Chapitre 01. Approches de développement des Systèmes Embarqués

I. Génie logiciel classique

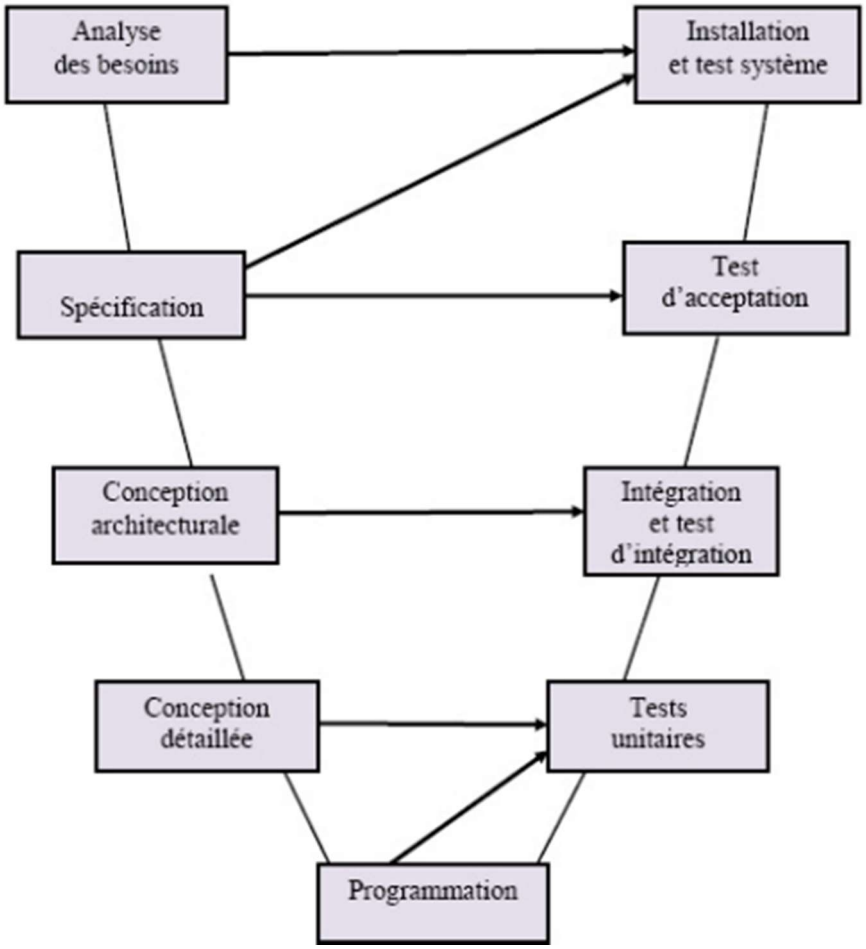
Dans ce cas, le développement d'une application suit un cycle comprenant essentiellement une étape d'expression des besoins, une étape de spécification, une étape de conception, une étape d'implémentation, une étape d'intégration et de test, une étape d'installation et de vérification et enfin une étape de maintenance.

La spécification est une description abstraite du futur système, elle décrit ce que ce dernier doit faire sans pour autant donner des détails de la manière dont il le fait. La conception aboutit à la création de modèles conceptuels qui seront par la suite implémentés dans la phase d'implémentation.

L'ordre des étapes définit un modèle de développement particulier. Néanmoins, ces étapes font partie de tous les cycles de développement de systèmes indépendamment de la nature, du domaine, de la taille et de la complexité du système à développer, aussi, les systèmes distribués embarqués temps réel en sont-ils concernés. Il existe divers modèles de développement d'une application, nous en citerons le modèle en cascade, le développement incrémental et le modèle en V, ce dernier étant très communément utilisé dans le développement des systèmes temps réel en général.

Le modèle en V est constitué de deux branches : La première branche est un modèle en cascade dont le rôle est d'analyser les besoins, de spécifier le logiciel, de le concevoir et enfin de l'implémenter. La deuxième branche quant à elle, consiste à assembler les constituants et à effectuer un ensemble de tests afin de valider le système.

Notons que les étapes de la deuxième branche utilisent celles de la première branche pour établir les validations ainsi que les vérifications nécessaires. La validation d'une étape de conception entraine la remontée vers une étape de validation de niveau hiérarchique supérieur tandis que la non validation d'une étape de conception entraine la réexécution de cette étape et de certaines ou éventuellement de toutes les étapes inférieures ainsi que leur revalidation.



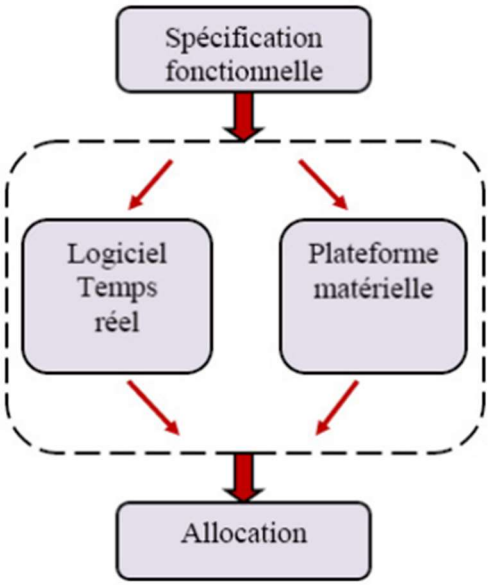
Trois étapes de validation complètent le modèle à savoir les tests unitaires, l'intégration et les tests d'intégration et enfin la validation du système. Une des raisons de l'utilisation de ce modèle de développement pour les systèmes embarqués temps réel est le fait que beaucoup d'importance est accordée à la partie 'tests', étape très pertinente pour ce type de systèmes qualifiés de 'critiques'. Par ailleurs, dans cet axe de recherche, on distingue deux sous-axes : le premier a pour but l'enrichissement d'étapes du cycle de développement tandis que le second concerne l'amélioration des transitions entre les différentes étapes. — Enrichissement intra-étape Dans ce cas, il s'agit de se concentrer sur une des étapes du développement telles que la spécification, la conception ou l'implémentation avec l'objectif d'y apporter des modifications. Les travaux menés par Carlson [Carlson, 2002] ont pour objectif d'exprimer les contraintes temps réel dans un langage de spécification. L'intégration des techniques d'ordonnancement temps réel dans une conception orientée objet est également une approche d'amélioration de l'étape de conception où le langage utilisé est UML-RT. L'étape d'implémentation est prise en compte à travers la traduction automatique d'un programme ESTEREL vers un code.

— Enrichissement inter-étapes : Dans ce cas, l'intérêt est porté sur le passage d'une étape à une autre plutôt que sur une étape donnée. La génération du code à partir

d’un modèle conceptuel établi en SDL (Specification and Description Language) ainsi que son optimisation . UML et SDL sont combinés où des règles sont définies permettant la transformation automatique d’un sous-ensemble formalisé d’OMT noté OMT* vers SDL. Un profil UML (SDL with UML) pour SDL est défini, il permet d’utiliser un ensemble de concepts d’UML en conjonction avec SDL.

II. Co-design

Les systèmes embarqués temps réel étant avant tout des systèmes hybrides (matériel et logiciel), leur développement doit prendre en compte les deux aspects. Le processus de conception de tels systèmes est caractérisé par une partie conjointe matériel/logiciel :



Spécification fonctionnelle Prenant comme base le cahier des charges, cette étape consiste en une description des fonctionnalités du système. Le standard SysML qui applique l'IDM à l'ingénierie système définit deux diagrammes de blocs : Block Definition Diagram et Internal Block Diagram appropriés à la spécification fonctionnelle. Le concept de block en SysML est similaire à celui de composant dans UML.

Partionnement :

Pour chaque composant matériel identifié lors de l'étape de spécification fonctionnelle, le partitionnement consiste à fixer un choix d'implémentation. On distingue trois flots possibles:

— **La réutilisation d'IPs :** Il s'agit de réutiliser des composants déjà conçus et validés qu'on appelle 'intellectual properties';

— **La conception de circuits matériels :** concerne aussi bien les circuits imprimés que les circuits configurables;

— **La conception de logiciels :** consiste en deux flots fortement couplés qui sont le logiciel embarqué et sa plateforme d'exécution.

L'objectif de l'étape de partitionnement est de trouver le meilleur compromis entre les différents flots tout en minimisant les coûts et en maximisant les performances.

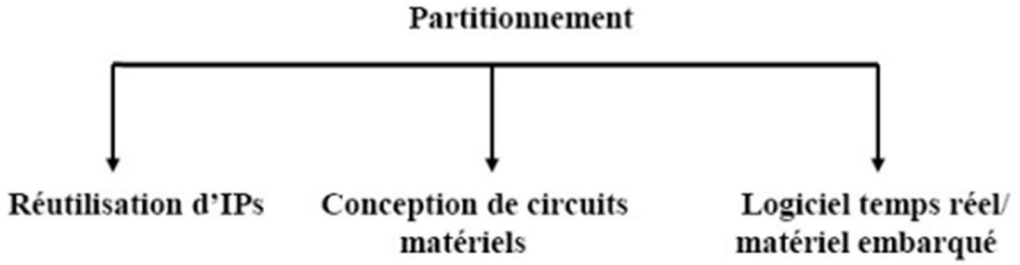


FIGURE 2.3: Flots de conception.

Deux flots parallèles permettent la conception du logiciel et la réalisation de la plateforme d'exécution.

Les deux flots utilisent des environnements de développement différents, ce qui rend la communication entre les deux équipes délicate. Malgré l'existence de points de synchronisation, des erreurs peuvent avoir lieu et tard dans le processus de développement.

Allocation :

L'allocation consiste à affecter à chaque partie du logiciel le matériel supportant son exécution, Ptolemy II et Metro II sont des exemples d'approches de co-design :

— Ptolemy II

Ptolemy II est un environnement de modélisation d'un système en termes de composants hétérogènes indépendants de toute plateforme d'exécution. La communication entre composants est le point central de l'approche, elle est définie par des sémantiques appelées 'Modèles de Calculs' (MoC). La modélisation se base sur la notion d'acteurs qui définissent des composants en Java et de directeurs qui définissent la sémantique de communication entre les acteurs. L'environnement Ptolemy permet la modélisation fonctionnelle de haut niveau.

— Metro II

Metro II est un environnement de conception basé sur la technique Platform-based design. Le langage SystemC est utilisé pour la description du comportement interne des composants. Disposant des spécifications textuelles ou graphiques de l'application, de l'architecture matérielle et des contraintes de conception, Metro II suit un processus itératif afin d'aboutir à un modèle du système respectant le cahier des charges. Cette approche permet également l'intégration des IPs (Intellectual Properties) et donc la possibilité de réutilisation de modèles.

III. Méthodologies orientées UML

UML (Unified Modeling Language), préconisé par l'OMG comme langage de modélisation, s'articule autour d'un ensemble de diagrammes permettant de décrire un système selon plusieurs points de vue. Il existe deux catégories de diagrammes, la première concerne les diagrammes structurels dont le rôle est de décrire l'architecture du système tandis que la seconde concerne les diagrammes comportementaux qui permettent la description de l'aspect dynamique du système. UML est un langage applicable dans plusieurs domaines comme il peut également être utilisé avec différentes plateformes d'implémentation dont CORBA (Common Object Request Broker Architecture), J2EE (Java 2 Enterprise Edition) ou encore Microsoft.NET.

UML est un langage de modélisation adapté à de nombreux domaines d'application. Cependant, il existe des domaines spécifiques auxquels ce langage n'est pas complètement approprié, en l'occurrence, celui du temps réel. La notion de 'profil', apparue dans le standard UML 1.3, a été créée afin de permettre la spécialisation d'UML à différents contextes. L'OMG propose un ensemble de profils standardisés adaptés à différents domaines, nous en citerons :

— MARTE

MARTE (Modeling and Analysis of Real Time and Embedded systems) est un profil UML standard de l'OMG, il a pour but la spécification et la validation des propriétés temps réel d'un système. Pour cela, il offre le langage VSL (Value Specification Language) sous la forme d'une extension du langage déclaratif OCL (Object Constraint Language) afin de supporter les contraintes comportementales. Le profil MARTE est constitué essentiellement de trois paquetages à savoir le paquetage foundation prenant en charge, entre autres, la modélisation des propriétés non fonctionnelles, le paquetage design qui permet la modélisation des applications ainsi que des plateformes d'exécution et enfin le paquetage analysis qui fournit les mécanismes permettant d'annoter les modèles à des fins d'analyse. En plus, il existe un quatrième paquetage annexes qui contient, entre autres, la bibliothèque des modèles prédéfinis MART EM odel Library et le langage de spécification de valeur VSL (Value Specification Language).

— TURTLE

TURTLE (Timed UML and RT-LOTOS Environment) est un profil UML, il spécialise le langage UML en prenant en charge les systèmes temps réel. TURTLE est basé sur l'algèbre des processus temporisés RT-LOTOS (Real Time LOTOS) construite elle-même sur la technique de description formelle LOTOS (Language of Temporal Ordering of Sequences).

— RT-LOTOS

RT-LOTOS (Real Time LOTOS) est une extension de LOTOS, il y intègre trois opérateurs temporels :

— *delay(d) :* est l'opérateur de délai, il permet de retarder un processus d'un temps d;

— *latency :* est l'opérateur de latence, il permet de retarder un processus d'un certain temps appartenant à l'intervalle de latence [0 :1];

— *aT :* est l'opérateur de restriction temporelle, il limite le temps pendant lequel une action observable a peut être offerte à son environnement.

L'approche est supportée par un ensemble d'outils TTOOL (TURTLE Toolkit), celui-ci inclut un éditeur graphique de diagrammes, un analyseur syntaxique, un générateur de code ainsi que des outils de visualisation et d'analyse des résultats de simulation et de validation.

— PEARL

La spécification PEARL permet de définir un système embarqué temps réel par l'allocation d'une configuration logicielle à une configuration matérielle. La spécification des communications ainsi que celle des architectures tolérantes aux pannes est possible.

Cependant, les concepts définis par la spécification PEARL ne permettent pas de spécifier des contraintes temporelles complexes. Par ailleurs, en se basant sur les concepts introduits dans la spécification PEARL, un patron de reconfiguration UML pour les systèmes embarqués est décrit par un profil UML-RT. Il permet la modélisation et la gestion des reconfigurations des architectures logicielles et matérielles.

IV. Approches à base de composants

Le développement à base de composants est une approche qui a considérablement été adoptée depuis plus d'une dizaine d'années. Elle permet essentiellement de :

— Réduire la complexité des systèmes et ce en les concevant par assemblage de composants préexistants;

— Accélérer le développement des systèmes grâce à la réutilisation des composants dans différentes applications;

— Faciliter l'évolution des systèmes vu les structures modulaires engendrées par une telle approche.

L'approche par composants se trouve être une approche très appropriée à la conception des systèmes en général. Néanmoins, les modèles de composants connus tels que CORBA/CCM de l'OMG,

(D)COM/COM+ de Microsoft et Enterprise JavaBeans de SUN ne sont que très rarement utilisés pour la conception de systèmes temps réel. Ceci est dû à leur manque de support pour les propriétés temporelles. Il existe également d'autres modèles de composants dont la technologie AutoComp, le modèle RTCOM du projet ACCORD, KOALA, SaveCCM du projet SAVE, PECOS. . . etc.

— AADL

AADL (Architecture Analysis and Design Language) est un langage de description d'architecture destiné aus systèmes embarqués. Il est un standard international publié par la SAE (Society of Automative Engineers). AADL repose essentiellement sur la notion de composant pour lequel une interface précise est définie. La description de l'interface est séparée de son implémentation. L'architecture d'un système y est décrite comme un ensemble de composants hiérarchisés, composés et inter-connectés.

On distingue trois types de composants : les composants logiciels, les composants matériels et les composants composites. D'autre part, la communication entre composants est décrite à l'aide de ports et de connexions permettant plusieurs types de communication. Il est possible de compléter la sémantique de AADL en ajoutant des éléments composant un modèle.

V. Ingénierie dirigée par les modèles

La complexité croissante des logiciels a créé des exigences dans le domaine du génie logiciel. En effet, la séparation des préoccupations ainsi que la multiplicité des plateformes a renforcé le passage au paradigme des modèles. Force est de constater que dans le domaine de l'ingénierie, la réalisation d'un système passe par une bonne compréhension de celui-ci à toutes les étapes du cycle de développement.

Une telle compréhension est d'autant plus réussie que lorsque la modélisation du système a lieu. En effet, modéliser un système permet de le comprendre, de l'analyser et éventuellement de l'améliorer.

L'IDM ou Model Driven Engineering (MDE) en anglais, est une approche du génie logiciel qui s'articule autour du concept de modèle en se plaçant d'abord à un niveau abstrait de description et en affinant à travers des niveaux jusqu'à atteindre celui de la programmation. C'est ainsi que l'IDM se démarque par l'usage systématique des modèles tout au long du développement logiciel.

L'architecture dirigée par les modèles (MDA) dérivée de l'IDM par l'OMG, est basée sur les modèles, ceux-ci sont décrits dans un langage de modélisation dont le méta-modèle est exprimé en MOF (Meta Object Facility).

L'approche MDA se base sur les notions de modèle, méta-modèle et transformation de modèles. Via les modèles, plusieurs facettes d'une application peuvent être décrites tout en faisant abstraction des détails d'implémentation.

— Modèle

Un modèle est une description abstraite d'un système, il peut être textuel et/ou graphique ou encore formel. Un système peut être modélisé de plusieurs manières correspondant à différents point de vue. Le rôle d'un modèle est celui de comprendre, de concevoir, de simuler et d'améliorer un système.

— Méta-modèle

Les concepts et la sémantique d'un modèle sont définis par une grammaire, ces règles auxquelles le modèle doit être conforme s'appelle 'méta-modèle'. Par conséquent, tout modèle doit être conforme à un méta-modèle.

Quatre couches d'abstraction permettent de délimiter différents concepts :

Le niveau M0 correspond à la réalité. Celle-ci est représentée par un premier niveau d'abstraction qui est le modèle. Ce dernier est conçu en respectant une syntaxe qui est le méta-modèle du niveau M2. Le méta-modèle est à son tour un modèle et doit par conséquent être conforme à une syntaxe. Le méta-méta-modèle est le langage qui permet de décrire le méta-modèle et il constitue le niveau M3. Notons que dans la pratique, on considère qu'un méta-méta-modèle est conforme à lui-même.

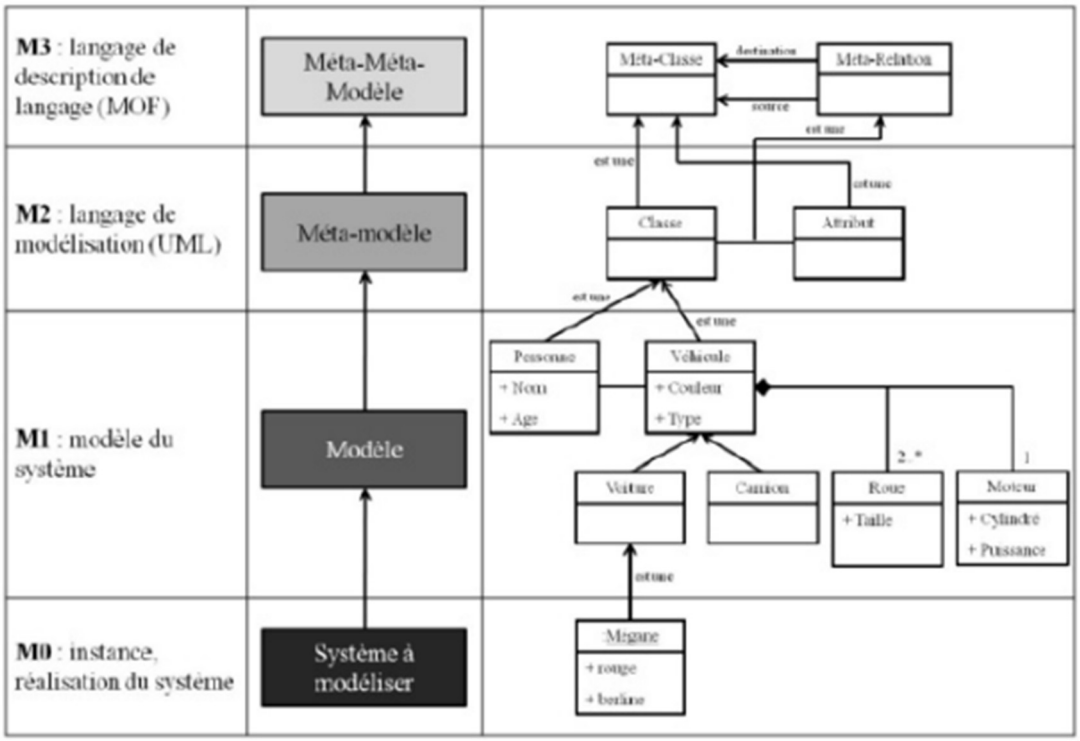
— Transformation

La transformation consiste à passer d'un modèle à un autre tous deux conformes à leurs métamodèles respectifs. Un jeu de règles définit le cadre de la transformation en précisant comment

un élément du méta-modèle source sera traduit en un élément du méta-modèle destinataire.

Plusieurs langages peuvent décrire les jeux de règles, nous en citerons l'EMF (Eclipse Modeling Framework) ou encore le standard de l'OMG QVT (Query Views Transformation).

Il est à noter, toutefois, que MDA n'est pas une méthodologie de conception mais plutôt un cadre générique pour mettre en application l'utilisation de modèles dans un processus de développement d'un système.



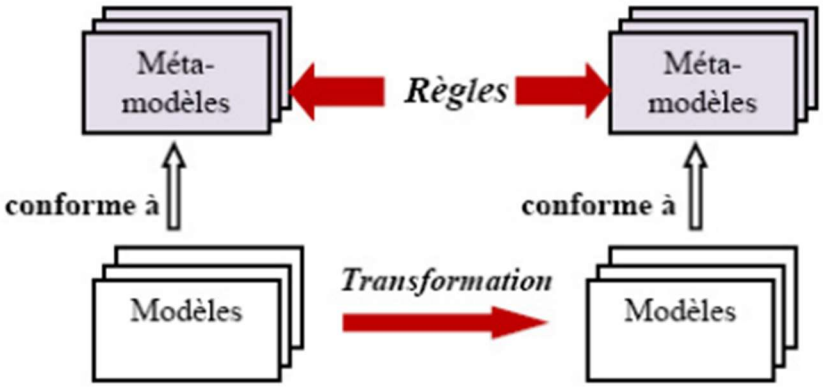


FIGURE 2.5: Transformation de modèles.

— ACCORD|UML

La méthodologie ACCORD|UML permet le développement de modèles d'applications temps réel distribuées et embarquées. Basée sur UML et dirigée par les modèles, son objectif est de masquer autant que possibles les détails d'implémentation. ACCORD|UML prend en compte la modularité au niveau de la structure ainsi que le parallélisme au niveau des services Trois phases constituent le processus de la méthodoogie ACCORD|UML : une première phase d'analyse permet de produire le modèle de haut niveau PAM (Preliminary Analysis Model). Se basant sur ce dernier, la seconde phase d'analyse qui est plus détaillée se charge de produire le DAM (Detailed Analysis Model). Enfin, la phase de prototypage produit le PrM (Prototype Model). Celui-ci étant un modèle complet, l'atelier ACCORD peut alors procéder à une génération automatique du code. Concernant le temps réel, la méthodologie ACCORD|UML définit deux concepts qui sont RealTimeObject et RealTimeFeature.

— GASPARD2

GASPARD2 (Graphical Array Specification for Parallel and Distributed Computing) est un environnement qui permet une modélisation MDA associant une modélisation UML avec le profil MARTE dédié à la modélisation de systèmes embarqués temps réel.

L'approche est particulièrement adaptée à la modélisation d'applications massivement parallèles et distribuées supportées par des plateformes d'exécution homogènes. GASPARD2 permet la génération de code VHDL et SystemC à partir de modèles UML. L'aspect reconfigurable d'un système aux deux niveaux applicatif et matériel est également pris en compte.

— ModES

ModES (Model-driven dESign approach) est une approche basée sur l'IDM. A la fois une méthodologie et un ensemble d'outils de conception, d'estimation et de génération de code, ModES permet de concevoir des systèmes embarqués.

Des modèles d'application ainsi que des modèles de plateforme, spécifiés dans le langage UML, sont en premier lieu transformés en des modèles conformes respectivement au méta-modèle d'application interne IAMM (Internal Application Meta-Model) et au méta-modèle de plateforme interne IPMM (Internal Platform Meta-Model). Une seconde phase consiste à réaliser un mapping entre ces deux modèles conformément au méta-modèle de mapping MMM (Mapping Meta Model). La troisième phase se charge enfin de transformer les modèles en des modèles d'implantation conformes à un méta-modèle d'implantation IMM (Implementation MetaModel). Des outils permettent d'analyser les propriétés du système telles que la consommation d'énergie, l'empreinte mémoire . . . etc.

VI. Techniques formelles

En génie logiciel, on parle de technique formelle lorsque celle-ci utilise un langage formel dans la définition et la manipulation des différentes entités ainsi que dans le système de preuve qu'elle possède.

Les systèmes embarqués temps réel étant des systèmes particulièrement à contraintes, beaucoup de travaux proposent des techniques formelles afin de s'assurer que les contraintes exigées soient satisfaites. Lors du développement, le recours aux techniques formelles permet d'une part de s'assurer que les systèmes répondent aux exigences des cahiers des charges et que les contraintes exigées sont respectées et de vérifier d'autre part que le développement en soi est correct.

En effet, on distingue deux cadres d'utilisation des techniques formelles :

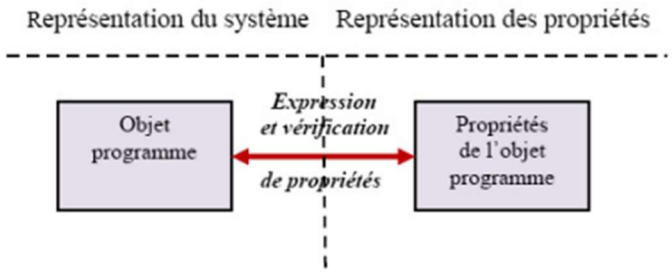


FIGURE 2.8: Techniques formelles pour la vérification.

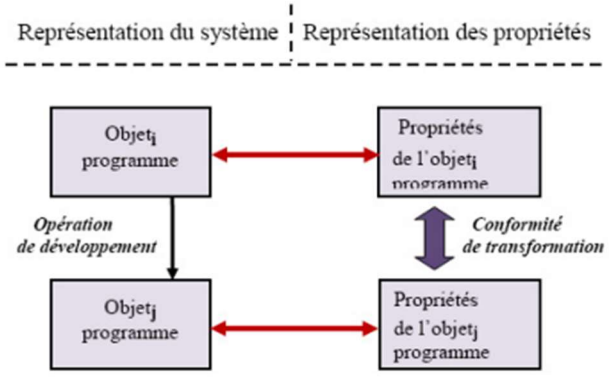


FIGURE 2.9: Techniques formelles lors d'une transformation.

— Vérification des propriétés d'une application :

Les propriétés d'une application sont exprimées tel que leur représentation formelle soit possible.

Il s'agit alors de vérifier la correction du modèle (Figure 2.8).

— Maintien des propriétés du programme lors d'une opération de transformation :

Il s'agit, dans ce cas, de vérifier la conformité de la transformation d'un modèle (Figure 2.9), autrement dit, c'est le processus de développement en soi qui est vérifié.

Bien que la démarche formelle soit fastidieuse, quelques travaux y ont eu recours :

— PROSEUS

PROSEUS (Development method for PROtotyping embedded SystEms by using UML and SDL) est une méthode de développement de systèmes embarqués basée sur le langage SDL. Afin d'intégrer les contraintes temps réel dans le modèle du système, l'approche offre un typage des signaux échangés entre système et évènements qui modélise les contraintes de temps de l'application à développer. La définition de l'architecture matérielle et logicielle s'appuie sur des structures types définies en SDL qui représentent les unités de traitement du système, les médiums de communication et l'environnement.

— MeMVaTEx

MeMVaTEx (Méthode de Modélisation pour la Validation et la Traçabilité des Exigences) est une méthodologie basée sur le profil UML, elle prend en compte le traitement des exigences en utilisant les diagrammes SysML. Les exigences sont modélisées comme des classes et demeurent telles quelles durant tout le cycle de développement.

Quelques critères de comparaison

Le tableau suivant est un récapitulatif des approches soulignant quelques critères à savoir l'élément central de l'approche, le type de processus méthodologique adopté ainsi que le type d'outils utilisé pour la modélisation.

Description des méthodologies			
Critères de description / **Approches de développement**	*Elément central*	*Processus*	*Outils*
GL classique	étape	complet	graphique/formel
Co-design	composant	complet	graphique/formel
Orientées UML	profile UML	complet/partiel	graphique/formel
A base de composant	composant	complet	graphique/formel
IDM	modèle	complet	graphique/formel
Techniques formelle	syntaxe formelle	complet	formel