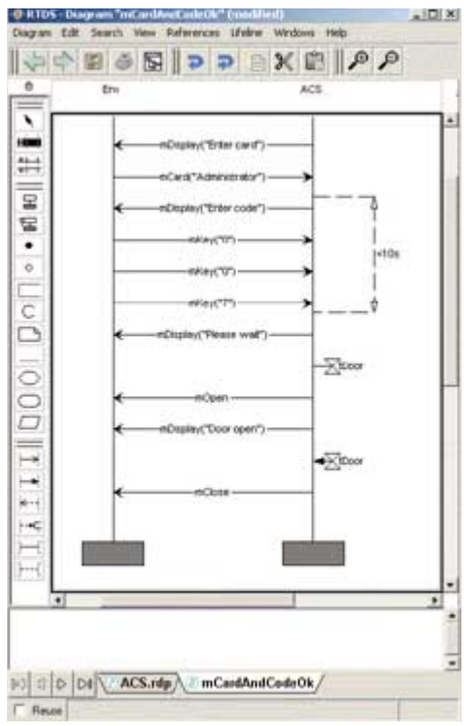
LANGAGE DE DÉVELOPPEMENT

SDL et UML : mariage de raison pour la conception des logiciels temps réel

Pour le temps réel, le langage SDL pose problème car il ne couvre pas certaines caractéristiques essentielles des systèmes déterministes. Pour compenser cette faiblesse, qui freine une diffusion plus large du SDL, PragmaDev propose l'environnement SDL-RT qui intègre le C, mais aussi le langage UML, la grande tendance du moment.

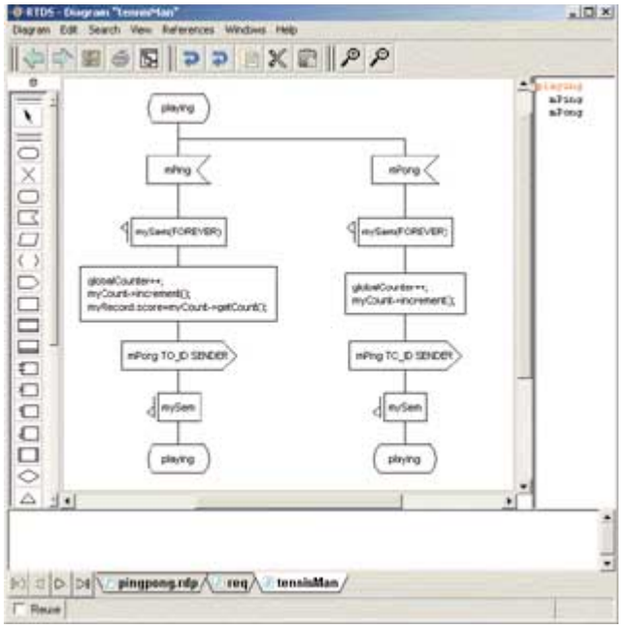
Les systèmes temps réel s'appuient sur un système d'exploitation temps réel (RTOS) dont l'élément structurant de base est la tâche. Plusieurs tâches s'exécutent en parallèle pour réaliser une fonction de base, fonctions de base qui sont ensuite regroupées pour réaliser des fonctions plus complexes et ainsi de suite jusqu'à couvrir toute l'application. Il est rapidement apparu que le langage SDL (pour Specification and description language), qui permet d'architecturer son application en regroupant les tâches en blocs fonctionnels qui, eux-mêmes, peuvent être regroupés en blocs de plus haut niveau, est un bon moyen de représenter graphiquement l'architecture de n'importe quel système temps réel.
Rappelons que ce langage a été initialement défini par l'Union internationale des télécommunications (ITU-T, International telecom union - Telecommunication standardization sector) pour spécifier les protocoles de télécommunications. L'expérience a montré que ses principes de base pouvaient être étendus à beaucoup d'autres domaines d'application du fait de cette approche fonctionnelle graphique.
Dans le cas d'un système temps réel, la définition de l'architecture de l'application se doit d'être complétée par la définition des interfaces entre les différentes fonctions du système. Une interface se définit, d'un point de vue statique, par un format d'échanges basé sur des données structurées et, d'un point de vue dynamique, par l'écriture d'un scénario qui décrit le séquencement des échanges. Pour l'écriture de ces interfaces, le SDL est relativement bien adapté. En effet, les signaux SDL accompagnés de paramètres typés basés sur les données du langage (appelées Types de données abstraits ou ADT, Abstract data types) permettent une description complète d'une interface statique. Le MSC (Message sequence chart, défini dans la même série de standards par l'ITU-T) fournit de son côté une représentation graphique des échanges de données dynamiques entre les entités au sein du modèle SDL, ou avec l'environnement extérieur tel que d'autres modules logiciels ou des drivers (photo A).
Autre point important : les systèmes temps réel sont basés sur l'exécution en parallèle de tâches indépendantes. Il est donc important dans ce contexte de ne pas gaspiller du temps CPU lorsqu'une tâche n'a rien à faire. Ceci a conduit la plupart des applications temps réel à se baser sur des machines à états finis, dans lesquelles le principe de base est de se mettre en attente sur un objet du RTOS, comme une file d'attente de messages, dès que la tâche a terminé son action. Ici, les machines à états finis du langage SDL sont parfaites pour représenter graphiquement ce type de comportement.
Enfin, depuis sa version 92, le SDL est orienté objet à tous les niveaux de la représentation graphique, ce qui permet de construire des librairies de composants logiciels spécialisés, adaptés au marché du temps réel.



A.- L'introduction du mécanisme de représentation MSC, Message sequence chart, dans le « sequence diagram » de l'UML est une des avancées majeures de la version 2.0 de ce langage.

Le SDL a dû s'adapter aux contraintes du temps réel

Sur le papier, le SDL apparaît donc comme un langage idéal pour la spécification et la conception des systèmes temps réel. Mais la réalité technique est tout autre. Premier problème, les ADT ne sont pas adaptés aux impératifs de la conception d'un système temps réel. En effet, leur syntaxe de manipulation a été définie pour spécifier des protocoles, non pour les concevoir. Les développeurs sont alors frustrés de ne pas avoir la précision qu'ils avaient avec des langages de programmation classique comme le C. De plus, l'intégration de code existant est difficile car il faut réaliser une passerelle entre les types de données SDL et les types de données C ou C++.
Autre difficulté, il n'existe pas de compilateur SDL natif ou croisé sur le marché. En conséquence, une phase intermédiaire de génération de code en C est nécessaire pour implémenter le système SDL sur cible.
Enfin, les ADT s'appuient sur des concepts qui ne peuvent pas être traduits directement en C ou en C++, ce qui oblige à écrire des opérateurs spécifiques ren-



B.- Le SDL-RT permet de représenter graphiquement les envois et les réceptions de messages ainsi que les prises et les libérations de sémaphores, caractéristiques essentielles des systèmes temps réel non prises en compte par le seul langage SDL.

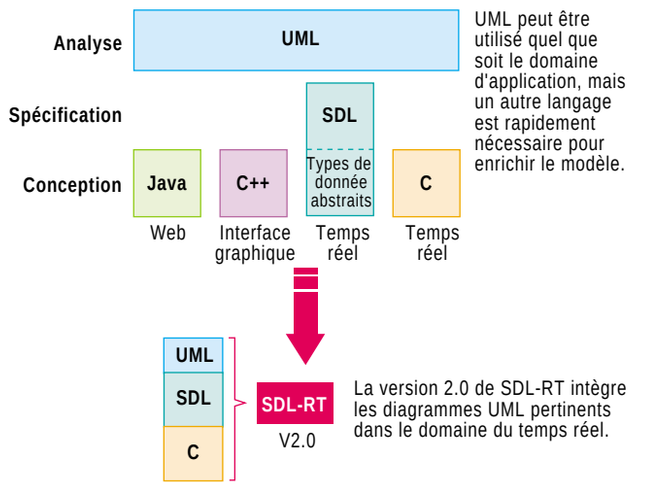
dant le code généré quasi illisible.
Le second problème est lié à l'intégration dans un code SDL des mécanismes propres aux systèmes temps réel. En effet, lorsqu'il s'agit d'intégrer du code généré à partir d'un modèle SDL sur un RTOS, certains principes SDL ne sont plus supportés. Prenons deux exemples simples : le mécanisme de priorité SDL s'applique sur les messages alors que les RTOS offrent classiquement le concept de priorité sur les tâches ; le standard SDL dit qu'une transition d'un automate ne peut être interrompue, alors qu'avec un RTOS elle peut l'être à tout moment, en particulier lors d'appels systèmes si les tâches ont des niveaux de priorité différents. Pour garantir que le code généré se comporte comme spécifié, une machine virtuelle SDL doit alors être générée, rendant l'intégration sur cible difficile et l'accès à certains services du RTOS impossible.
Enfin, dernier problème, les sémaphores n'existent pas dans le langage SDL, alors que c'est un des mécanismes de synchronisation les plus classiques des RTOS. Ceci est dû au fait que le SDL a été conçu pour spécifier des protocoles de télécommunications, qui font communiquer des entités distantes où les sémaphores n'ont évidemment pas lieu d'être.
Pour toutes ces raisons, les outils SDL, lorsqu'ils respectent toutes les composantes du standard, sont devenus complexes à manipuler, chers à l'achat et nécessitent de longues formations avec l'intervention de consultants pointus pour arriver à réaliser l'intégration sur cible. Au bout du compte, les bénéfices de l'emploi d'un modèle graphique sont perdus au moment de cette intégration de l'application sur la cible. Pour contourner ces difficultés, sur le terrain, les utilisateurs du langage SDL se sont mis à écrire du code C à l'intérieur de leurs diagrammes SDL. Mais, comme ce mélange de C et de SDL n'était pas supporté par les outils du marché, car non conforme au standard, ces mêmes utilisateurs ont développé leur propre chaîne d'outils. Rapidement, chaque grand équipementier, comme Alcatel, Nokia, Nortel, EADS ou Sagem, a disposé en interne de son propre générateur de code C à partir de ce mélange de SDL et de C.
La première version de l'environnement SDL-RT, développé par PragmaDev, avait donc pour objectif de formaliser ces pratiques industrielles qui consistent à mélanger les représentations graphiques SDL avec du code C (figure 1). Afin d'étendre l'utilisation de ce formalisme à toutes les applications temps réel, le SDL-RT a aussi introduit la gestion des sémaphores (photo B).

SDL face à la vague UML

Alors que le langage SDL, issu du Pascal, est utilisé dans l'industrie depuis la fin des années 80, l'UML (pour Unified modeling language) est apparu plus tardivement. Fruit de l'unification de plusieurs représentations, il a connu une diffusion assez rapide dans de nombreux domaines depuis sa première version en 1997.
Ce langage se caractérise par son approche entièrement objet, où tout élément d'un système est décrit sous la forme d'une classe. Cette approche générique offre un très haut niveau d'abstraction permettant d'appliquer ses principes de modélisation à n'importe quel concept. En contrepartie l'UML n'est pas assez précis pour décrire de manière détaillée les applications. De fait, on l'utilise souvent dans les phases en amont d'un développement pour les opérations d'analyse et de spécifications système, mais rarement durant les opérations de spécifications détaillées et de conception. Une partie de son succès peut d'ailleurs être attribuée au fait que les équipes peuvent travailler comme elles le faisaient auparavant (avec du C, du SDL...), et documenter leur travail de manière indépendante avec le formalisme d'UML. Cependant, ce lien faible entre les diagrammes UML et la conception pose deux problèmes récurrents : la synchronisation entre la documentation et le code final est mal assurée, et il n'existe pas au sein de l'UML de cadre structurant le processus de développement d'une application.
Bien qu'au sein de l'UML, dans sa version 1.5, il existe neuf types de diagrammes différents pour décrire une application, dans la pratique, pour des développements temps réel, seuls le « class diagram », le « sequence diagram » et le « state chart » sont en général utilisés. Et comme les outils UML sont avant tout des éditeurs graphiques dotés de mécanismes de génération de code très limités, leurs coûts sont beaucoup plus raisonnables que ceux des outils basés sur la technologie SDL, ce qui a facilité une diffusion rapide de cette technologie. Ainsi, plusieurs expériences de grands développements avec des équipes de près de 50 personnes ont poussé l'utilisation du formalisme jusqu'au bout. Mais, souvent, ces programmes ont connu des échecs cuisants. Pourquoi ? D'abord, lors de ces expériences les outils généraient un code deux fois trop volumineux, peu performant, et le débogage sur cible était un véritable chemin de croix. Ensuite, les responsables outils et méthodes ont soulevé une question fondamentale : est-ce qu'une démarche entièrement objet est applicable dans le domaine du temps réel et de l'embarqué, où l'approche traditionnelle est plutôt fonctionnelle ? Difficile de répondre de manière

Domaine de couverture des langages de programmation — FIGURE 1

L'environnement SDL-RT intègre les diagrammes UML adaptés aux contraintes du temps réel, tout en garantissant la cohérence entre les diverses représentations.



tranchée à une telle question, mais les expériences de développement temps réel orienté objet réussies ont souvent eu une approche initiale fonctionnelle, pour ensuite factoriser sous forme de librairie des composants objets pouvant être réutilisés au sein de la société.

UML 2.0 : la tour de Babel !

Une chose est sûre, la pression marketing envahissante d'UML a forcé le SDL à se repositionner en tant que langage de modélisation graphique. La dernière version du SDL, SDL 2000, a intégré le « diagramme de classe » UML et préparé le terrain en vue d'une fusion dans l'UML version 2.0. Celui-ci a pour objectif de supporter le MDA (Model driven architecture) qui permet de définir des profils par domaines d'application. On peut, d'ores et déjà, faire un point sur les évolutions de ces deux langages.

Du côté SDL, malgré les engagements initiaux des différents éditeurs, la dernière version du standard n'a été implémentée dans aucun outil. Plus généralement, le positionnement du langage vers UML, donc vers moins de précision, n'a pas du tout séduit les utilisateurs du SDL. Le SDL-Forum, en juillet dernier, a même pour la première fois initié une « task force » pour définir un sous-ensemble du SDL avec pour objectif la simplification du langage.

La version 2.0 de SDL-RT a néanmoins intégré le « diagramme de classes » et le « diagramme de déploiement » d'UML, très intéressants dans les développements temps réel, car ils complètent parfaitement les représentations SDL tout en garantissant la cohérence entre les différentes vues.

De son côté, les grandes avancées qu'apporte la version 2.0 d'UML, en cours de validation, reprennent partiellement ce qu'on pouvait trouver en standard depuis de nombreuses années dans le SDL, tel que le diagramme d'architecture (structural diagram) et l'intégration du MSC dans le diagramme de séquence (sequence diagram). Mais l'apport majeur de l'UML 2.0 est de pouvoir définir des profils spécifiques par domaine d'application. Le problème est que lorsque la version sera finalisée aucun profil temps réel ne sera défini. Certains grands acteurs industriels doutent même de la standardisation de tels profils pour des raisons d'intérêts stratégiques des éditeurs.

Les premiers outils UML 2.0 apparaissent déjà sur le marché, mais alors que l'objectif initial du nouveau standard était de faciliter l'interopérabilité, les réalités apparaissent bien différentes. Plus concrètement, on retrouve un métamodèle UML commun à tous les éditeurs et des profils propriétaires par domaine d'application. Ces profils ne sont pas publics et les outils proposent rarement un éditeur de profils. On risque donc fort d'avoir autant de versions d'UML 2.0 que d'éditeurs sur le marché et, au bout du compte, les utilisateurs ne gagneront pas beaucoup du fait que ce soit un standard puisque les profils changeront suivant l'outil, et que le portage d'un modèle d'un outil vers un autre risque de se transformer en réécriture des modèles.

Que les outils soient basés sur le SDL ou l'UML, les éditeurs ont eu une démarche prophétique sur leur langage et se faisaient fort de démontrer, à grands renforts de séminaires et d'actions de communication, qu'ils détenaient la vérité. Cela a tellement bien fonctionné qu'il n'est pas rare de rencontrer chez les utilisateurs des fanatiques, pourtant compétents, aveuglés par de tels discours (figure 2). La réalité technique ramène finalement ces brebis égarées à la raison, mais à quel coût !

Des démarches mieux pensées, telles que le SDL-RT, proposent des solutions pragmatiques inspirées des souhaits des utilisateurs finaux et, bien sûr, des tendances présentées dans cet article. On retrouve donc, dans la version 2.0 de cet environnement, certains diagrammes UML, une grande partie des diagrammes SDL, et les langages de programmation C et C++ au sein d'une même représentation homogène (figure 1). L'objectif est d'offrir tous les avantages de ces langages sans leurs inconvénients, le tout dans un format de stockage XML, permettant éventuellement un traitement des modèles sans outils.

Mais, hormis le langage de modélisation, c'est bien sûr les qualités de l'outil qui le supporte qui déterminera le choix des utilisateurs, puisque l'objectif d'une équipe de développement temps réel n'est pas d'être conforme à un langage, ce que semblent parfois oublier les éditeurs, mais de produire rapidement du logiciel de qualité.

Emmanuel Gaudin
(Pragmadev)

Cycle de Hype des langages de développement — FIGURE 2

Dans les langages de développement utilisés actuellement, c'est le C qui bénéficie de la plus grande maturité alors que l'UML est en phase de visibilité maximale, en raison de la très forte activité marketing actuelle des différents éditeurs.