DÉVELOPPEMENT AU NIVEAU SYSTÈME

Mobiles: l'exploration d'architecture renforce une méthode *«orientée service»*

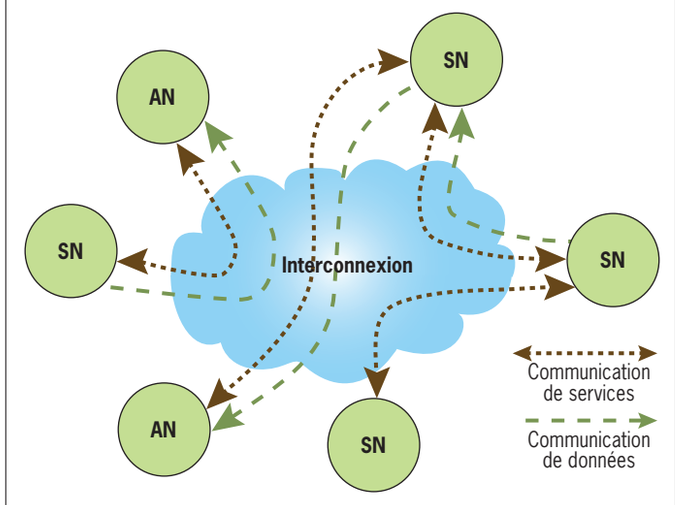
Une étude sur l'architecture d'un mobile, effectuée par Nokia, a permis de réaliser des liens entre modèles de spécifications basés UML et architecture d'exploration non UML, sous forme de traces d'exécution. L'intégration d'outils au moyen de méta-modèles implantés par Telelogic et CoFluent autorise l'emploi de modèle «use case» à la place des traces d'exécution.

NoTA (Network-on-terminal architecture) est une architecture modulaire orientée services et centrée interconnexions qui s'adresse aux plates-formes actuelles et futures de terminaux mobiles. Développée par le centre de recherche de Nokia (Nokia Research Center, NRC), NoTA apporte une plus haute performance et une «horizontalisation» très efficace des éléments d'architecture d'un système en facilitant les processus d'intégration (encadré I). La méthode de développement qui lui est associée donne notamment l'assurance que les designs sont vérifiables pas à pas vis-à-vis des spécifications d'utilisateurs. La méthode NoTA autorise en outre une réutilisation à plusieurs niveaux.

Concrètement, une plate-forme NoTA consiste en services connectés par des liens lâches et s'exécutant sur des sous-systèmes hétérogènes. Dans les systèmes basés NoTA, les communications de services et de données sont routées à travers la pile de protocoles réseau. La méthode NoTA repose sur un flot de développement de plate-forme assurant que les services, les sous-systèmes et la topologie d'interconnexion correspondent bien aux spécifications de l'utilisateur final. Elle fournit aussi des spécifications formelles réutilisables pour chaque entité de la plate-forme. Pour ce faire, l'architecture logique NoTA repose sur trois types d'éléments de base appelés «nœuds applicatifs» (AN), «nœuds de service» (SN) et «interconnexion» (Interconnect) (figure 1).

Routage des communications dans le système NoTA — FIGURE 1

On voit sur cette représentation comment les communications de services et les communications de données sont organisées au sein d'une plate-forme NoTa.



NoTA définit aussi deux grands niveaux de protocoles pour l'interconnexion, «H_IN» et «L_IN». Le premier, «H_IN», est une pile de haut niveau qui apporte la fonctionnalité de communication aux services et applications de la plate-forme. Le second, «L_IN», est un protocole de bas niveau qui prend en charge la connexion physique entre sous-systèmes. Enfin, un sous-système NoTA comprend un ensemble de services. Il s'agit ici d'un concept architectural qui ne s'aligne pas nécessairement sur les limites d'un circuit intégré (figure 2).

Développer et modéliser l'architecture d'une plate-forme

Les pratiques habituelles en matière de développement d'architectures de plates-formes sont informelles, et s'appuient largement sur l'expérience de l'architecte système. Des tableurs sont souvent utilisés pour projeter des résultats à partir de précédents designs, ce qui est faisable lorsque les modifications entre

I.- Un centre de recherche et un projet européen

→ **Nokia Research Center (NRC) est une division autonome de Nokia qui ne se rattache à aucune unité spécifique de développement de produit.**
NRC dispose de plus de 1 100 chercheurs, focalisés sur des sujets de recherche liés à la mobilité dans les domaines voix, données, gestion et communications. Dans le cadre du projet de recherche européen ITEA Martes (Model-based approach to real-time embedded systems ; www.martes-itea.org), NRC conduit une étude de terminal mobile axée sur les architectures de terminaux centrés communications et conçus pour la convergence numérique. Dans ce contexte, NRC a adopté la méthode MCSE (Méthodologie de conception des systèmes électroniques) de CoFluent Design pour la modélisation d'architecture, et utilisé CoFluent Studio comme outil de modélisation de plate-forme de terminal mobile. NRC, qui mène d'importants travaux de recherche dans le domaine des architectures de terminaux mobiles, a développé en outre un concept d'architecture orientée services appelé NoTA (Network-on-terminal architecture). La contribution de Nokia au projet Martes repose étroitement sur NoTA.

générations successives d'architectures sont relativement mineures. L'approche informelle devient problématique en présence de concepts architecturaux réellement innovants, requérant une exploration systématique d'alternatives très différentes les unes des autres. En outre, les spécifications de plates-formes sont généralement exprimées en des termes techniques qui ont peu de rapport avec les besoins de l'utilisateur final.

La méthode de développement d'architecture de plate-forme NoTA a pour but de surmonter ces pièges des pratiques informelles, grâce à une démarche systématiquement dirigée par les spécifications. Cette approche repose sur plusieurs caractéristiques.

La séparation des problèmes : l'objectif affiché ici est d'offrir la possibilité de travailler sur les différents aspects du système indépendamment les uns des autres, de faciliter le principe de réutilisation d'un design et de mieux gérer la complexité d'un projet. Ainsi dans les spécifications de l'utilisateur final, la fonctionnalité de la plate-forme (à savoir, les services qu'elle délivre), son architecture (c'est-à-dire définition des sous-systèmes et de l'infrastructure de communication) et l'implantation des sous-systèmes (matériel et logiciel) et protocoles d'interconnexion sont des domaines bien séparés. Chacun d'entre eux dispose de modèles autonomes qui, dans le système final, sont mis en relation les uns avec les autres.

Une approche d'ingénierie dirigée par les modèles : dans la méthode NoTA, les objets développés aux différentes phases du processus sont des modèles dont la sémantique est bien définie. Cette approche vise à éviter les malentendus et les erreurs engendrés par l'ambiguïté d'une documentation informelle. La démarche imposée par Nokia exige aussi que des outils d'analyse, de vérification, de transformation, de génération de code et de synthèse, pouvant être mis en œuvre sur les modèles, soient disponibles.

La réutilisation des modèles : Nokia est convaincu qu'une réutilisation efficace des modèles, dans différents contextes, améliore considérablement la productivité par rapport aux méthodologies conventionnelles. Avec la méthode NoTA, les différents types de modèles sont stockés dans des référentiels. Ils peuvent être récupérés et réutilisés pour composer de nouvelles configurations système.

Une validation et une vérification précoces : l'une des motivations de l'ingénierie dirigée par modèles est de valider et de vérifier très tôt les spécifications et la conception. Dans la méthode NoTA, les processus de validation et de vérification débutent dès la phase de spécification des besoins de l'utilisateur final, avec les modèles exécutables de « cas d'utilisation », communément appelés « use cases ». Plus tard, l'accent est mis sur la validité de la spécification de plate-forme et de l'analyse de performance, pendant les phases de spécification et d'implantation. Au sein de la méthode NoTA, la validation et la vérification ne se limitent pas à la logique, mais couvrent également des aspects non fonctionnels comme la performance temps réel et la consommation d'énergie.

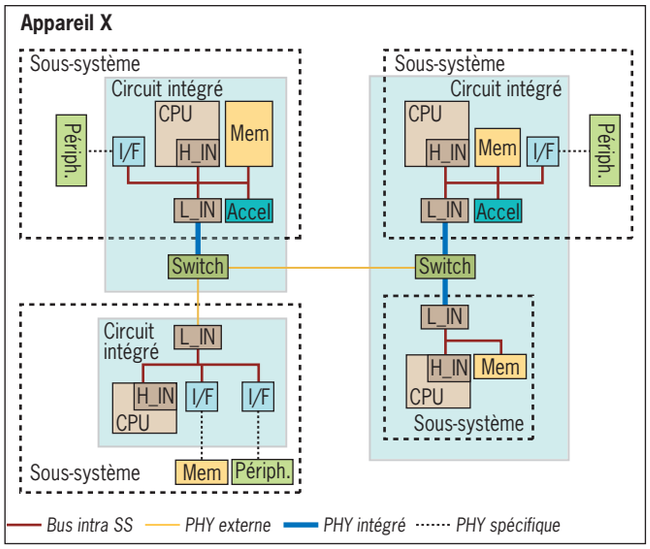
Pour cette méthode NoTa, les ingénieurs du centre de recherche de Nokia ont adopté la solution de modélisation d'architecture de CoFluent Design. Ainsi, grâce à l'outil CoFluent Studio (1) un modèle d'architecture est développé en construisant d'une part l'architecture fonctionnelle (sous forme d'un modèle comportemental temporel, c'est-à-dire un modèle fonctionnel du système doté d'informations de timing), d'autre part l'architecture de plate-forme (c'est-à-dire la structure d'exécution). L'étape suivante consiste à établir une correspondance (mapping) entre les blocs fonctionnels et la structure d'exécution. En termes de spécifications, celles d'une plate-forme NoTA découlent des besoins de l'utilisateur final, exprimés à l'aide de modèles de « cas d'utilisation ». L'ensemble retenu de ces « use cases » est d'abord étudié, et tous les services mis en œuvre dans les modèles « use cases » détaillés, appelés services primaires, sont identifiés. Puis, cet ensemble de services est réduit afin de minimiser recouvrements et redondances. Le processus aboutit in fine à la définition de l'ensemble de services primaires nécessaires.

Les modèles de « use cases », avec l'ensemble associé de services primaires nécessaires, servent ensuite à construire le modèle d'architecture fonctionnelle. Celui-ci consiste en des modèles de nœuds de services (notés SN) et des modèles de nœuds applicatifs (notés AN). Un modèle SN représente une instance d'un service, et il peut exister plusieurs instances du même service. Un modèle AN définit la manière dont l'application utilise les services dans une séquence de « cas d'utilisation » particulière.

Dans une plate-forme NoTA, un service est spécifié dans un format spécial appelé SIS (service

Sous-systèmes et circuits intégrés d'un terminal mobile — FIGURE 2

On le voit ici, plusieurs sous-systèmes peuvent coexister sur une puce, et un sous-système peut dépasser les limites d'une puce.



Quelques définitions

UML™ (Unified Modeling Language™)	Langage de modélisation graphique géré par l'OMG (Object management group)
Classe	En programmation objet, catégorie d'objets ayant des propriétés communes
Diagramme de structure composite	Diagramme UML décrivant les relations entre composants d'une classe.
Ingénierie dirigée par les modèles - Model Driven Engineering (MDE)	Paradigme de développement centré sur la modélisation de l'application à différents niveaux d'abstraction et sur la transformation des modèles, pour aboutir à une implémentation.
Mapping	Mécanisme de transformation des éléments d'un modèle en éléments d'un autre modèle.
Méta-modèle (modèle de modèle)	Description en langage standard (type Ecore) d'un langage de modélisation et de sa sémantique
Modèle	Abstraction d'un objet permettant son étude
Transformation horizontale	Transformation de modèles, où la source et la destination ont le même niveau d'abstraction.
Use case - Cas d'utilisation	Notation UML décrivant les besoins de l'utilisateur. Décrit un ensemble d'actions réalisées par le système en réponse à l'action d'un « acteur » (opérateur, autre système…).

(1) CoFluent Studio est un outil de modélisation graphique pour l'exploration architecturale et l'analyse de performance au niveau système. Il supporte le langage SystemC et fonctionne sous Eclipse.

interface spécification) qui inclut la signature d'interface du service et une description de son comportement vu de l'extérieur, sous forme de machine à états finis. Ce format SIS englobe aussi les attributs non fonctionnels significatifs du service, comme le timing et la consommation de puissance. Pour la modélisation d'architecture, les modèles SN sont dérivés directement de la description SIS. Les modèles AN sont dérivés des traces d'exécution des modèles «use cases».

Enfin, le modèle d'architecture de la plate-forme consiste en un assemblage de blocs représentant les sous-systèmes et des commutateurs de routage (RS). Lorsque la correspondance entre les nœuds SN et AN et les sous-systèmes est établie, ainsi que la topologie du réseau de communication entre sous-systèmes, le modèle d'architecture final est obtenu. La fonctionnalité de nœud d'interconnexion (IN) est alors intégrée dans les composants du modèle d'architecture de plate-forme (figure 3).

Modèle fonctionnel et de plate-forme

Le modèle fonctionnel, ou modèle comportemental temporel, décrit la partition logique et le comportement du système. Dans NoTA, le modèle fonctionnel est constitué par l'AN et les SN. Les SN incluent tous les services primaires du use case et tout service secondaire additionnel utilisé par les services primaires.

L'éditeur fonctionnel de CoFluent Studio est ici employé pour saisir la description graphique des SN et de l'AN. Le modèle interne de chaque SN est dérivé directement de la description SIS correspondante. Le comportement de l'AN est défini par la séquence «use case», en utilisant la trace générée par son exécution (en format XML). Les requêtes de services aux nœuds SN sont modélisées en tant que messages et, dans NoTA, toutes ces requêtes de services sont transmises à l'IN. Le comportement fonctionnel du système comprenant le nœud AN, les nœuds SN et un nœud IN idéal peut être simulé indépendamment, sans se préoccuper d'une architecture de plate-forme définie.

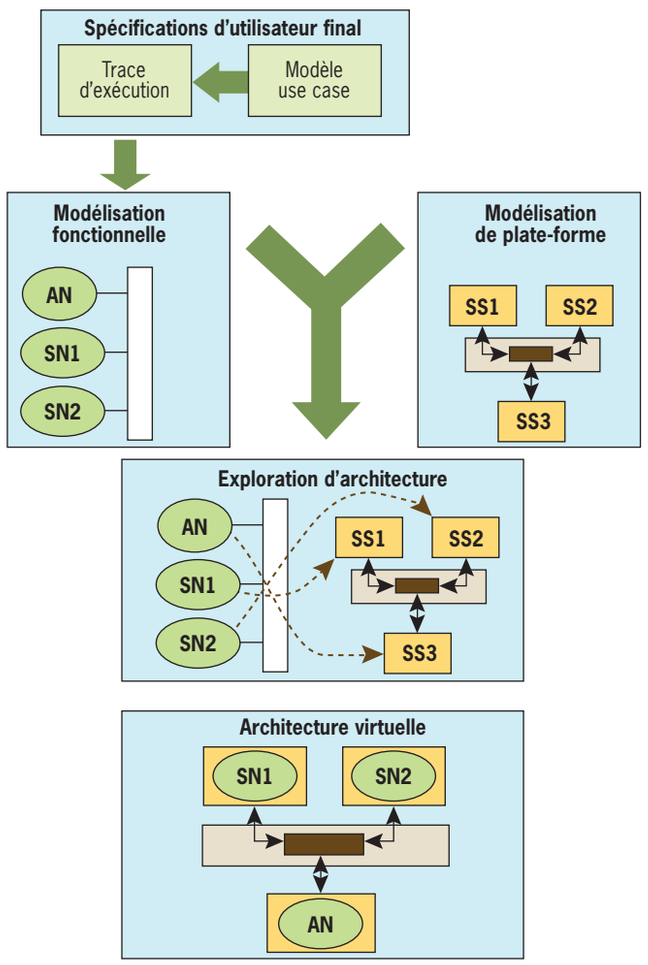
Dans NoTA, le comportement des SN est modélisé sous forme de machines à états finis. Au sein d'un modèle CoFluent, ils sont représentés soit sous forme de modèles SystemC, soit sous forme de structures fonctionnelles. Ces dernières sont des modèles de niveau hiérarchique inférieur utilisés comme boîtes noires dans un modèle CoFluent. Le comportement des fonctions CoFluent peut être défini plus en détail avec des algorithmes C ou C++. Nokia a adopté cette dernière approche, en ajoutant des algorithmes C++ pour lire et interpréter les fichiers XML. Un module d'analyse syntaxique (parser) extrait l'information nécessaire du modèle XML et lui assigne l'emplacement voulu dans le formulaire de SN de CoFluent.

Il y a deux types principaux de communication dans le réseau NoTA. Des modèles de communication d'objets données sont ajoutés aux SN afin d'avoir une idée de la quantité de données échangées entre deux nœuds. En pratique, le même lien fonctionnel est utilisé pour la communication de services et le trafic de

Suite p.42

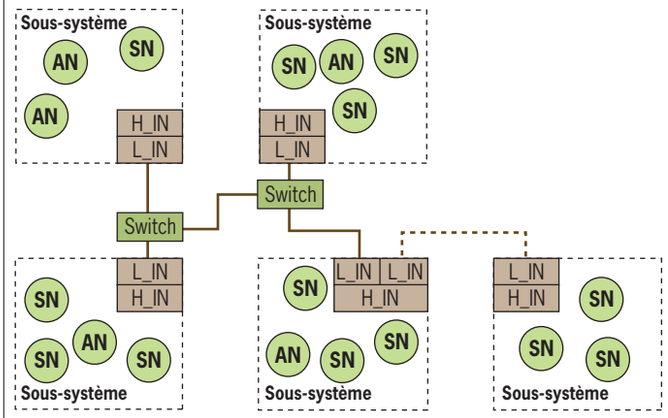
Modélisation d'un système d'architecture NoTA — FIGURE 3

On voit ici comment l'approche de CoFluent Design est appliquée à l'architecture NoTA en utilisant CoFluent Studio comme outil de modélisation de plate-forme.



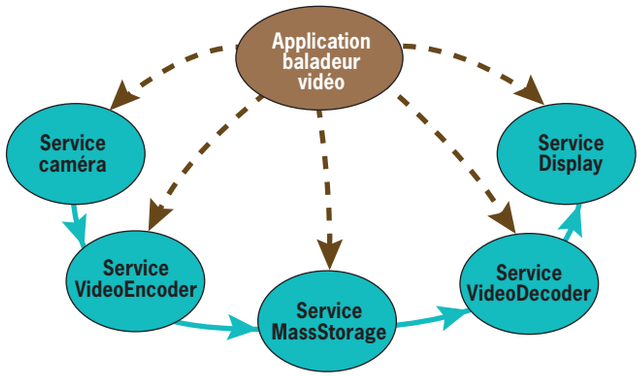
Topologie de réseau de sous-systèmes — FIGURE 4

Cette topologie de réseau représente l'interconnexion nécessaire pour les simulations architecturales



Exemple d'un baladeur vidéo — FIGURE 5

L'application «Baladeur vidéo» est décrit ici par cinq classes dans l'outil Tau de Telelogic: Camera, Display, VideoEncoder, VideoDecoder et MassStorage.



II.- Les étapes d'une exploration architecturale

➔ Les étapes d'une exploration architecturale :

1. Définir le nombre et les types de sous-systèmes.
2. Définir la topologie d'interconnexion entre sous-systèmes. Ceci détermine le nombre et les types de commutateurs de routage.
3. Etablir les correspondances entre SN et AN et les sous-systèmes.
4. Exécuter les simulations vis-à-vis des use cases d'origine.
5. Analyser les résultats.
6. Retourner à l'étape 3 pour optimiser l'architecture courante.
7. Retourner à l'étape 1 pour essayer différentes architectures.
8. Choisir le(s) cas le(s) plus optimisé(s).
9. Sélectionner l'architecture virtuelle.

Lorsque l'exploration des architectures avec CoFluent Studio est terminée et que le cas optimal est sélectionné, la solution d'architecture se présente ainsi :

- Un ensemble de sous-systèmes connectés avec une certaine topologie.
- Une allocation des services issus des use cases décomposés dans les sous-systèmes mentionnés.
- Des paramètres de performance vérifiés et affinés pour les services.
- Tout ce qui précède est vérifié par rapport aux use cases d'utilisateur final qui ont été donnés au départ.
- Tous ces éléments définissent les spécifications de chaque sous-système.

➤ données. Le type du lien et le message envoyé à travers lui sont modélisés par une classe C++. Cette classe contient des champs pour le routage, le type de message, sa taille et les sous-classes pour le contenu.
Le comportement du « use case » est importé dans le modèle CoFluent sous forme de fichier trace XML. Ce fichier consiste en une séquence de requêtes de services avec des paramètres additionnels éventuels. Le fichier est lu dans l'AN et les requêtes de services correspondantes sont envoyées aux services appropriés.
Le modèle de plate-forme, de son côté, est une représentation abstraite de l'architecture physique. CoFluent fournit un ensemble de blocs de base pour la construction d'une telle architecture : processeurs, mémoires partagées, signaux et connexions utilisant les nœuds de communication. A ce stade de la conception, les sous-systèmes sont esquissés sous forme d'emplacements pour les SN et l'AN. Un sous-système donné est donc constitué par un processeur exécutant les SN en parallèle. L'implémentation réelle du sous-système n'est pas modélisée. Le commutateur de routage (RS) est décrit par un modèle de routage contenant des fonctions de récupération de statistiques de performance. Un processeur est réservé pour chaque RS, et les sous-systèmes sont connectés au RS par les nœuds de communication. La topologie de réseau résultante représente l'interconnexion précise qui est nécessaire pour les simulations architecturales (figure 4).

Diagramme de structure composite — FIGURE 6

Ce diagramme de structure composite est élaboré dans l'outil Tau, et concerne ici le modèle applicatif VideoPlayer.

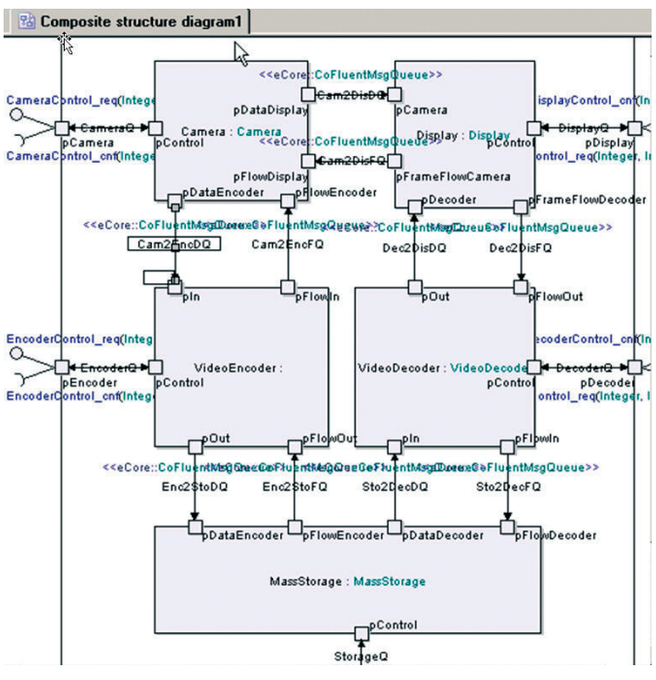
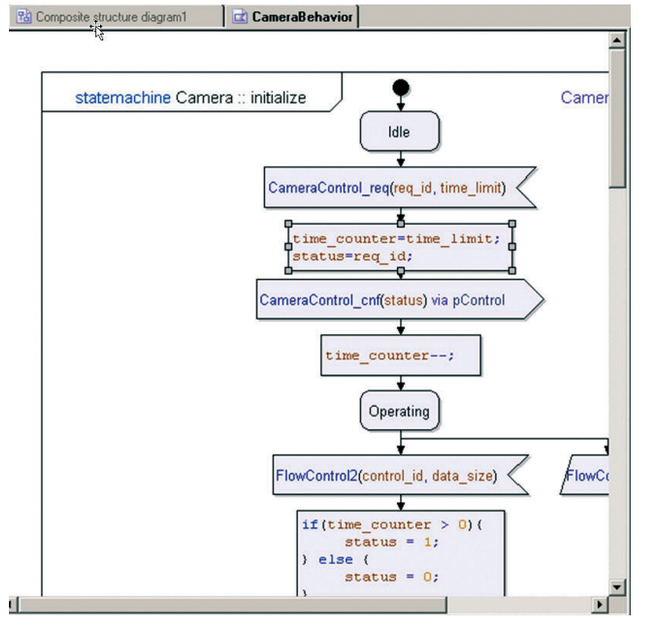


Diagramme d'états — FIGURE 7

Dans ce diagramme d'états on voit le comportement de la classe Camera.



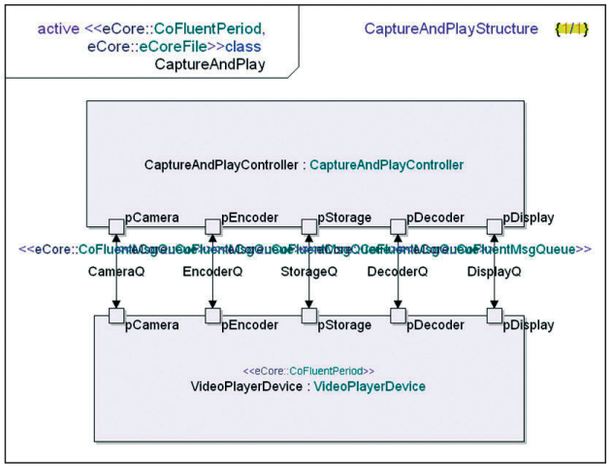
La conception d'architecture quant à elle comprend trois grandes phases : décider du nombre et du type de sous-systèmes inclus dans le terminal ; définir la topologie d'interconnexion entre les sous-systèmes ; et enfin définir l'allocation des nœuds SN et AN au sein des sous-systèmes (encadré II).
Un sous-système peut être vu comme un ensemble de SN dotés d'une connexion IN. Un sous-système peut contenir plusieurs SN. Par exemple, un sous-système de stockage est à même de servir de stockage de masse traditionnel ou de serveur de streaming média. Ces fonctions utilisent des services complètement différents, mais il peut s'avérer bénéfique de les localiser dans le même sous-système car elles emploient les mêmes ressources matérielles. Les SN sont susceptibles d'être distribués entre sous-systèmes de différentes manières. En conséquence, plusieurs configurations d'architecture peuvent être évaluées pour identifier les goulets d'étranglement potentiels et maximiser la performance système.
L'analyse du trafic réseau est un élément clé lors de la vérification de l'architecture qui a été conçue. Chaque architecture doit donc être simulée en fonction de tous les use cases. Certains paramètres associés aux SN, AN, IN et RS, comme la taille de tampons locaux, sont à optimiser durant la conception d'architecture.

Intégration avec des outils « sur étagère »

Dans le cadre de leurs travaux pour le projet Martes, Telelogic et CoFluent Design ont réalisé une intégration basée sur un méta-modèle de leurs outils à l'aide du framework EMF (Eclipse modeling framework) et de son format Ecore.
L'intégration offre une alternative intéressante pour le transfert entre outils du comportement

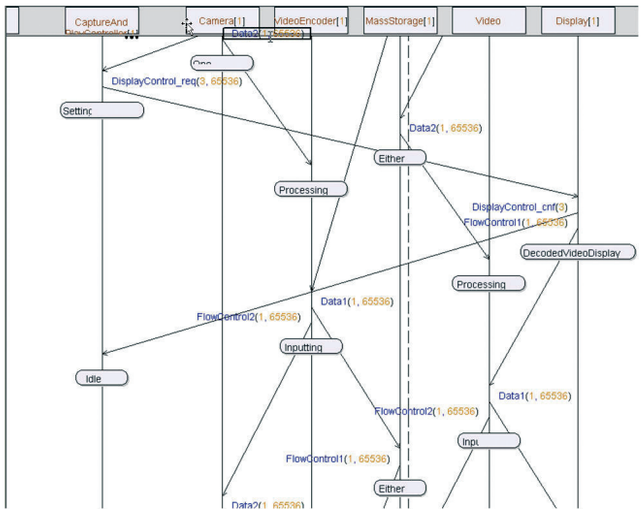
Diagramme de structure FIGURE 8

Ce diagramme de structure décrit le use case «Capture&Play».



Exécution de l'application dans Tau FIGURE 9

Cette simulation n'intègre pas les timings, car le simulateur Tau ne s'intéresse pas aux performances temps réel du système.



des «use cases». L'idée ici est de transférer le modèle «use case» exécutable en entier, au lieu de transférer sa trace d'exécution comme décrit précédemment. L'outil Telelogic Tau (2) est utilisé dans ce cadre uniquement pour le modèle applicatif. Des attributs de performance temps réel peuvent être ajoutés au modèle Tau sous forme de valeurs marquées («tagged values») du profil UML utilisé, et être traités en tant qu'adaptation du modèle d'application Martes. Le grand bénéfice tiré du transfert du modèle entier est que les utilisateurs sont à même de recevoir un retour du modèle d'architecture de plateforme qui affecte potentiellement le comportement du «cas d'utilisation».

Un exemple d'étude d'un baladeur vidéo est présenté ici pour montrer comment fonctionne ce principe de transfert de modèle (figure 5). Dans cet exemple, les services sont modélisés sous forme de machines d'états dotés d'attributs de performance. Le nœud d'application contient une machine d'états modélisant son comportement dans un «cas d'utilisation» particulier. La modélisation est faite sur deux niveaux, le niveau «use case» et le niveau

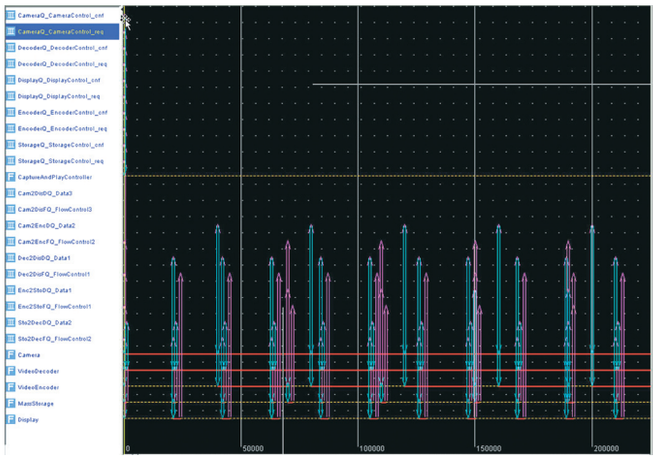
Suite p.44

(2) Telelogic Tau est un environnement de développement centré modèles, basé sur UML. Il permet notamment de valider les spécifications système avant leur transmission pour implantation. Le vérificateur de modèles fournit une trace XML de l'exécution du système.

Simulation de fonction

FIGURE 10

On voit ici le déroulement temporel d'une fonction simulée dans CoFluent Studio.

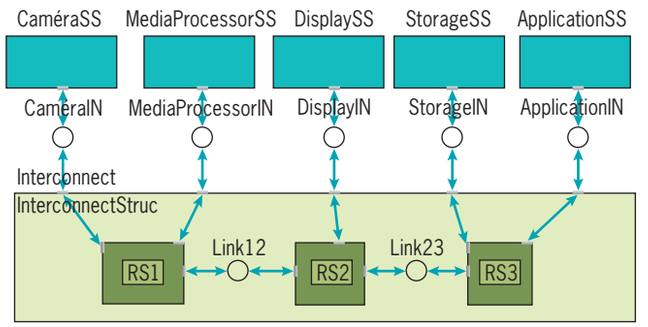


service. Plusieurs use cases peuvent être examinés en changeant le nœud d'application.
La structure de niveau supérieur du modèle d'application est définie dans Tau par un diagramme de structure composite (figure 6). Le comportement des classes est modélisé sous forme de machines d'états (figure 7). La composition du use case est également créée en UML (figure 8). Les stéréotypes du profil UML mis en œuvre dans l'intégration des outils contiennent des «valeurs marquées», qui permettent d'introduire des attributs de performance utilisés par CoFluent Studio. Le modèle résultant est exécutable et peut être simulé dans Tau (figure 9).
L'extension de l'outil Tau développé dans le cadre du projet Martes génère ensuite un fichier Ecore, conforme au méta-modèle défini. Ce fichier Ecore peut être importé dans CoFluent Studio, ce qui induit que la configuration de modèle d'application résultante dans CoFluent Studio a la même structure que le modèle Tau d'origine. La représentation graphique du modèle est perdue dans le transfert. Cependant, le comportement des machines d'états et les attributs de performance sont préservés. Le modèle applicatif peut alors être simulé dans CoFluent Studio (figure 10).
Le modèle de plate-forme d'exécution et le modèle d'architecture système sont ensuite construits dans CoFluent Studio. Le modèle d'architecture système est généré en créant les correspondances entre éléments de l'application sur le modèle de plate-forme d'exécution (figure 11). Le modèle obtenu peut alors être simulé dans CoFluent Studio pour étudier l'impact de la plate-forme et des choix d'allocation (figure 12).

Modèle de plate-forme d'exécution

FIGURE 11

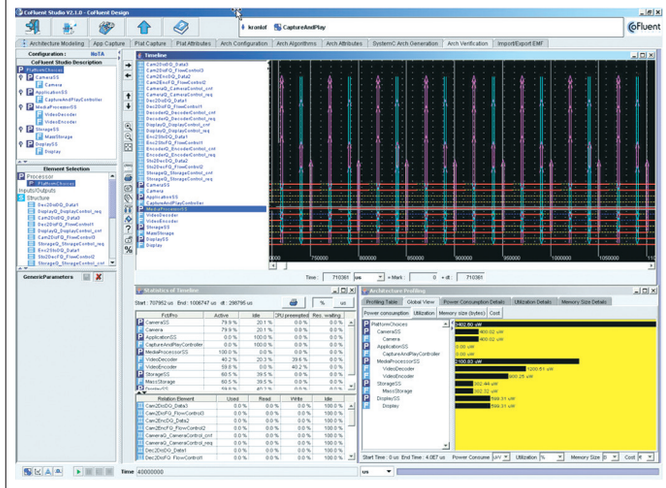
Ce modèle simple de plate-forme d'exécution de l'application dans CoFluent Studio montre une plate-forme dotée de cinq unités processeurs et de l'interconnexion, avec lequel l'ensemble du modèle applicatif peut être mis en correspondance.



Simulation d'architecture

FIGURE 12

On voit ici la simulation d'architecture de plate-forme dans CoFluent Studio. Celle-ci montre le séquencement de l'exécution des services VideoEncoder et VideoDecoder dans l'unique processeur du sous-système MediaProcessorSS.



Vincent Perrier (CoFluent Design) et Klauss Kronlöf (Nokia Research Center)