OpenEmbeDD : La plate-forme et ses outils d'ingénierie de modèles

Didier Vojtisek

OpenEmbeDD : La plate-forme et ses outils d'ingénierie de modèles

Didier Vojtisek
didier.vojtisek@irisa.fr
INRIA, Centre Rennes - Bretagne Atlantique
http://www.inria.fr

Résumé

La plate-forme du projet OpenEmbeDD a été conçue en couches afin d'offrir une méthodologie générique pour les besoins du domaine particulier du temps réel. L'approche suivie repose sur l'utilisation de l'Ingénierie Dirigée par les Modèles (IDM) dans l'environnement Eclipse. Les outils dédiés au temps réel étant alors soit conçus grâce à cette méthodologie, soit rendus compatibles avec celle-ci.

En particulier, les efforts conjugués des partenaires ont permis d'intégrer un ensemble cohérent d'outils de modélisation génériques open source qui permettent de traiter la plupart des tâches d'une conception dirigée par les modèles.

Elle s'articule autour de composants tels que le framework EMF de manipulation des modèles, le générateur de modeleurs graphiques Topcased, le langage de transformation de modèles ATL, l'environnement de méta-modélisation Kermeta, l'éditeur de profils UML Papyrus…

D'une part, la plate-forme offre des outils conformes aux standards industriels les plus répandus. Elle incorpore un modeleur UML complet, mais aussi les méta-modèles et profils AADL, SysML, MARTE, SDL.

D'autre part, la plate-forme supporte aussi les approches basées sur l'utilisation de Langages Spécifiques au Domaine (ou DSL) et propose des composants vous permettant de construire et d'outiller votre propre langage. Toutes les activités de la création de DSL sont couvertes : de la conception de sa structure avec le modeleur Ecore, la spécification de sa sémantique opérationnelle avec Kermeta, à la génération automatique d'éditeur et de modeleur avec EMF et le générateur Topcased, en passant par les transformations de modèles avec ATL ou bien la vérification, …

Offrant déjà une bonne couverture fonctionnelle pour faire de l'IDM, la plate-forme est extensible et ouverte, et d'autres composants open source peuvent lui être ajoutés pour la compléter et l'adapter à vos besoins spécifiques.

Mots clés : Modeleur, générateur d'outils, UML, DSL, IDM, transformation de modèles.

1. Introduction

La modélisation apporte une rigueur dans la conception qui apporte de nombreux avantages. Non seulement cela permet d'améliorer la compréhension d'un problème ou d'une solution, mais surtout, dans le cadre de l'ingénierie logicielle, l'un des avantages majeurs est la possibilité d'automatiser les tâches qui s'y rapportent. S'il faut développer aujourd'hui peu d'effort pour convaincre de l'utilité des modèles, il n'en est pas de même quant au prix à payer pour leur définition et leur exploitation. De manière pragmatique, pour permettre cela de façon efficace et satisfaisante, il faut bien évidemment des modèles, mais aussi une infrastructure technique sur laquelle bâtir les outils qui seront donnés à l'utilisateur final.

L'un des résultats les plus visibles du projet OpenEmbeDD [1] est la constitution d'une plate-forme d'Ingénierie Dirigée par les Modèles (IDM) fonctionnelle et réutilisable. Bien que visant le domaine du temps réel et de l'embarqué, elle fournit aussi les briques de base de l'IDM qui permettent d'atteindre cet objectif. En effet, l'approche est utilisée non seulement pour partager des informations entre les outils sous forme de modèles, mais aussi pour tirer parti des techniques de génération offertes. La plate-forme OpenEmbeDD offre donc, non seulement une structure technique sur laquelle assembler des solutions, mais aussi un atelier permettant de les mettre au point.

2. Une plate-forme d'intégration

OpenEmbeDD est avant tout une plate-forme d'intégration et offre plusieurs facilités en ce sens.

2.1. Pragmatique et ouverte

La plate-forme se veut aussi pragmatique et ouverte que possible. En effet, plutôt que de contraindre à l'utilisation de tel ou tel outils, elle va plutôt proposer un choix d'outils ou au minimum proposer une ouverture pour une installer un outil alternatif. Bien entendu certaines solutions seront plus matures que d'autres, mais c'est à chacun de décider de la méthodologie qu'il souhaite suivre et des outils qu'il souhaite utiliser.

Ainsi, la plate-forme propose des langages de modélisation à visée générale, tels qu'UML, qui ne prennent pas en compte les spécificités d'un domaine. Mais elle propose aussi des solutions pour travailler avec des langages métier (ou DSL : Domain Specific Language) qui sont spécifiques pour une communauté donnée.

Il suffit à l'utilisateur de sélectionner les outils qu'il installera et utilisera suivant ses besoins. Son choix pourra être guidé par les expérimentations du projet OpenEmbeDD lui-même ou par celles d'autres projets qui se sont basés dessus. Par exemple, le projet Mopcom-SOC [2] met au point une méthodologie IDM, pour partie basée sur les outils d'OpenEmbeDD, pour développer des composants électroniques.

2.2. Multi-niveaux

Dès la conception de la plate-forme, plusieurs niveaux ont été mis en place. Tout d'abord, Eclipse [3] constitue l'infrastructure technique de base. Il accueille les différents services sous forme de composants (plugins) et permet de construire une interface graphique pour les différentes activités. Il constitue la base l'atelier. Ensuite, EMF [4] introduit un socle de programmation pour traiter de modèles. Ainsi, tous les modèles manipulés dans le cadre de cette plate-forme ont au minimum un méta-modèle décrit en Ecore. Ceci assure une compatibilité technique. Pour compléter cette base technique, certains outils ont été ajoutés, soit en les prenant dans la communauté Eclipse, soit en les étendant ou en les développant dans le cadre du projet OpenEmbeDD. En particulier, le projet s'est attaché à proposer et améliorer des outils qui permettent de créer d'autres outils. Ce nouvel ensemble d'outils constitue un cœur de modélisation générique qui permet par exemple de créer et d'outiller des langages métier.

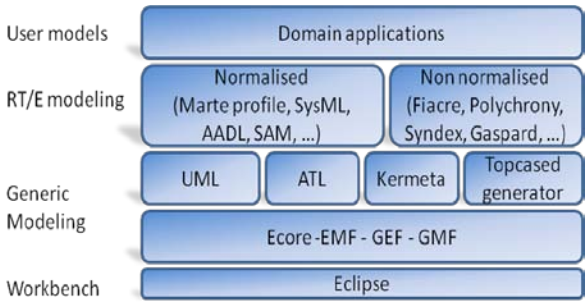

Figure 1. Organisation de la modélisation dans la plate-forme

Les autres outils de la plate-forme viennent alors s'appuyer sur ce cœur générique pour intégrer des solutions propres au domaine. Dans le cadre du projet, ces solutions portaient sur des problématiques de temps réel et d'embarqué. On distingue deux grands groupes d'outils qui interagissent : tout d'abord, ceux s'appuyant sur des méta-modèles normalisés, ensuite, ceux visant des DSL plus spécifiques. Parmi ces derniers, on retrouvera les outils permettant de réutiliser des technologies plus anciennes issues d'un

domaine particulier ou au contraire, des technologies émergentes, telles que celles développées par les partenaires académiques. Ainsi, l'IDM joue un rôle de médiation entre différentes disciplines et offre un cadre méthodologique qui permet d'unifier différentes technologies dans un processus homogène. Ceci permet d'utiliser la technologie la mieux adaptée à chacune des étapes du développement, tout en ayant un processus global de développement qui soit unifié dans un paradigme unique.

Les bonnes pratiques de l'IDM sont donc appliquées à plusieurs niveaux : on automatise non seulement les activités de l'utilisateur du domaine final (par exemple, un concepteur de système embarqué), mais aussi les activités de développement de solutions orientées modèle (par exemple, le concepteur d'un DSL métier).

2.3. Recherche-industrie

La structure technique ouverte évoquée ci-dessus offre des possibilités d'interaction importantes entre les différents intervenants, des fournisseurs de solutions aux utilisateurs. Le projet OpenEmbeDD a catalysé ces interactions grâce à l'implication de nombreux partenaires académiques et industriels tels qu'Airbus, Anyware, CEA/List, CS SI, LAAS/CNRS, France Télécom, INRIA, Thales et Verimag. De plus, l'ouverture de la plate-forme open source rend possible une intégration au plus tôt des avancées en provenance du monde académique et de la recherche en connexion avec des applications industrielles.

3. Les outils orientés DSL

Pour répondre aux problématiques de complexité, d'évolutivité et d'hétérogénéité des systèmes, il semble nécessaire aujourd'hui de recourir à des méthodes relativement nouvelles, fondées sur des techniques de nature générative organisées autour de nombreux DSL s'appuyant sur des méta-modèles précis.

La plate-forme OpenEmbeDD a contribué à développer et améliorer plusieurs DSL génériques visant à promouvoir cette approche et apporte ainsi des solutions pragmatiques aux problématiques évoquées plus haut.

3.1. Le méta-méta-modèle Ecore

Dans la base technique d'Eclipse commune à tous les outils de la plate-forme, les modèles sont décrits conformément à leur méta-modèle qui lui-même est décrit en Ecore. Ce méta-méta-modèle fourni par EMF permet de définir la structure des modèles. Cette structure constitue le cœur des DSL construits, même si cela ne suffit pas pour les préciser complètement. En effet, la structure ne permet pas d'exprimer toutes les contraintes, les syntaxes concrètes ou la sémantique comportementale qui sont associées au DSL, mais ces points sont traités par d'autres outils qui interagissent avec le méta-modèle.

Dans la plate-forme OpenEmbeDD, de nombreuses possibilités sont offertes pour obtenir le fichier Ecore et s'adaptent à tous les besoins. Cela va du simple éditeur arborescent fourni avec EMF, au modeleur graphique avec sa vue diagramme de classe (Figure 2 et Figure 3). Certains pourront préférer utiliser une syntaxe textuelle telle que celle de KM3 [5] ou de Kermeta [6] (pour cette dernière, en utilisant seulement les définitions structurelles). Il est aussi possible d'importer ou d'exporter vers d'autres formalismes si nécessaire, comme par exemple UML, MOF, etc.

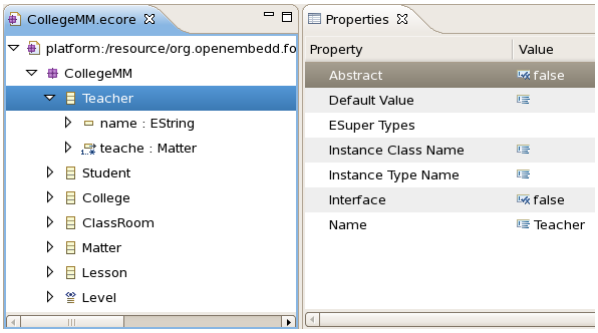

Figure 2. Exemple Ecore avec l'éditeur arborescent

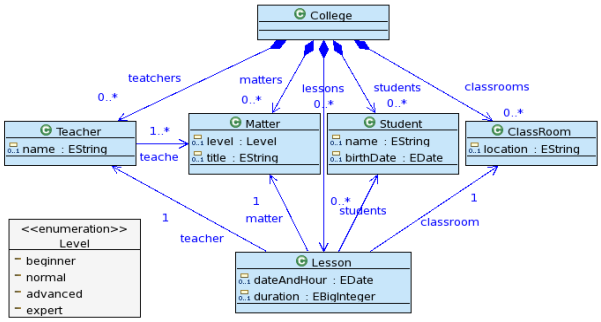

Figure 3. Exemple de diagramme de classe Ecore

3.2. Transformation de modèle avec ATL

L'IDM permet aussi de capitaliser le savoir-faire de modélisation grâce aux transformations de modèles. En effet, celles–ci vont automatiser partiellement ou complètement les actions qui sont habituellement faites

par les concepteurs et les aider dans leurs tâches. De plus, de par leur reproductibilité, cela apporte des éléments de confiance dans le processus et les résultats produits. Il est donc important de disposer d'outils dédiés à la manipulation et la transformation de modèles à modèles. Du fait de la diversité des transformations envisageables, il existe un nombre conséquent d'approches qui visent à traiter les différentes situations [7]. Le choix d'une approche et d'un outil dépend de nombreux facteurs comme par exemple le nombre de modèles et méta-modèles en entrée et en sortie, le domaine d'application, de la réutilisation souhaitée ou tout simplement de l'expérience et des compétences du programmeur.

Néanmoins, issu des travaux de recherche et en partie grâce aux efforts fournis dans le cadre du projet OpenEmbeDD, le langage ATL [8] a atteint un niveau de maturité suffisant pour susciter un réel attrait de l'industrie. Il est maintenant diffusé directement dans l'un des projets d'Eclipse et dispose d'une version Pro supportée par la société Obeo.

Initialement inspiré du standard OMG QVT [9] et basé sur OCL [10], ATL est un langage dédié pour faire de la transformation de modèles proposant une approche hybride. En effet, même si le style de programmation recommandé suit une approche déclarative, il permet néanmoins d'utiliser une approche impérative lorsque la première n'est pas adaptée au problème traité.

Le cœur déclaratif de l'outil est basé sur un système de règles qui seront déclenchées lorsque le moteur d'exécutions trouve des éléments de modèle pour lesquels les conditions sont remplies. Ce mécanisme de sélection permet de rapidement et intuitivement poser les bases d'une transformation. Les transformations décrites en ATL sont unidirectionnelles et si la transformation inverse est requise, il faudra écrire un second jeu de règles ATL.

En complément du moteur de règles, ATL déploie toute une panoplie de constructions dédiées qui permettent de simplifier nombre de problèmes courant lorsque l'on spécifie une transformation de modèles. Par exemple les requêtes (Query) permettent de faire de la génération vers des types primitifs et donc de générer du texte. Afin d'affiner l'ordre d'appel, il existe aussi plusieurs sortes de règles ayant des comportements différents, comme par exemple la possibilité d'être appeler explicitement plutôt qu'au travers du moteur de règle.

Si les premières versions d'ATL ne permettaient pas de modifier le modèle d'entrée (in-place transformations), cette limitation a été retirée des versions récentes. Les règles peuvent être factorisées dans des modules pour pouvoir être réutilisées. Il donc maintenant facile de transformer un modèle par raffinements successifs.

Enfin, en tant que langage pour l'IDM, ATL utilise une syntaxe textuelle mais il donne bien évidement accès au modèle de la transformation elle-même qui pourra aussi être transformée.

Intégré à Eclipse et à la plate-forme OpenEmbeDD, le langage dispose d'un environnement de développement moderne avec le support du debug, la coloration syntaxique, la complétion, etc.

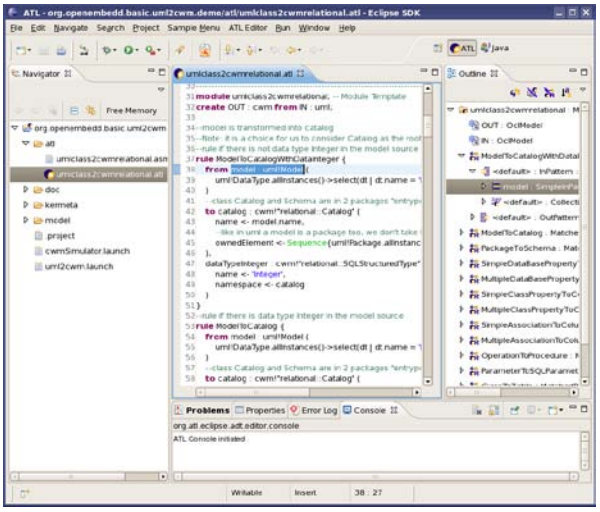

Figure 4. Workbench et exemple de transformation en ATL

3.3. Ajout de sémantique avec Kermeta

Comme décrit plus haut, le méta-méta-modèle Ecore utilisé par les outils de la plate-forme permet de définir la structure des modèles. Même si c'est déjà un premier pas vers la formalisation et permet déjà de générer des outils, plusieurs aspects ne sont pas couverts par celui-ci. Par exemple, certaines contraintes ne peuvent pas être exprimées uniquement avec la structure (comme l'unicité des noms dans un modèle). De même, en ne connaissant pas précisément le comportement attendu d'un modèle, celui-ci peut facilement être détourné de son usage initial et l'on peut obtenir des incohérences entre les différentes implantations et leur exploitation par les différents outils. En effet, même si le méta-modèle (qu'il soit écrit en Ecore ou en MOF) est une sorte de diagramme de classe, il peut définir des opérations mais celles-ci n'ont pas de corps qui en décrivent le comportement.

Pour traiter ces problèmes, nous pouvons utiliser le langage Kermeta [5]. Ce langage met en avant un mécanisme astucieux inspiré du tissage d'aspects qui permet d'étendre un méta-modèle écrit en Ecore. Il est ainsi possible de ré-ouvrir les définitions d'un méta-

modèle pour intégrer des contraintes (invariants, pré et post conditions) et un corps pour les opérations. Ce même mécanisme peut s'appliquer à l'ajout de nouvelles propriétés et d'opérations pour créer des outils qui dériveront du méta-modèle d'origine.

Typiquement, à partir d'un jeu d'invariants écrits soit en OCL, soit directement en Kermeta, on pourra obtenir un validateur de modèle. Le système d'aspects permet alors d'envisager plusieurs jeux de contraintes qui s'adaptent à l'utilisation que l'on fait d'un méta-modèle donné. Par exemple, on n'utilisera pas les mêmes concepts d'UML de la même manière si le modèle représente la conception détaillée d'un composant matériel ou l'expression des besoins d'une application web.

En ajoutant un corps aux opérations des méta-modèles, le langage Kermeta n'est plus uniquement un langage de métadonnées comme Ecore ou MOF, mais un langage de métaprogrammation. En effet, de même que Niklaus Wirth définissait les programmes comme des structures de données et des algorithmes, Kermeta présente les méta-modèles comme des métadonnées et des actions. Cette approche permet d'aller au-delà de la seule syntaxe et de pouvoir spécifier également la sémantique opérationnelle des méta-modèles. Kermeta propose ainsi à la fois un langage de méta-modélisation et un langage d'action. Il a une syntaxe proche de Java mais permet de travailler directement des instances de modèles, et de manipuler les concepts définis au niveau du méta-modèle comme des types de base.

En combinant les avantages de la programmation orientée aspects avec les patrons de conception de la programmation objet sur des modèles, ce langage d'action peut être utilisé pour construire de nombreux outils qui pourront alors compléter la plate-forme OpenEmbeDD. Une application typique sera par exemple la construction d'un simulateur (basé sur le patron interpréteur) pour avoir rapidement une implémentation de référence, le mécanisme de tissage permettant alors de traiter élégamment les problèmes de variation sémantique.

Enfin, puisque ce langage d'action manipule des éléments de modèle, il peut être utilisé aussi comme langage de transformation. Dans cet usage, il peut combiner un nombre quelconque de méta-modèles en entrée et en sortie pour créer des transformations arbitrairement complexes dans un style de programmation orienté objet simplifié grâce aux aspects.

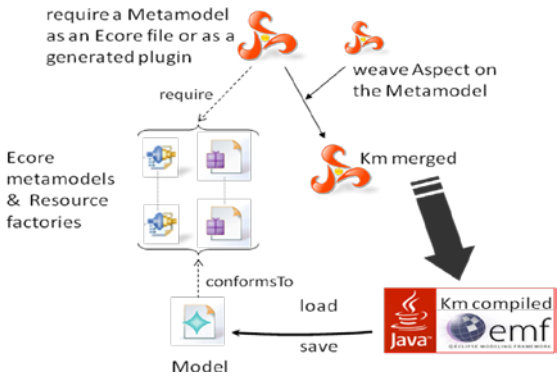

Figure 5. Tissage et compilation avec Kermeta

Au niveau environnement, le langage est fourni non seulement avec des éditeurs, mais aussi avec un interpréteur, un débuggeur et, grâce aux travaux dans le cadre d'OpenEmbeDD, un compilateur. Ce compilateur permet de finaliser les outils construit avec Kermeta pour fournir aux utilisateurs finaux une version java indépendante de Kermeta qui fonctionne avec tous les outils d'EMF. On peut par exemple, s'en servir pour compiler en java un ensemble de contraintes OCL pour obtenir un vérificateur de contrainte plus efficace.

3.4. Générateur de modeleur Topcased

Pour un même point de vue ou un même objectif, les modèles peuvent prendre plusieurs formes : graphiques, descriptions formelles, descriptions textuelles, etc. Manipuler les modèles requiert alors non seulement de définir les concepts principaux associés à la problématique sous-jacente mais aussi de les représenter de manière efficace qui permette de raisonner sur les modèles. Il convient donc de construire un certain nombre d'outils adéquats. Historiquement, la représentation privilégiée dans le monde des modèles est une représentation graphique sous forme de diagrammes. Malheureusement, malgré l'existence de différents frameworks graphiques dans les environnements de programmation, l'écriture manuelle d'un modeleur pour un modèle donné reste très coûteuse à mettre au point et à maintenir en cas d'évolutions.

Le générateur Topcased [11] vient réduire l'impact de ce problème en lui appliquant les techniques de l'IDM, à savoir modéliser les éléments graphiques pour pouvoir générer la quasi-totalité du modeleur que l'on souhaite fournir aux utilisateurs finaux. Ainsi, non seulement il n'est plus nécessaire d'être un spécialiste de la programmation graphique, mais en plus, une

évolution du méta-modèle aura un impact moins important.

Pour cela, le générateur utilise un DSL de configuration. Il vise à définir les informations permettant de construire les différents éléments d'un modeleur. On retrouvera des informations pour associer un ou plusieurs diagrammes (par exemple un pour chacun des différents types de diagrammes d'UML). On pourra configurer d'autres éléments de l'éditeur comme le contenu de la palette, des menus contextuels ou comment les éléments sont représentés dans la vue propriété. Tous ces éléments seront mis en relation avec les concepts du méta-modèle central que l'on outille, par exemple pour indiquer que tel concept doit être représenté sous forme de boite ou sous forme de lien. Grace à ces informations, le générateur créera automatiquement le code de l'éditeur qui fonctionnera avec Eclipse. Il s'appuiera sur le framework GEF pour la partie graphique et sur EMF pour la partie modèle ce qui lui assure une intégration avec les autres outils.

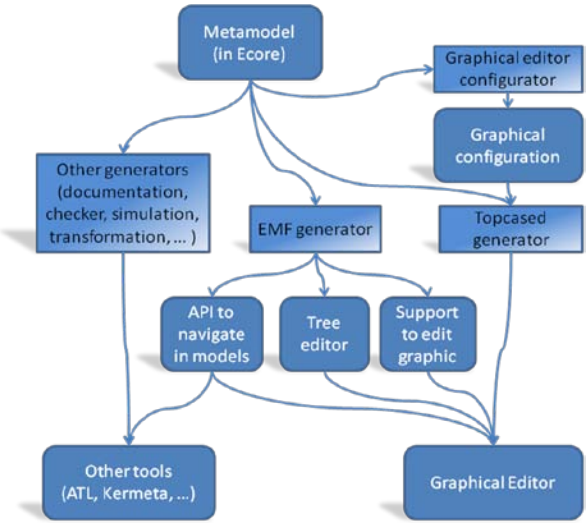

Figure 6. Principe de génération d'un éditeur graphique et connexion avec les autres outils

Suivant les mêmes principes que le générateur de code d'EMF, il est ensuite possible d'effectuer quelques dernières retouches dans le code java afin d'affiner les quelques points qui n'auraient pas pu être gérés directement par le générateur ou que vous souhaitez personnaliser (icônes, comportements graphiques, connexion à d'autres outils,…).

L'éditeur généré respecte bien les concepts de l'IDM. En effet, une fois sauvée sur le disque, la présentation (ie. les diagrammes) est bien distincte du modèle. On retrouvera un fichier pour le modèle de l'utilisateur et un pour les diagrammes, les dépendances allant bien du diagramme vers le modèle et non l'inverse. Là encore, cela permet de partager le modèle et d'interagir avec d'autres outils, y compris d'autres éditeurs…

Ce générateur Topcased est utilisé en de multiples endroits de la plate-forme OpenEmbeDD, aussi bien pour des outils dédiés à la modélisation (par exemple l'éditeur UML2, l'éditeur graphique de Kermeta, …) que pour des outils dédiés au temps réel et à l'embarqué (par exemple l'éditeur de l'outil de conception synchrone Polychrony/SME [12], les éditeurs pour les méta-modèles SAM[1]et AADL[2], Syndex …).

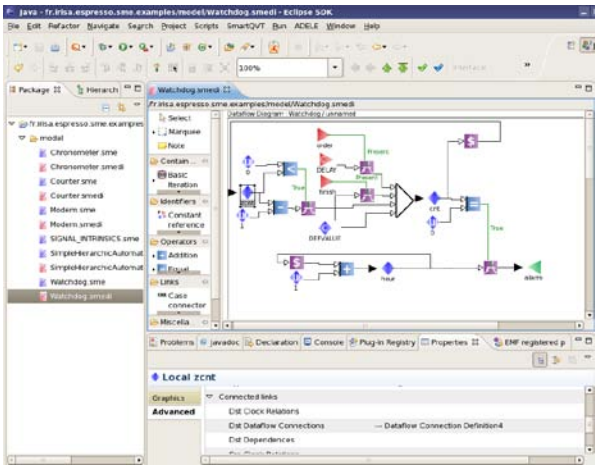

Figure 7. Editeur Polychrony/SME généré avec Topcased

3.5. Exemple de réalisation Topcased-ATL-Kermeta

Afin de faciliter la prise en main des différents outils, chacun d'entre eux propose ses propres tutoriaux qui sont distribués avec la plate-forme OpenEmbeDD. En complément et pour illustrer l'intégration des outils entre eux, la plate-forme propose une « BasicDemo » qui enchaine plusieurs des outils ci-dessus. Dans cette petite expérimentation, le générateur Topcased est utilisé pour générer un modeleur UML. En prenant un modèle UML créé avec cet éditeur, on utilise une transformation écrite en ATL pour générer un modèle de base de données conforme au méta-modèle CWM (Common Warehouse Metamodel) [13]. Ce dernier méta-modèle aura été préalablement étendu grâce à Kermeta pour lui donner

[1] *Structured automata Metamodel*, formalisme à base d'automates hiérarchiques utilisé par Airbus

[2] Architecture Analysis & Design Language, langage de description d'architecture système destiné aux systèmes embarqués cf. http://www.aadl.info/

un comportement. On peut ainsi simuler le comportement de la base de données issue du modèle UML sans avoir à passer par les étapes de déploiement sur une plate-forme réelle.

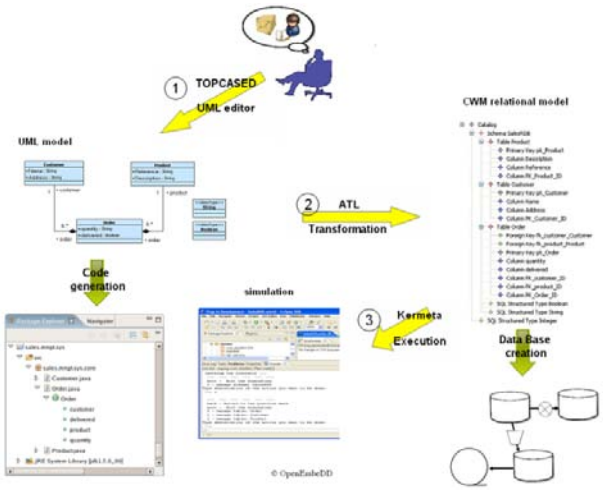

Figure 8. Schéma général de la BasicDemo OpenEmbeDD

4. Les outils orientés normes ou standards

Grâce aux outils de génération comme ceux présentés précédemment, il est maintenant plus facile et moins coûteux de créer et d'outiller des DSL adaptés à ses propres besoins spécifiques à un métier ou une communauté. Il est par exemple possible d'avoir des outils basés sur une abstraction suffisamment précise pour pouvoir la traiter de manière sûre et fiable et donc d'en tirer des résultats automatiques qui auraient été difficiles voire impossibles à obtenir sur un modèle inadapté ou trop générique. Par exemple, il est très difficile de faire de la vérification formelle sur des langages généraux tels qu'UML.

Néanmoins, quand les besoins sont similaires entre plusieurs groupes d'utilisateurs, il reste malgré tout nécessaire de se raccrocher à certains d'entre eux. C'est le rôle des organismes de normalisation que de proposer des standards correspondant à un ensemble de besoins. Le minimum étant de fournir une passerelle entre un DSL propriétaire (ou moins commun) et un modèle plus largement accepté qui jouera le rôle de pivot. Ainsi, il est possible de réutiliser les outils fournis par d'autres.

La plate-forme OpenEmbeDD tient compte de cette possibilité et fourni par défaut un ensemble d'outils basés sur des implémentations de normes.

On retrouvera tout d'abord un modeleur UML2.1 open source qui supporte les profils. Le modeleur gère tous les diagrammes d'UML. Ensuite, sont fournis plusieurs profils pour UML. En premier lieu, une implémentation open source du profil MARTE [14] est disponible grâce au travail du projet. En effet, les partenaires du projet étaient fortement impliqués dans la standardisation à l'OMG de ce profil dédié à la modélisation et l'analyse de systèmes temps réel et embarqués. La plate-forme fourni aussi le profil SysML [15] qui est un standard de l'OMG pour concevoir et vérifier des systèmes et systèmes de systèmes. Enfin, la plate-forme propose une implémentation du langage et d'un modeleur AADL qui est un standard du SAE pour la description de systèmes destiné aux systèmes embarqués notamment dans les contextes automobile, aéronautique et spatial.

En plus de modeleurs et d'outils spécifiques pour ces standards, OpenEmbeDD propose des outils pour faciliter les connexions de ces modèles standards avec d'autres langages, en fournissant des passerelles vers certains outils et langages des mondes synchrones et asynchrone comme Polychrony, Gaspard2 [16], Syndex [17], ou Fiacre [18]…

5. Conclusion

L'ingénierie des modèles offre un cadre méthodologique qui permet d'unifier différentes technologies dans un processus homogène. Ceci permet d'utiliser la technologie la mieux adaptée à chacune des étapes du développement, tout en ayant un processus global de développement qui soit unifié dans un paradigme unique. La plate-forme OpenEmbeDD en est une implantation qui, même si elle est initialement dédiée au temps réel et l'embarqué, propose une base constituée d'outils génériques réutilisables dans de nombreux autres domaines.

De plus, sa structure ouverte et extensible a permis d'intégrer au plus tôt les avancées des évolutions technologiques en provenance du monde académique et de la recherche, tout en les appliquant à des solutions industrielles.

Grâce à l'implication des partenaires d'OpenEmbeDD dans le nouveau projet Européen CESAR [19], les efforts d'intégrations et les outils présentés ici pourront être pérennisés dans une nouvelle plate-forme de plus grande ampleur.

6. Références et liens

[1] OpenEmbeDD : http://www.openembedd.org

[2] Aulagnier D., Koudri A., Lecomte S., Soulard P., Champeau J., Vidal G., Perrouin G., and Leray P. -- Soc/sopc development using mdd and marte profile. -- In *Model Driven Engineering for Distributed Real-time Embedded Systems*. Hermes, 2009.

[3] Eclipse : http://www.eclipse.org

[4] EMF : Eclipse Modeling Framework project http://www.eclipse.org/modeling/emf/

[5] Jouault, F, and Bézivin, J: KM3: a DSL for Metamodel Specification . In: Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, LNCS 4037, Bologna, Italy, pages 171-185. 2006.

[6] Muller P.-A., Fleurey F., and Jézéquel J.-M.. -- Weaving executability into object-oriented meta-languages. -- In S. Kent L. Briand, editor, Proceedings of MODELS/UML'2005, volume 3713 of LNCS, pages 264--278, Montego Bay, Jamaica, October 2005. Springer. http://www.kermeta.org

[7] Czarnecki, K. and Helsen, S. Feature-based survey of model transformation approaches. /IBM Syst. J./ 45, 3 (Jul. 2006), 621-645. 2006.

[8] Allilaire, F., Bézivin, J., Jouault, F., Kurtev, I.: ATL – Eclipse Support for Model Transformation. In: Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France. (2007) http://www.eclipse.org/m2m/atl/

[9] OMG/RFP/QVT MOF 2.0 Query / Views / Transformations RFP, OMG document ad/2002-04-10.

[10] OMG. Object Constraint Language (OCL). (2003) OMG Document ptc/03-10-14.

[11] Farail P., Gaufillet P., Canals A., Le Camus C., Sciamma D., Michel P., Crégut X., and Pantel M.. The TOPCASED project : a Toolkit in OPen source for Critical Aeronautic SystEms Design. In 3rd European Congress EMBEDDED REAL TIME SOFTWARE (ERTS), Toulouse, France, January 2006.

[12] Brunette, C., Talpin, J.-P., Gamatié, A., Gautier, T. : A metamodel for the design of polychronous systems In: Journal of Logic and Algebraic Programming, Special Issue on Applying Concurrency Research to Industry. Elsevier, 2008.

[13] OMG Common Warehouse Metamodel (CWM) Specification, v1.1, OMG document formal/2003-03-02.

[14] The Object Management Group. 2005. The UML Profile for Modeling and Analysis of Real-Time and Embedded systems, beta 1. OMG Adopted Specification, ptc/07-08-04. Also : http://www.omgmarte.org/

[15] OMG/SysML Systems Modeling Language specification (2008) OMG document formal/2008-11-01. also http://www.omgsysml.org/

[16] Gaspard2 : http://www2.lifl.fr/west/gaspard/

[17] Grandpierre T., Lavarenne C., and Sorel Y. : Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In Proceedings of 7th International Workshop on Hardware/Software Co-Design, CODES'99, Rome, Italy, May 1999.

[18] Berthomieu B., Bodeveix J.-P., Farail P., Filali M., Garavel H., Gaufillet P., Lang F., and Vernadat F. FIACRE: an Intermediate Language for Model Verification in the TOPCASED Environment. In Proceedings of the 4th European Congress on Embedded Real-Time Software ERTS'08 (Toulouse, France), 2008.

[19] CESAR : http://cesarproject.eu/

7. Glossaire

AADL : Architecture Analysis and Design Language, un standard SAE.

DSL : Domain Specific Language.

IDM : Ingénièrie Dirigée par les Modèles.

Profil MARTE : Modeling and Analysis of Real Time and Embedded systems.

MDA : Model Driven Architecture.

OMG : Object Management Group.

UML : Unified Modeling Language.

SAM : Structured automata Metamodel, formalisme à base d'automates hiérarchiques utilisé par Airbus.