de la fonction en code logiciel sont interprétés par le « processeur logiciel » à partir de son jeu d'instructions; soit les bits issus de la traduction de la fonction en code matériel servent à programmer (ou implanter) directement le « processeur matériel ».

Bien que l'état de l'art actuel des technologies informatiques n'autorise pas encore cette manière de travailler, il existe une alternative à la traditionnelle dichotomie « hard/soft ». Il s'agit de considérer un système comme étant la composition d'une application s'exécutant sur une plate-forme. En apparence cette vision n'a rien de révolutionnaire. Mais ce couple « application/plate-forme » appliqué non seulement en phase de spécification, comme c'est souvent le cas, mais aussi en phase de conception, comme c'est encore rare, permet d'assurer une meilleure maintenance, une meilleure évolutivité et une meilleure adaptabilité des systèmes. De plus, cette approche apporte plus de souplesse et d'efficacité dans les développements. Ces avantages, ainsi que la satisfaction des fonctionnalités au meilleur rapport performances/coûts, sont possibles grâce à l'identification et à la séparation claire des fonctionnalités du système et de ses ressources d'exécution, qui toutes deux peuvent être logicielles et/ou matérielles. Une fois seulement cette différence entre application et plate-forme établie, le partitionnement

I.- Les types d'outils de modélisation d'un système

Modélisation & simulation	Outils pour les logiciels embarqués MDA/MDD (Analyse au niveau logiciel)	CoFluent Studio SLD (Analyse au niveau système)	Outils de CAO ESL (Analyse au niveau plate-forme SoC)
Niveau application	Logiciel (C/C++)	Logiciel + Matériel (C/C++/SystemC)	
Niveau plate-forme			Matériel (SystemC)
Niveau comportemental	Atemporelles	Temporelles/messages	Transactionnelles/précises au cycle près
Niveau architectural		Niveau application (Analyse de type prospective)	Niveau plate-forme seul (Analyse de type confirmative)

entre logiciel et matériel peut dès lors se faire sur la base d'une analyse de performances. Attention, il ne faut pas confondre ici le mot application, pris au sens de « l'ensemble des fonctions logiques du système », avec l'application logicielle embarquée. Car toute application n'est pas forcément logicielle. De la même manière, toute plate-forme n'est pas nécessairement matérielle (par exemple, une machine virtuelle Java). C'est de cette confusion que proviennent la plupart des difficultés de spécification et d'intégration entre matériel et logiciel car la fonctionnalité du système (son application) n'a jamais été abordée d'un point de vue global. Mais pour ce faire, il faut utiliser un modèle de l'application qui permette de décrire des fonctions qui pourraient être aussi bien implantées en « hard » qu'en « soft ».

Ainsi, dans cette approche, si les fonctions du système servent de fil conducteur et sont validées tout au long des phases de développement, il est possible d'éviter toute erreur fonctionnelle et une mauvaise interprétation des spécifications. De plus, si l'application est développée et évolue de manière séparée et indépendante de la plate-forme, il est plus

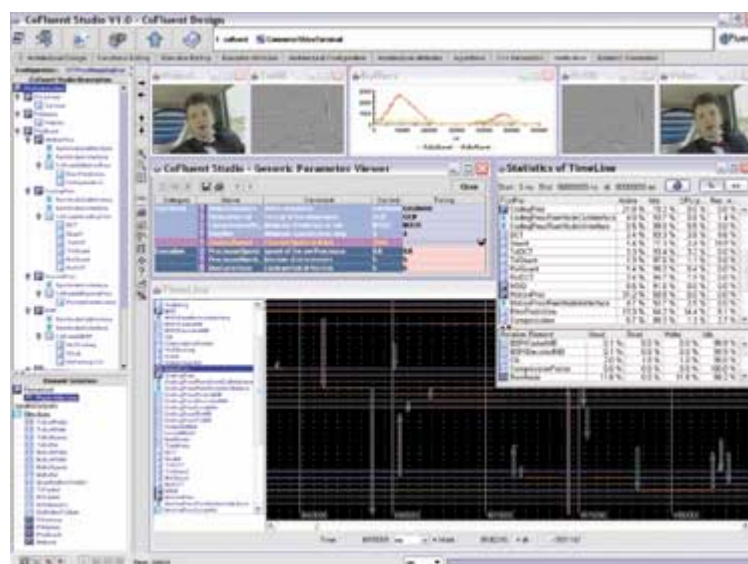
aisé de gérer le changement et d'éventuels ajouts de fonctions ou de mises à jour de matériel ou même de créer des produits dérivés à performances/fonctionnalités/coûts différents.

Savoir modéliser avec les bons outils

Il est donc essentiel d'aborder la conception de l'application d'un point de vue fonctionnel pur, sans idée préconçue quant à son implantation en matériel ou en logiciel. Une fois l'application définie, certaines parties (fonctions) peuvent se retrouver sous forme matérielle et donc faire partie de la plate-forme matérielle. De même, il est essentiel de repousser toute autre contrainte technologique ou économique (bus, protocole réseau, chipset, API logicielle, langage de programmation, système d'exploitation, compilateur, driver...) à la phase de définition de l'architecture. La conception d'un système peut être ainsi vue comme :

- la modélisation de son application, indépendamment de toute considération technologique ou économique;
- la modélisation de sa plate-forme d'exécution.

▼ A.- La modélisation et la simulation de l'application avec ses contraintes temps réel permettent d'analyser l'évolution temporelle des variables, des profils d'exécution des algorithmes, des messages de débogage...



▲ B.- La simulation au niveau architectural permet d'adapter la plate-forme pour respecter des contraintes de coûts/performances, avec par exemple des diagrammes de séquence temporelle ou des mesures de performances.

tion (ensemble de ressources permettant d'exécuter l'application) avec la prise en compte des contraintes technologiques et économiques;

- le déploiement (ou mapping) des fonctions de l'application sur les ressources de la plate-forme avec prise en compte des contraintes technologiques et économiques.

Ce processus, aujourd'hui largement reconnu dans l'industrie, est souvent présenté comme un modèle de développement dit en « Y » qui sépare la modélisation de l'application et de la plate-forme dans chacune de ses branches, pour ensuite les réunir et former le modèle d'architecture du système final (figure 1).

Doté de cette nouvelle façon d'analyser les problèmes et les solutions, il devient alors plus facile de se repérer dans la multitude d'offres d'outils de conception au niveau système. Chaque outil se base en effet sur sa propre vision et sa propre définition d'un système adapté aux problèmes qu'il résout et aux utilisateurs qu'il cible.

Pour simplifier, existent aujourd'hui sur le marché deux grandes catégories d'outils. Ceux issus de la sphère des spécifications et de la modélisation et ceux issus de la CAO électronique. Les premiers ont aujourd'hui tendance à descendre dans le cycle de conception pour atteindre l'application logicielle embarquée seule. On peut citer par exemple les outils de modélisation mathématique, les outils de modélisation à base des langages UML et SDL, les outils d'analyse à base d'automates ou de diagrammes d'états. Les seconds, quant à eux, font le chemin inverse et ont tendance à remonter les niveaux d'abstraction, en passant du niveau registre (RTL pour Register transfer level) au niveau transaction (TL pour Transaction level) pour adresser la conception de la plate-forme matérielle seule. On peut citer par exemple les outils de type « TLM » (Transaction level

modeling) ou « Platform-based design » qui ciblent essentiellement le domaine particulier des systèmes sur une puce.

Ainsi, avec une vision orientée « hard/soft » on pourrait penser que ces outils couvrent tout le spectre de la conception au niveau système, bien qu'ils se basent encore sur des standards et pratiques divergentes (VHDL/SystemVerilog et SystemC pour la conception des blocs IP et de la plate-forme matérielle, C/C++ côté application logicielle embarquée). Mais ce n'est pas vraiment le cas. Car si l'on raisonne avec une vision orientée application/plate-forme, on se rend compte, qu'un fossé marqué existe encore entre les outils de CAO et les outils de développement logiciel (tableau I).

Du côté de la modélisation et de la simulation comportementale, dont l'objectif est de décrire le comportement (les fonctionnalités) du système, il existe un manque entre les modèles UML fonctionnels atemporels (sans prise en compte des contraintes de temps) et les modèles transactionnels basés sur des cycles de lecture et écriture de bus. Une solution consiste à ajouter des simulateurs de jeux d'instruction (ISS pour Instruction set simulators) aux modèles transactionnels afin d'exécuter le code logiciel pour le processeur choisi, mais cela nécessite l'obtention du code final, ralentit les simulations et augmente sensiblement le coût des outils. C'est typiquement une solution suivant la vision orientée « hard/soft » et donc acceptable seulement une fois la séparation application/plate-forme effectuée.

Si l'on regarde maintenant du côté de la modélisation et de la simulation architecturale, dont l'objectif est de décrire l'architecture du système afin de réaliser toutes les fonctionnalités au meilleur rapport performances/coûts, seule la plate-forme bénéficie d'une étude d'architecture alors que bien

souvent la phase d'architecture côté logiciel embarqué se limite à compiler et à télécharger le code sur la cible.

Ce fossé entre outil de CAO et outils de développement logiciels tend cependant à être comblé aujourd'hui par des outils comme CoFluent Studio, environnement de modélisation de système (ou SLD System-level design) qui réunit application et plate-forme, et qui s'inscrit à la charnière entre les outils de CAO dits ESL (Electronic system-level) orientés plate-forme matérielle et les outils de modélisation de logiciel embarqué de type MDA/MDD (Model-driven architecture/design) (tableau I).

Lier application et plate-forme

Un système, quel que soit son type de modélisation, est décrit en fait sous trois angles : le niveau structurel/organisationnel, le niveau comportemental et le niveau communications. La vue la plus significative pour caractériser le niveau d'abstraction d'un modèle, quel qu'il soit, est celle des communications. Ainsi, on peut distinguer cinq niveaux d'abstraction pour décrire ces communications au sein d'un système : les niveaux service, message, transaction, transfert et registre. Avec CoFluent Studio, le niveau de modélisation « message » permet une conception de haut niveau rapide et facilement transposable au niveau transactionnel (tableau II). L'environnement permet en outre de modéliser l'application avec sa plate-forme, en intégrant du code logiciel C/C++ existant ou des blocs IP matériels écrits en SystemC, moyennant une adaptation des modèles. De plus, l'introduction des contraintes de temps permet de modéliser et de vérifier efficacement les aspects temps réel de l'application (figure 2).

Enfin, l'étude d'architecture prospective autorise une définition de la plate-forme dans

ses grandes lignes à partir de l'application, pour ensuite la raffiner dans une étude plus confirmative à un niveau inférieur, typiquement le niveau RTL (tableau II).

D'un point de vue pratique, avec l'outil CoFluent Studio for Timed-Behavioral Modeling, il est possible de modéliser – essentiellement de manière graphique et, si nécessaire, à l'aide de code C pour les parties séquentielles élémentaires – le comportement global du système dans son environnement et dans le temps. A l'aide de notations simples et intuitives, adaptées aux

II.- Les différents niveaux d'abstraction de la communication d'un système (1)

Caractéristiques des modèles de communication

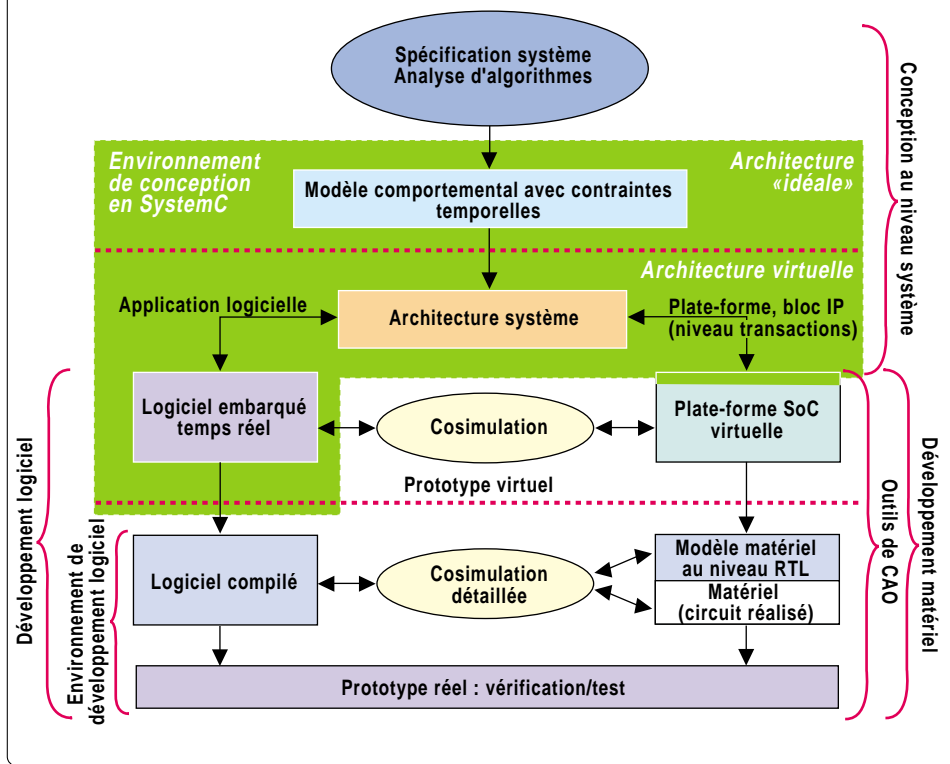
Niveaux d'abstraction	Nature	Media	Adressage	Protocole	Données	Comportement temps réel
Service (ex: Corba)	Fonctionnel atemporel	Réseau abstrait	Automatique	Service de requêtes : « do something »	Abstraites	Non prédictible (service /requêtes causales)
Message (ex: MCSE (2), SDL, UML)	Fonctionnel temporel (ou atemporel)	Réseau logique sur canaux actifs	Abstrait et explicite	Primitives de haut niveau : « send/receive »	Abstraites	Possibilité de prédictibilité (séquence des messages)
Transaction (ex: CSP, SystemC, Cossap)	Précis au niveau du cycle du bus	Liens logiques	Logique et explicite	« Read/write », « Wait for new event »	Taille variable	Synchronisation au cycle du bus
Transfert (ex: VHDL, Verilog)	Précis au niveau cycle	Liens logiques	Logique et explicite	« Read/write », « Wait for new clock cycle »	Taille fixe	Synchronisation à l'horloge du cycle
« Register transfer » (ex: VHDL, Verilog)	Précis au niveau des broches des transistors	Liens physiques	Physique et explicite	« Set/reset ports », « Wait for new clock cycle »	Bit/données sur le bus	Synchronisation à l'horloge du cycle

(1) Source: La spécification et la validation des systèmes hétérogènes embarqués. A. Jerraya et G Nicolescu, Hermes, collection Techniques de l'ingénieur. (2) MCSE: Méthodologie de conception des systèmes électroniques.

Processus de développement avec l'outil CoFluent Studio

FIGURE 2

Le concept d'environnement de conception d'un système, réalisé avec l'outil CoFluent Studio, s'inscrit dans le cadre global de la conception de systèmes ou de circuits, avec comme objectif une exploration sans a priori de différents modèles de partitions entre le logiciel et le matériel.



pratiques des ingénieurs système (qu'ils travaillent au niveau logiciel ou matériel), l'application est décrite sous la forme de tâches ou de processus (appelés fonctions) concurrents et communicants (synchrones et asynchrones), et ce de manière indépendante de toute considération technologique (la nature physique du système, le matériel...). La description de l'environnement permet ensuite de bâtir des scénarios de test de l'application pendant la simulation. Au cours du développement, l'ensemble des contraintes de temps est spécifié, qu'il s'agisse du temps d'exécution des algorithmes, des temps de communications et des synchronisations. Enfin, pour détecter facilement les erreurs fonctionnelles et de temps réel, des outils de visualisation complets assurent l'observation du comportement du système (photo A).

Au sein de l'outil CoFluent Studio for System Architecting, qui inclut la partie Timed-Behavioral Modeling, on peut rechercher et définir la meilleure architecture système en fonction de ses propres critères : coûts, performances, réutilisation, etc. La plate-forme matérielle du système (ou structure d'exécution) est modélisée graphiquement à partir de composants matériels universels (processeurs logiciels ou matériels, nœuds de communication, mémoires partagées), dont les critères macroscopiques (vitesse, temps d'accès, degré de concurrence) permettent de

caractériser individuellement les performances. Aucun modèle particulier de microprocesseur ou de bus n'est nécessaire. Quant aux fonctions du système et leurs éléments de communication/synchronisation, elles sont ensuite réparties sur les différentes ressources de la plate-forme par une simple manipulation du type « glisser-déposer » sur l'écran. Ainsi, autant de plates-formes que de stratégies d'allocations peuvent être analysées. Ce cycle rapide d'exploration d'architecture interactif permet notamment de détecter l'architecture optimale, tout en analysant les résultats de simulation et l'impact du matériel sur le comportement du système et ses performances (photo B).

Au final, les développeurs système disposent alors de spécifications exécutables de l'application complète du projet dans son environnement, qui servent de modèle de référence commun pour les développeurs du matériel et du logiciel. Ces derniers ont aussi à leur disposition la description à haut niveau de la plate-forme matérielle optimale en terme de processeurs, mémoires partagées et leurs interconnexions qui servent de spécifications pour les développeurs de matériel. Enfin, les développeurs ont une plus grande assurance sur la capacité du système à respecter les exigences du cahier des charges, tant du point de vue fonctionnel que des performances et des coûts. ■