République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université de Guelma

Faculté des Mathématiques, d'Informatique et des Sciences de la matière

Mémoire de Master

Département d'Informatique

Spécialité : Ingénierie des médias

Thème :

Génération Automatique des Modèles de Simulation SystemC à Partir des Profils UML

Présenté par: Khattab Ahmed.

Berkani Nadjet.

Sous la direction de:

Mr. Berrehouma Nabil.

Juin 2012

Remerciement

Nous tenons à remercies en premier lieu Allah qui nous a donné vie et santé pour le parachèvement de ce modeste travail.

*Nous remercions après de tout cœur notre encadreur « **Mr Berrehouma Nabil** » pour son soutien, sa sympathie, ses encouragements, la confiance qu'il nous témoignée en acceptant de diriger ce travail et pour avoir mis à notre disposition ses conseils pour une meilleure maitrise du sujet.*

Nous remercions nos familles qui nous ont toujours donné la possibilité de faire ce que nous voulions durant nos études et qui ont toujours cru en nous.

En fin, nous remercions tous ceux qui ont contribué à ce travail par leurs remarques, leurs suggestions et leurs soutiens.

Khattab Ahmed

Berkani Nadjet

Dédicace

Je dédie ce modeste travail de mes années d'étude :

A mon exemple supérieur dans ma vie mon cher père qui m'a toujours servi de modèle et des baux principes pour une vie idéale par sa patience, ses conseils, sa compréhension et pour m'avoir donné la possibilité de faire ce que je voulais et pour leur soutien sans faille tout au long de mes années d'étude.

A ma très chère mère qui m'a assuré un soutien inconditionnel par ses encouragements sur tous les plans, sa disponibilité, leur affection et leur grand amour aussi.

A mon frère Mohamed Amine et mes sœurs que j'aime beaucoup et qui n'ont jamais cessé de renouveler qu'ils ont en moi, et surtout Amina, Fayza, la fille de ma tente la princesse « Tasnim», la belle tente leila et son fil « Walid», mes tente wassila et son enfant « Yasser», souhila et ses enfants « Ahmed, Meryem», zinouba, et mes oncles Sassi et ses enfants « Mouhamed, Anis, Khayro», Taher et ses enfants « Akram, Wisal, Haroun, Kawther», Younes, Ali, Tayeb, rachid, malek, mahjoub.

A mon binôme Ahmed, mon intime la princesse «Mouna», mes amies Najla, Meryem, Majda, Oumaima, Houssem, Azzedine, Amine, Rachid, Cherif, Abd lmomen, brahim.

Berkani Nadjet

Dédicace

Je dédie ce modeste travail de mes années d'étude :

A mon exemple supérieur dans ma vie mon cher père qui m'a toujours servi de modèle et des baux principes pour une vie idéale par sa patience, ses conseils, sa compréhension et pour m'avoir donné la possibilité de faire ce que je voulais et pour leur soutien sans faille tout au long de mes années d'étude.

A ma très chère mère qui m'a assuré un soutien inconditionnel par ses encouragements sur tous les plans, sa disponibilité, leur affection et leur grand amour aussi.

A mes frères Karim, Brahim, Khaled, et ma sœur Ghalia que j'aime beaucoup et à mes nièces et neveux Chirine, Dania, Amina et Amdjed

A mon binôme Nadjette, à ma chère princesse Hadjer que j'aime beaucoup a Aida, Loubna, Mehieddine, Khaled, Nabil, Ayoub, Ahmed et a tous mes amis.

Khattab Ahmed

Résumé

La conception des systèmes multiprocesseurs mono-puce (MPSOC) est devenue de plus en plus complexe et très coûteuse. Afin de pouvoir maîtriser la complexité toujours croissante de Tels systèmes, les spécialistes du domaine insistent sur la nécessité de lever le niveau d'abstraction et de faire recourir aux technologies maîtrisées du génie logiciel telles que le paradigme visuel objet et les techniques formelles. Dans ce domaine, l'UML est considéré aujourd'hui comme l'un des langages les plus prometteurs dans le développement des systèmes informatiques. Notre travail est la suite d'une étude précédente. Dans le cadre de ce mémoire nous intéresserons sur la transformation de ces modèles et la génération du code SystemC en utilisant le langage déclaratif ATL. A ce propos nous avons utilisées les outils MDA de l'environnement Eclipse pour définir un méta-modèle qui définit la syntaxe régissant les modèles des systèmes embarqués en se basant sur un langage de description de hardware bien connu : SystemC. Notre éditeur est renforcé par l'option de génération automatique du code SystemC à partir des modèles graphiques édités.

Mots Clés :

Systèmes embarqués, MPSoC, UML, SystemC, Eclipse, EMF.

Sommaire

Chapitre3 : EMF

Chapitre 4 :SystemC

Chapitre 5 : Conception et implémentation

Liste des figures

Liste des tableaux

Glossaire

JET : (Java Emitter Tempalte) JET est un moteur de Template générique qui peut être utilisé pour générer du XML, code source Java et autre sortie à partir de modèles.

API : Une interface de programmation (Application Programming Interface) est un ensemble de fonctions, procédures ou classes mises à disposition des programmes informatiques par une bibliothèque logicielle, un système d'exploitation ou un service. La connaissance des APIs est indispensable à l'interopérabilité entre les composants logiciels.

AWT : Abstract Window Toolkit, ensemble de classes spécialisées dans la construction d'interface graphique pour des applications écrites en Java.

CIM : (Computation Independent Model), Modèle Indépendant de l'Informatisation.

CPU : Central Processing Unit, « Unité centrale de traitement »), est le composant de l'ordinateur qui exécute les programmes informatiques.

DMA : mémoire ou DMA (sigle anglais de Direct Memory Access) est un procédé informatique où des données circulant de ou vers un périphérique ...

Eclipse : Le projet Eclipse : http://www.eclipsetotale.com/.

EMF : Eclipse Modeling Framework est un framework de modélisation et de simplification de génération de code pour la construction d'outils et d'autres applications basées sur une structure de modèle de données.

Framework : Infrastructure logicielle qui facilite la conception des applications par l'utilisation de bibliothèques de classes ou de générateurs de programmes.

Introduction Générale

Face à la complexité croissante de spécification, conception, implémentation et vérification des systèmes embarqués, Des langages de modélisation des systèmes qui permettent un niveau assez élevé d'abstraction se trouvent de plus en plus indispensables. Ce qui permet de créer des modèles de haut niveau rapidement, faire des simulations pour optimiser la conception de ces modèles, investiguer l'efficacité des différents algorithmes avant leur implémentation en hardware.

SystemC est un HDL (Hardware description Langage) où la modélisation des parties software et hardware d'un système embaqué s'évoluent en parallèle. SystemC est une extension de C++ qui offre la possibilité de modéliser des concepts hardware (module, ports, signaux, horloges, etc.) pour supporter ses caractéristiques inhérentes : la concurrence, la synchronisation et la communication. Par ailleurs, il englobe un noyau de simulation à événements discrets qui rend possible des simulations ultra-rapides des systèmes embarqués de grande complexité.

L'adoption d'un langage de modélisation des systèmes embarqués dans un niveau d'abstraction élevé comme SystemC nécessite la définition d'une méthodologie de design adaptée. Des outils formels en première phase de spécification telle que FSM, RdP ou UML se trouvent très pratique pour donner des définitions précises de la structure et la dynamique des différents composants du système. Ils sont aussi de grande importance durant toutes les étapes intermédiaire du processus de raffinement pour les vérifications de ses propriétés. Récemment, UML avec sa puissance d'expression et sa capacité d'extension a suscité l'intérêt de la communauté des systèmes embarqués, en effet, UML offre une variété des formalismes graphiques pour modéliser la structure et le comportement des systèmes qui distinguent dans leurs sémantiques et leurs domaines d'application. Cependant, générer du code SystemC à partir de formalismes graphique se voit une tache excessivement délicate du au gap large entre UML et SystemC.

L'objectif de ce travail est de continuer l'étude des étudiants de l'année passée qui sont basés sur la conception et implémentation d'un environnement visuel -basé sur les méta-modèles. A partir de ce travail nous intéressons pour transformer ces modèles et générer le code SystemC en utilisant le langage déclaratif JET.

De façon générale, le contexte de notre travail se situe à l'intersection de trois axes. Le premier axe est relatif à la définition des méta-modèles pour la modélisation structurelle et comportementales des systèmes embarqués. Le deuxième axe recouvre la conception et implémentation d'un environnement visuel -basé sur les méta-modèles déjà définies. En s'appuyant sur un langage de transformation des modèles, le troisième axe est lié à la génération du code SystemC.

Le reste de ce travail est organisé comme suite :

- ✓ **Chapitre 1 :** Les profiles UMLs ainsi que la modélisation en UML, Méta-Modèle et l'architecture a quatre niveaux de MDA
- ✓ **Chapitre 2 :** Les profiles UMLs pour MPSOC ainsi que leur caractéristiques et le model de calcul (MOC).
- ✓ **Chapitre 3 :** Le framework EMF Eclipse pour la modélisation de notre modèle de donnée, les étapes de parcours d'EMF, les stéréotypes et la validation d'EMF.
- ✓ **Chapitre 4:** Le langage de description SystemC Qui sera le langage cible dans notre processus de génération automatique du code.
- ✓ **Chapitre 5 :** Conception et implémentation de notre application et la transformation des modèles et la génération du code SystemC.

Chapitre 1
Les Profils UML

Chapitre 1 : Les Profils UML

I. Introduction

La description de la programmation par objets a fait ressortir l'étendue du travail conceptuel nécessaire : définition des classes, de leurs relations, des attributs et méthodes, des interfaces etc.

Pour programmer une application, il ne convient pas de se lancer tête baissée dans l'écriture du code : il faut d'abord organiser ses idées, les documenter, puis organiser la réalisation en définissant les modules et étapes de la réalisation. C'est cette démarche antérieure à l'écriture que l'on appelle *modélisation* ; son produit est un *modèle*, et pour le langage on utilise un langage de modélisation tel qu'UML.

UML (*UnifiedModelingLanguage*). L'adjectif *unified* est là pour marquer qu'UML unifie, et donc remplace. En fait, et comme son nom l'indique, UML n'a pas l'ambition d'être exactement une méthode : c'est un langage. [11]

II. La Modélisation en UML

La modélisation consiste à créer une représentation simplifiée d'un problème: **le modèle**.

Grâce au modèle il est possible de représenter simplement un problème, un concept et le simuler. La modélisation comporte deux composantes :

- L'analyse, c'est-à-dire l'étude du problème
- la conception, soit la mise au point d'une solution au problème

Le modèle constitue ainsi une représentation possible du système pour un point de vue donné.

La Modélisation c'est une étape préalable en. Elle se fait en utilisant les outils conceptuels d'UML. Elle aboutit à un ensemble de modèles de la future application, représentés par des diagrammes de classes. Chaque domaine fonctionnel de l'application est représenté indépendamment des autres.

A partir de ces diagrammes de classes UML, des modèles manipulables sont générés (format XML).[37]

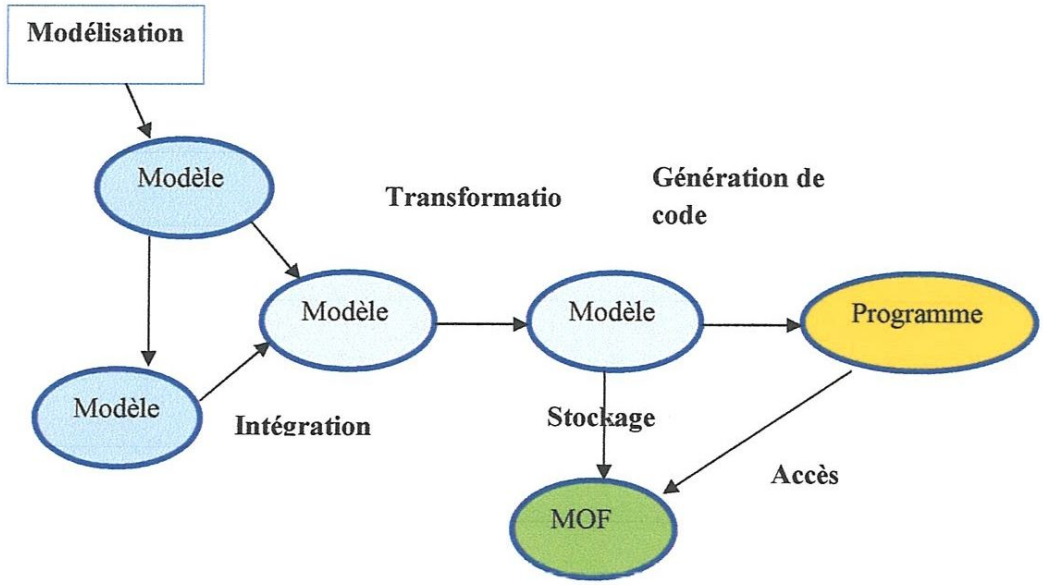


Figure 1.1 : *Les étapes de la transformation dans un processus MDA*

III. Méta-Modèle

Tout modèle doit respecter la structure définie par son méta-modèle. Par exemple, le méta-modèle UML défini que les modèles UML contiennent des packages, leurs packages des classes, leurs classes des attributs et des opérations....etc. Les méta-modèles fournissent la définition des entités d'un modèle, ainsi que les propriétés de leurs connexions et de leurs règles de cohérence. MOF les représente sous forme de diagrammes de classes.

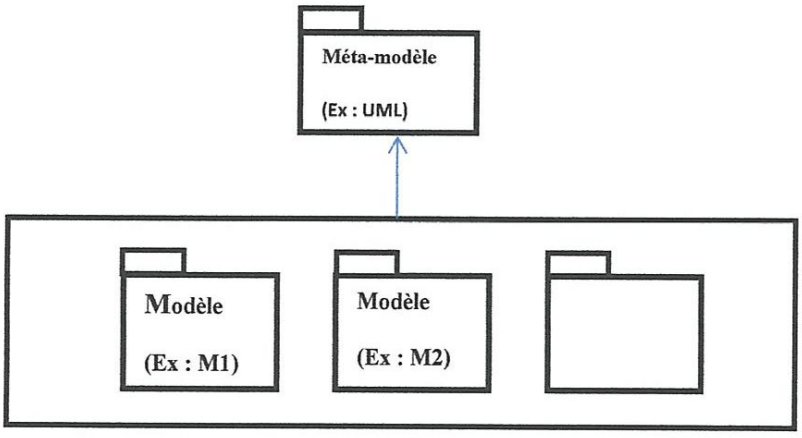


Figure 1.2: *Relation entre modèle et méta-modèle*

Je résume que les méta-modèles fournissent la définition des entités d'un modèle, ainsi que les propriétés de leurs connexions et de leurs règles de cohérence. Les méta-modèles Sont au cœur de MDA.

Ils permettent de pérenniser la structure des modèles car ils sont eux-mêmes représentés sous forme de modèles (diagrammes de classes). Ils permettent d'associer des traitements aux modèles grâce notamment aux méta-opérations des méta-classes. Les liens entre méta-modèles permettent enfin de bien séparer les aspects métier des aspects techniques et donc de prendre en compte les plates-formes d'exécution.

IV. L'architecture à quatre niveaux de MDA

Pour que la définition de l'infrastructure soit complète il faut définir les relations entre les niveaux de modélisation. [10]

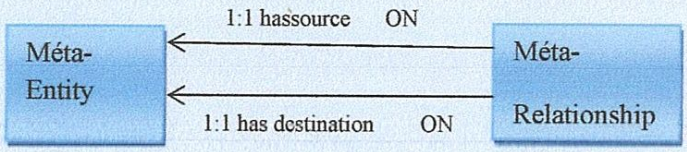
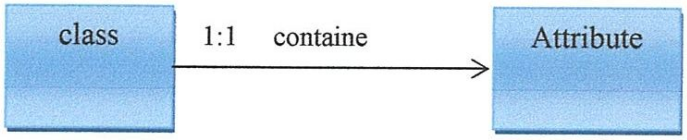
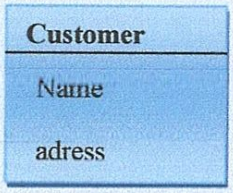
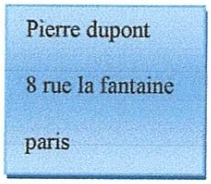
Layer	Conents	Example
Méta-Méta-Model	Language to describe Méta-models	
Meta-Model	Language to describe Models	
Model	Language to describe Data	
Data	Data	

Tableau 1.1 : *Architecture à quatre niveaux de MDA*

Associera chaque entité du niveau Mn sa définition au niveau Mn+1, le méta-méta-modèle est réflexif et se décrit lui-même, la réflexion implique des transformations du modèle, interprétation des entités de niveaux M2, M3 au niveau M1.

La figure 1.3 représente cette architecture. On y voit le niveau M0, contenant les entités à modéliser, ici les applications informatiques, le niveau M1, contenant les différents modèles de l'application informatique, le niveau M2, contenant les différents méta-modèles qui ont été utilisés, et le niveau M3, contenant le méta-méta-modèle qui a permis de définir uniformément les méta-modèles.

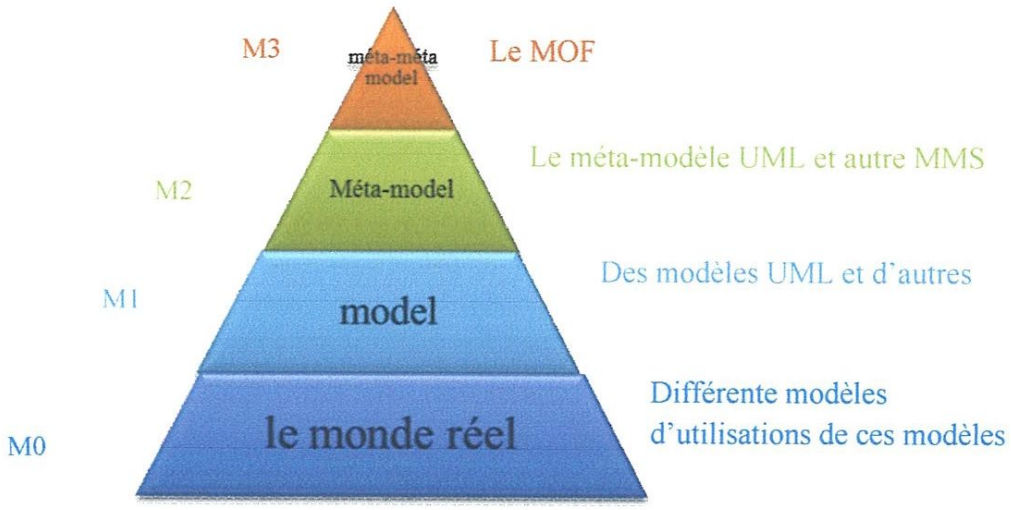


Figure 1.3: *Architecture à quatre niveaux de MDA.*

V. les profils UML

- Un profile UML est une spécialisation du modèle UML pour un domaine d'utilisation particulier.
- Il regroupe de manière cohérente les extensions du modèle UML.
- Un profile est composé de stéréotypes, de tagged values et des contraintes.
- Les profiles UML peuvent hériter d'autres profiles, avoir des dépendances entre eux, ou encore être regroupés. [16]

VI. Les extensions de l'UML

UML est un langage de modélisation "généraliste" pouvant être adapté à chaque domaine grâce aux mécanismes d'extensibilité offerts par ce langage tels que stéréotypes, tagged values (valeurs marquées) et contraintes. Les extensions UML ciblant un domaine

particulier forment des profils UML. Les mécanismes d'extensibilité offerts par UML permettent d'étendre UML sans modifier le méta modèle UML. [17]

VI.1 Stéréotypes

Un stéréotype est une chaîne de caractères qui associée à un élément UML existant permet de désigner un nouveau type d'élément UML et de Créer des nouveaux concepts UML à partir des concepts standard.

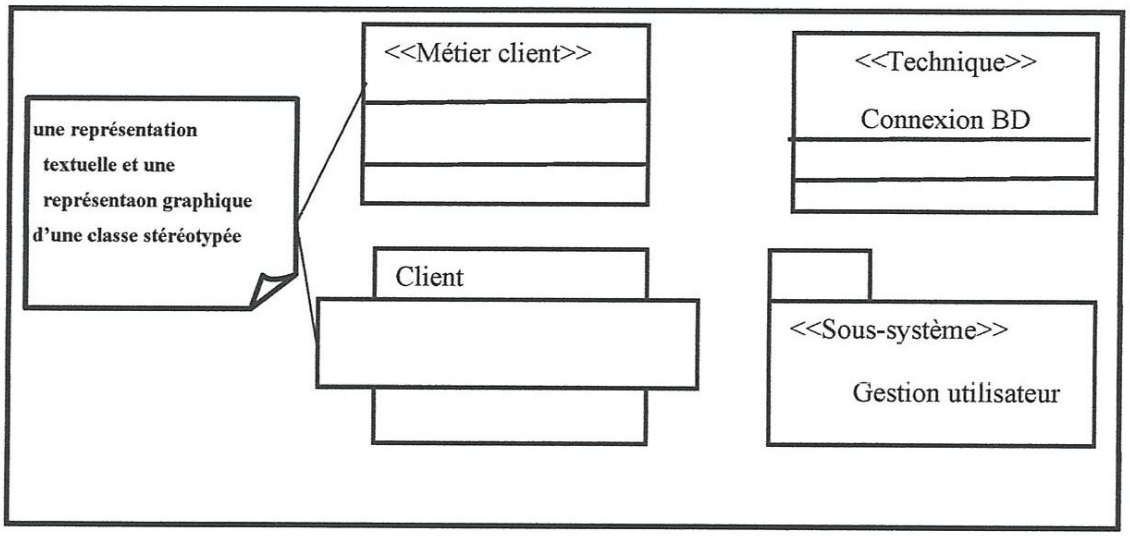


Figure 1.4: *Exemple d'un stéréotype.*

- Ajout de nouveaux éléments de modélisation dans le contexte métier ou technique.
- Une classe stéréotypée porte la sémantique du stéréotype.
- Les stéréotypes ne peuvent être utilisés que conformément à leur définition

VI.2 Tagged Values

- Les valeurs marquées sont principalement utilisées pour ajouter des informations sur les classes. En fait c'une annotation des éléments de modélisation
- Une tagged value peut être vue comme un nouvel méta-attribut.
- Peuvent être définies pour des éléments existants ou des stéréotypes
- Vitalisées sous la forme : nom de la propriété, valeur
- Exemples
  - {virtual}, {primarykey}
- Il est possible d'associer des tagged values à tout concept UML (Classe, Attribut, Association, Use Case, Component, Part…)

VI.3 Etiquette

Une étiquette est une chaîne de caractères qui associée à un élément UML permet de lui ajouter une propriété et sa valeur associée.

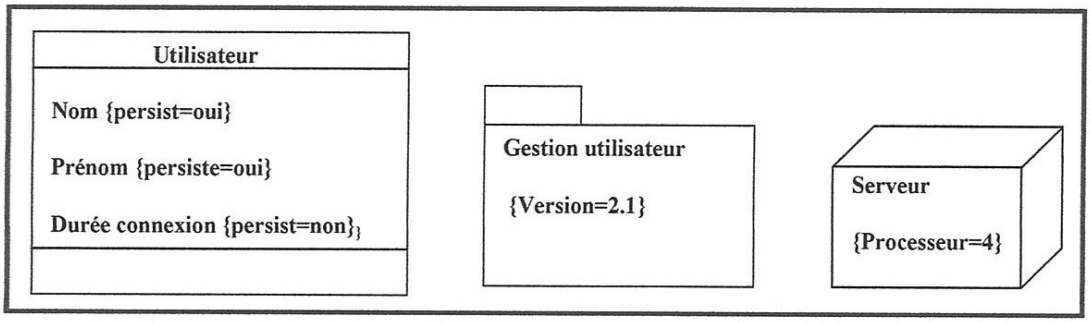


Figure 1.5: *Exemple d'une Etiquette*

Etiquette c'est la définition explicite d'une propriété sous la forme d'un couple nom-valeur. Certaines étiquettes sont prédéfinies dans le langage UML ; d'autres peuvent être définies par l'utilisateur. Les étiquettes constituent un des trois mécanismes d'extension du langage UML.

VI.4 Les Contraints

Une contrainte est une chaîne de caractères qui associée à un élément UML permet d'ajouter de nouvelles règles ou de modifier des règles existantes.

Contrainte c'est une condition sémantique ou restriction. Certaines contraintes sont prédéfinies dans le langage UML, d'autres peuvent être définies par l'utilisateur. Les contraintes constituent un des trois mécanismes d'extension du langage UML.

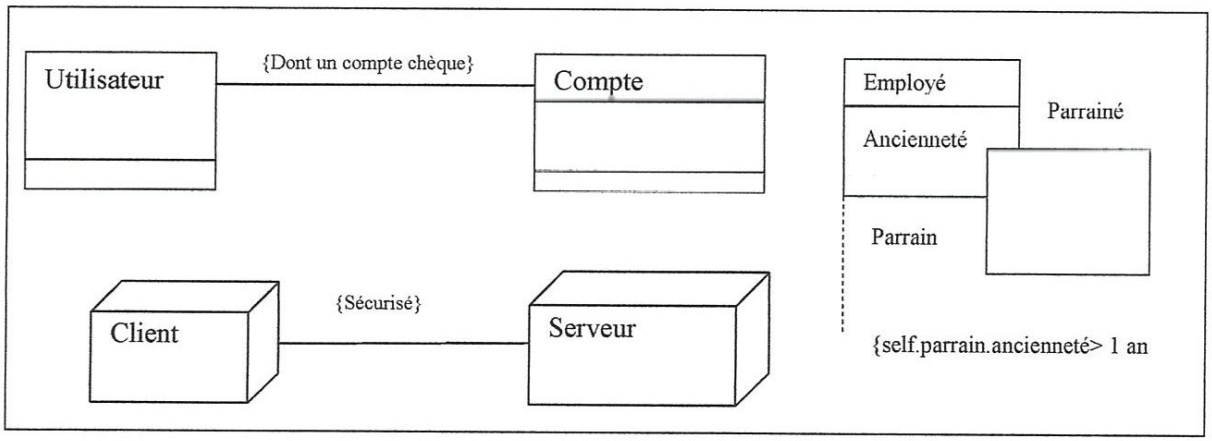


Figure 1.6: *Exemple d'un Contrainte*

VII. MOF

Un ensemble d'éléments de modélisation permet de représenté des Méta-Modèles (UML, Java, OMG-IDL, méta modèle work flowWfMF, ...).

Décomposé en 2 parties :

- **E-MOF** : essential MOF (Méta-modèle noyau)
- **C-MOF** : complète MOF (Méta-modèle plus complet)

On dit qu'un Méta-modèle c'est une ensemble d'instances de méta-éléments du MOF associées entre elles

- **UML Infrastructure**

C'est les constructions de langage de base nécessaires pour UML.

- **UML superstructure**

Qui définit le niveau requis pour l'utilisateur construit UML.

Les deux spécifications complémentaires constituent une spécification complète du langage de modélisation UML. [18]

VIII. Conclusion

Nous avons présenté dans ce chapitre, une démarche pour supporter la tâche de conception des architectures logicielles utilisant le formalisme UML. Nous pensons qu'UML serait une approche complémentaire à ArchJava puisque c'est un standard de modélisation largement utilisé par les industriels et les chercheurs. De plus, il a introduit dans sa version 2.0 un modèle de composant abstrait inspiré des ADLs pour prendre en compte la description des architectures logicielles.

La description architecturale est faite suivant un profil UML et ensuite transformée automatiquement en ArchJava pour garantir la consistance entre le modèle architectural UML et son implantation. Cette démarche a pour but de résoudre le problème de découplage entre la spécification des architecturcs logicicllcs et leur implantation dans un langage de programmation cible.

Chapitre 2
UML pour MPSOC

Chapitre 2 : UML pour MPSOC

I. Introduction

La conception de systèmes embarqués sur puces, pour une partie de ces applications, présente de nombreux avantages : petite taille, faible consommation d'énergie et performance de calculs raisonnable.

On parle de systèmes-sur-puce, en anglais System-on-Chip(SoC). Néanmoins, les technologies de conceptions adaptées pour ces SoC ne sont pas triviales. En effet, on a besoin d'aller au-delà des limitations en puissance de calculs et en consommation d'énergie induites par certaines plateformes, notamment les -processeur.

Actuellement, plusieurs processeurs peuvent être intégrés sur une même puce. Ce genre de SoC est qualifié de MPSoC pour Multi-Processor SoC. Par exemple, un MPSoC peut être composé de plusieurs types de processeurs tels que les microprocesseurs, les DSP (Digital Signal Processor) et les accélérateurs matériels; amenant à la conception des systèmes multiprocesseurs hétérogènes sur puce (MPSoC).

Cette richesse dans le nombre et le type de processeurs augmente significativement la puissance de calcul et améliore les performances. Cela permet d'implémenter des applications complexes tout en conservant un haut niveau de fiabilité et de sécurité.

Malheureusement, les approches existantes pour la conception de ces systèmes sur des MPSoC sont loin d'être satisfaisantes.

Bref, en raison de la complexité et de l'hétérogénéité de MP Soc, il est nécessaire de mettre en place des méthodes et des outils facilitant la conception de ces systèmes. [9]

I.1 Problématique

Une exploitation intelligente du parallélisme potentiel des ressources physiques permet d'avoir des exécutions plus performantes du système. Cependant, la conception demeure de plus en plus complexe nécessitant beaucoup de temps. Cela est dû principalement au grand nombre de ressources physiques à gérer et à la grande quantité de données à traiter dans le cas des applications hautes performances. D'où la nécessité d'un modèle de calcul efficace et facile à programmer.

I.2 Systèmes Mono Puce

Un système mono puce, appelé encore Soc (System-on-Chip) ou système sur puce, Désigne l'intégration d'un système complet sur une seule pièce de silicium.

Ce mot provient du verbe ′ synistanai ‘ qui veut dire combiner, établir, rassembler. Une définition plus appropriée de système mono puce serait : Un système complet sur une seule pièce de silicium, résultant de la cohabitation sur silicium de nombreuses fonctions déjà complexes en elles-mêmes : processeurs, DSP, mémoires, bus, convertisseurs, blocs analogiques, etc. Il doit comporter au minimum une unité de traitement de logiciel (un CPU) et doit dépendre d'aucun (ou de très peu) de composants externes pour exécuter sa tâche.

La Figure 2.1 présente un exemple de système mono puce typique. Il se compose d'un cœur de processeur (CPU), d'un processeur de signal numérique (DSP), de la mémoire embarquée, et de quelques périphériques tels qu'un DMA et un contrôleur d'E/S. Le CPU peut exécuter plusieurs tâches.

Le DSP est habituellement responsable de décharger le CPU en faisant le calcul numérique sur les signaux de provenance du convertisseur A/N.[9]

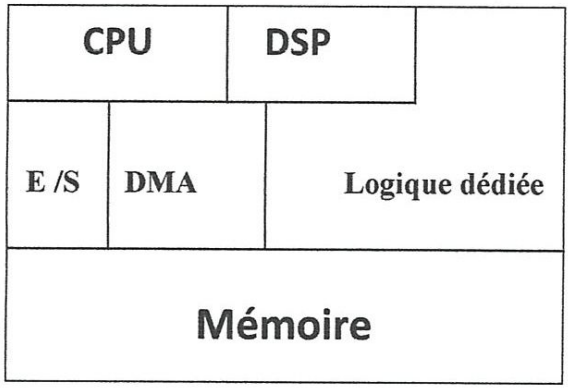


Figure 2.1 : *Un exemple de système mono puce (Soc)*

I.3 Caractéristiques Des Socs

I.3.1 : Hétérogénéité des composants du Soc

Le système mono puce n'encapsule pas des systèmes purement électroniques, il peut intégrer aussi des composants non électroniques. La portion électronique dominante dans un Soc peut être formée d'une partie analogique et d'une partie numérique

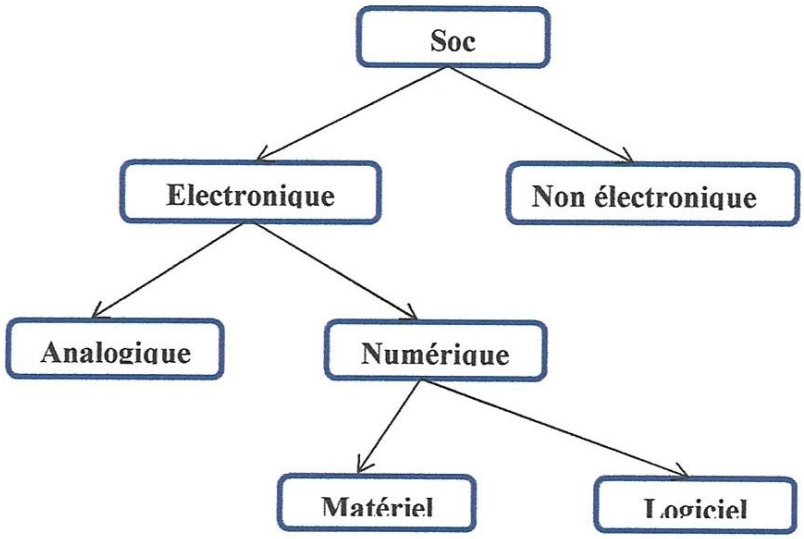


Figure 2.2 : *Les composants hétérogènes formants un Soc*

Le téléphone portable présente un exemple quotidien de tels systèmes hybrides où la partie radio (analogique) et la partie de traitement de signal de l'interface utilisateur (numérique) cohabitent sur la même puce.

La partie numérique, peut être décomposée selon qu'il s'agit d'un traitement effectué par un programme logiciel tournant sur un (ou plusieurs) processeur(s) représentant ainsi la partie logicielle, ou d'une fonction directement ciblée en matériel. Ces divers choix sont illustrés dans la Figure 2.2. [8]

I.3.2 L'embarquement du Soc dans un Système Complet

Un système embarqué est Un système qui englobe sur une seule puce de silicium des composantes logicielles et matérielles.

Les systèmes mono puces sont des systèmes embarqués (la réciproque ne tient pas forcément) dans le sens où ils sont toujours appelés à évoluer dans le contexte d'un système plus large qui constitue l'environnement du système mono puce.

L'interaction entre le Soc (numérique) est les éléments de l'environnement (capteurs et actionneurs) se fait généralement via des convertisseurs analogique/numérique.

I.3.3 Réduction de la Latence

Les processus de fabrication d'aujourd'hui (de plus en plus fins) permettent de combiner la logique et la mémoire sur une seule puce, réduisant ainsi le temps global des accès mémoire.

Étant donné que le besoin en mémoire de l'application ne dépasse pas la taille de la mémoire embarquée sur la puce, la latence de mémoire sera réduite grâce à l'élimination du trafic de données entre des puces Séparées.

I.3.4 Réduction du Coût

Le coût de l'encapsulation représente environ 50% du coût global du processus de fabrication de puce. Par rapport à un système- sur- carte ordinaire, un Soc emploie une seule puce réduisant le coût total d'encapsulation, et de ce fait, le coût total de fabrication.

Ces caractéristiques aussi bien que la faible consommation et la courte durée de conception permettent une mise sur le marché rapide de produits plus économiques et plus performants. [8]

II. Du Monoprocesseur au Multiprocesseur

II.1 Qu'est-ce qu'un MP Soc ?

Un MP Soc représente un système multiprocesseur embarqué sur une seule puce, conçu pour satisfaire les exigences d'une application embarquée. Il se compose habituellement d'un ou de plusieurs processeurs de nature différente (des microprocesseurs, des processeurs de traitement de signal (DSP), des microcontrôleurs), de mémoires et de quelques périphériques tels qu'un contrôleur d'accès mémoire (DMA) et un contrôleur d'entrées/sorties; le tout étant intégré sur une même puce.

II.2 Domaines d'application des Socs

Le domaine d'application couvert par des systèmes multiprocesseurs sur puce est très vaste, puisqu'il comprend toutes les applications dont les impératifs économiques et les contraintes de conception nécessitent une architecture mixte, miniature et efficace.

- Les applications télécommunications et multimédias (modems, téléphones mobiles, les décodeurs audio et vidéo, etc.
- Il constitue l'une des plus intéressantes applications, car il nécessite la production au moindre coût de systèmes à contraintes strictes (performance, poids, consommation, etc.).

II.3 Principales étapes de la conception des MP Socs

Il s'agit essentiellement de la spécification, la modélisation, le partitionnement, le raffinement du logiciel, le raffinement du matériel, la génération d'interface entre le logiciel et le matériel (i.e raffinement des communications) et la génération de code ou le prototypage. La succession de ces étapes forme le flot typique d'une approche de conception des systèmes multiprocesseurs schématisée dans la Figure.2.3.

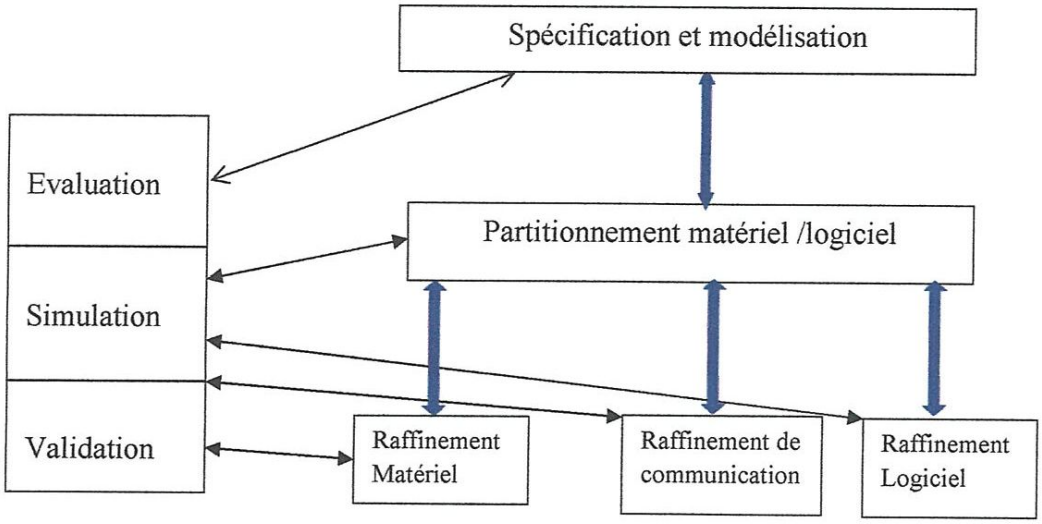


Figure 2.3 : *Flot de conception des Socs*

Après cette phase de spécification survient l'étape de partitionnement du système ayant pour but de décomposer ce dernier en trois parties :

- Une partie matérielle implémentée sous forme de circuits.
- Une partie logicielle implémentée sous forme d'un programme exécutable sur processeur.
- Une interface de communication entre les parties matérielles et logicielles.

Les trois parties obtenues doivent ensuite être vérifiées et validées avant de passer à la phase du raffinement et d'implémentation. Il est nécessaire de faire des retours aux étapes précédentes (feed-back), plus précisément à l'étape de partitionnement, tant que l'architecture obtenue ne répond pas aux contraintes auparavant fixées.

II.4 Caractéristiques d'un MP Soc

Un MP Soc est souvent dédié à l'exécution d'applications bien spécifiques. Il est cependant soumis à des contraintes physiques car les ressources peuvent être plus restreintes que dans un système fixe : par exemple le nombre et la fréquence des processeurs, la taille des mémoires.

- **Réduction de la consommation d'énergie** : un MP Soc doit généralement être optimisé pour la consommation d'énergie. Cela est dû au manque de source d'alimentation.
- **Fiabilité du système** : certains MP Soc sont de types critiques. Une panne mineure peut avoir des conséquences dramatiques telles que des morts, des dommages matériels importants, ou des conséquences graves pour l'environnement.
- **Amélioration des performances** : les performances de beaucoup de MP Soc disponibles aujourd'hui sont suffisamment élevées. Cependant, il y a encore un certain nombre d'applications nécessitant plus de performances, telles que la traduction automatique de voix, la radio logicielle, la reconnaissance d'images animées, etc.

- **Réduction du temps de mise à disposition sur le marché** : le temps de mise à disposition sur le marché (time-to-market) est un facteur important. Il est un des facteurs(le temps par exemple) qui déterminent le profit d'un MP Soc, une fois déployé sur le marché.
- **Réduction du coût de la conception** : la plus grande menace pour l'avenir du MP Soc est son coût de conception. Mis à part les coûts de fabrication, déployé un MP Soc sur le marché comporte plusieurs efforts tels que le marketing, la vente, l'administration et des frais divers. Cependant, le coût de conception reste énorme.

II.5 Implémentation

Nous appelons l'implémentation la réalisation physique du matériel (par la synthèse) et du logiciel exécutable (par la compilation).

Une caractéristique désirable des approches déco-design lors de l'implémentation est qu'il est préférable que les étapes successives de raffinement de la modélisation à la synthèse devraient toutes s'articuler autour du même modèle interne adopté en fixant à chaque étape certains détails d'implémentation.

Les informations architecturales concernant les détails d'implémentation seront prises en compte par le modèle lors de la phase d'implémentation.

II.6 Modèles de Calcul

Un modèle de calcul (MoC ou Model of Computation) représente un ensemble de lois qui régissent les interactions de composants dans un modèle. Plus généralement, c'est un ensemble de règles abstraites permettant au concepteur de créer des modèles selon une sémantique bien précise.

Les modèles de calcul les plus intéressants pour modéliser des systèmes de hautes performances (ou de traitement intensif de signal) doivent être en mesure de gérer la concurrence de composants, la réactivité et le temps d'exécution.

II.6.1 Classification des modèles de calcul

Afin de classifier les modèles de calcul, nous nous basons sur plusieurs critères et caractéristiques pour faire la comparaison

MOC	Structure de données	Accès donnée	Style	Saisie information
SDF[1]	multidimensional	Tableau	flot de donnée	Graphique
ARRAY-OL	multidimensional	Tableau	flot de donnée	Graphique
STREAM-IT	multidimensional	Tableau	Imperative	Textual
ALPHA	Polyhedron	Affine	Functional	Textual
SISAL[2]	multidimensional	Tableau	Functional	Textual
SAC[3]	multidimensional	Tableau	Imperative	Textual

Tableau 2.1 *: Classification des modèles de calcul*

II.7 Revue de Travaux existants

Nous présentons d'abord le langage de modélisation UML (Unified Modeling Language)

Qui est très répandu dans la modélisation de systèmes ainsi que des extensions sous forme de profils.

Ensuite, nous présentons des modèles de calcul dédiés aux traitements de signal intensif. Nous décrivons par la suite des environnements de conception à base de modèles et de plateforme qui utilisent certains de ces modèles de calcul. Enfin, des outils pour l'exploration de l'espace de conception sont présentés. [9]

III. UML (Unified Modeling Language)

Standardisé par l'OMG (Object Management Group), UML est dédié à la modélisation de n'importe quel système informatique. Sa force est de permettre de schématiser pratiquement tout et que ses modèles sont compréhensibles par une grande partie de gens. Son principal inconvénient est qu'il est très général. En particulier, il existe un grand nombre de points de variation sémantique, où la signification est laissée à l'appréciation du concepteur. D'où l'introduction de la notion de profil UML qui a apparue dans le Chapitre

[1]SDF ((abréviation de Synchronous Data Flow))

[2] SaC (abréviation de Single Assignment C)

[3]Sisal (abbreviation de Streams and Iteration in a Single Assignment Language)

precedent. Un profil est un moyen permettant de raffiner la sémantique d'UML pour la modélisation de systèmes bien précis.

Dans ce qui suit, on présente les profils UML pour les systèmes embarqués et les SOC les plus répandus. Parmi ces profiles on trouve SPT (Schedulability, Performance, and Time), Marte (Modeling and Analysis of Real-time and Embedded systems), SysML (System Modeling Language), UML4SystemC/C.

Bien que la plupart des profils ci-dessus proposent des concepts utiles pour la conception de MP Soc, ils ne prennent pas suffisamment en considération certains aspects cruciaux tels que l'analyse temporelle et la possibilité de spécifier du parallélisme massif. À notre connaissance, Marte est le seul profil qui offre un ensemble riche et satisfaisant de concepts pour la conception de MP Soc à haut niveau.

Nom	Cible	Modélisation
SPT	system temps reel	temps,ordonnancement, performance
MARTE	system temps reel	fonctionnalité, architecture, association,temps, ordonnancement, performance
SYSML	System general	specification, analyses, verification
UML4SOC	System sur puce	concepts porches de SystemC

Tableau 2.2 : *Classification de profils UML.*

IV. Profils UML2 pour Socs et Systèmes Embarqués

IV.1 Sys ML

IV.1.1 Qu'est-ce que SysML

Comme UML, SysML est un langage et non une méthode. SysML est un langage de modélisation graphique développé par l'OMG. SysML est un profil d'UML 2.0 fournissant aux ingénieurs un langage de modélisation allant bien au-delà des problématiques de l'informatique.

SysML intègre Les composants physiques de toutes **technologies,** les programmes, les données et les énergies, les personnes et les procédures et flux divers.

La généralisation des concepts utilisés en UML enrichis de quelques notions donnent SysML. Nous retrouvons des diagrammes de structure (diagrammes statiques) et des diagrammes de fonctionnement (diagrammes dynamiques) comme en UML.

L'objectif de cette partie est de montrer comment utiliser la notation SysML dans le cadre d'un processus complet partant des premiers contacts avec le client et les utilisateurs et allant jusqu'à l'exploitation de la solution.

IV.1.2 Passé d'UML à SysML

Le passage d'UML à SysML est très simple. Vous allez constater que les diagrammes sont moins nombreux et que SysML réutilise une bonne partie des diagrammes que vous connaissez déjà en UML.

IV.1.3 Objectifs de SysML

- SysML permettre à des acteurs de corps de métiers différents de collaborer autour d'un modèle commun pour définir un système.
- SysML est un moyen de regrouper dans un modèle commun à tous les corps de métiers, les spécifications, les contraintes, et les paramètres de l'ensemble du système.
- SysML n'aborde plus la conception avec la notion de classes mais avec la notion de blocs qui deviendront des parties mécaniques, électroniques, informatiques ou autres.
- SysML est fait pour spécifier et l'analyse la structure et le fonctionnement des systèmes.
- Décrire les systèmes et concevoir des systèmes composés de sous-systèmes.
- Vérifier et valider la faisabilité d'un système avant sa réalisation.

IV.1.4 Les avantages d'une modélisation SysML

- Partage des spécifications d'un système complexe entre tous les corps de métiers.
- Identification des risques et création d'une base d'analyse commune à tous les participants d'un projet.
- Facilite la gestion de projets complexes, l'évolutivité et la maintenabilité des systèmes complexes.

IV.1.5 Architecture SysML vs UML

Ci-dessous la description extraite de la spécification officielle de SysML

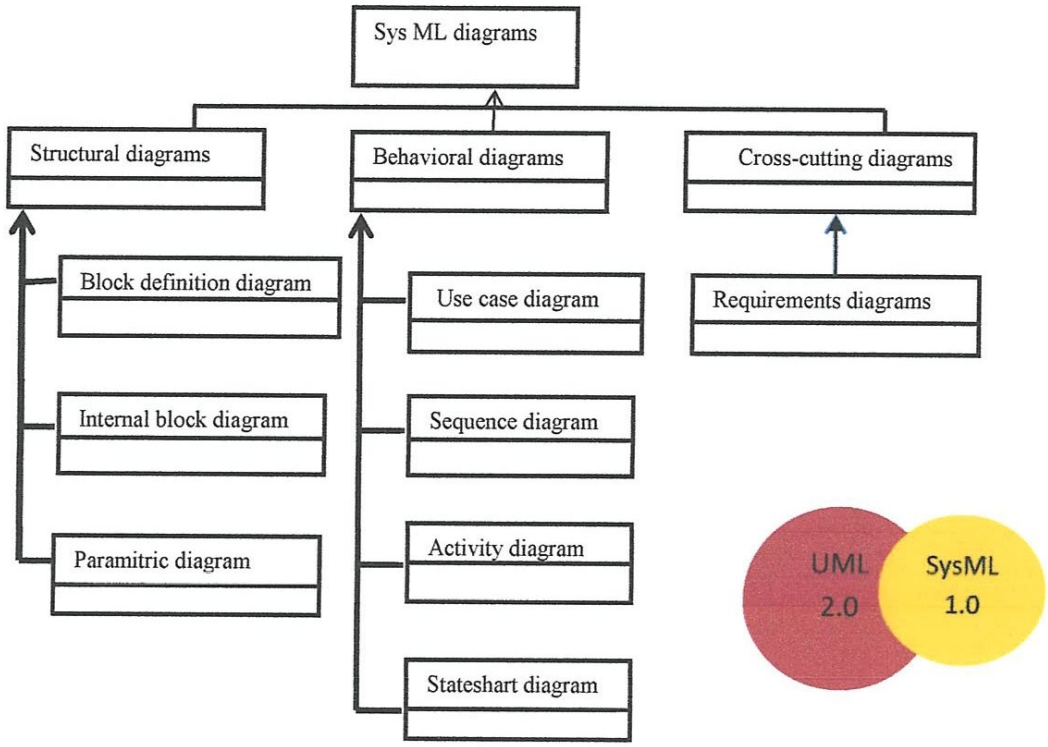
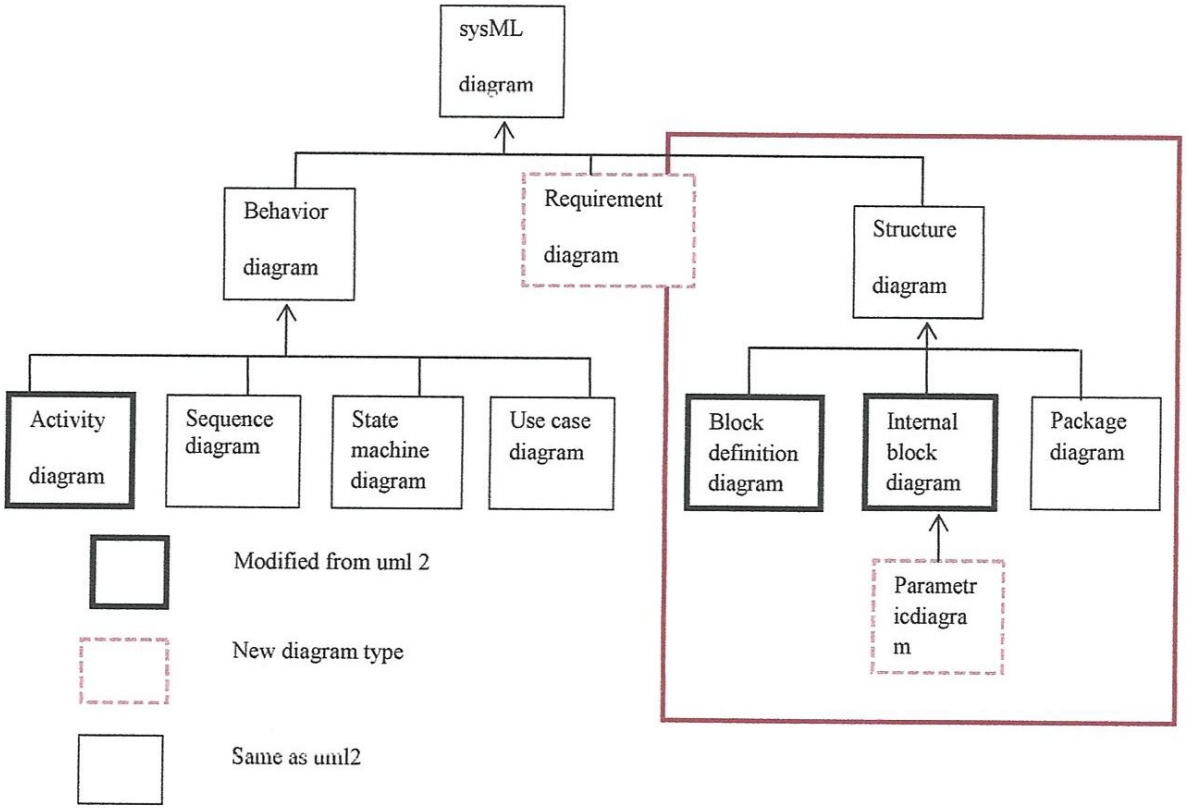


Figure 2.4 : *Représentation de l'organisation des diagrammes SysML comparée à UML*



IV.1.6 Les diagrammes SysML

En SysML chaque diagramme est nommé d'une façon bien précise et constitue un élément nommé du modèle. Pour cela SysML définit un en-tête standard à chaque diagramme qui contient obligatoirement :

- Le type de diagramme : act, bdd, ibd, sd, etc....
- Les éléments représentés dans le diagramme : packages, blocs, activités, etc....
- Le nom de l'élément modélisé, le nom du diagramme ou de la vue représentée.

IV.1.7 Différences entre UML et SysML

La principale différence réside dans le fait que SysML utilise des « Block » quand UML utilise des « Class ».

Par exemple nous pourrons modeliser le système ABS ci-dessous qui contient un bloc de contrôle « Système anti blocage des roues » lui-même composé d'un bloc de détection d'adhérence et d'un bloc de régulation de freinage.

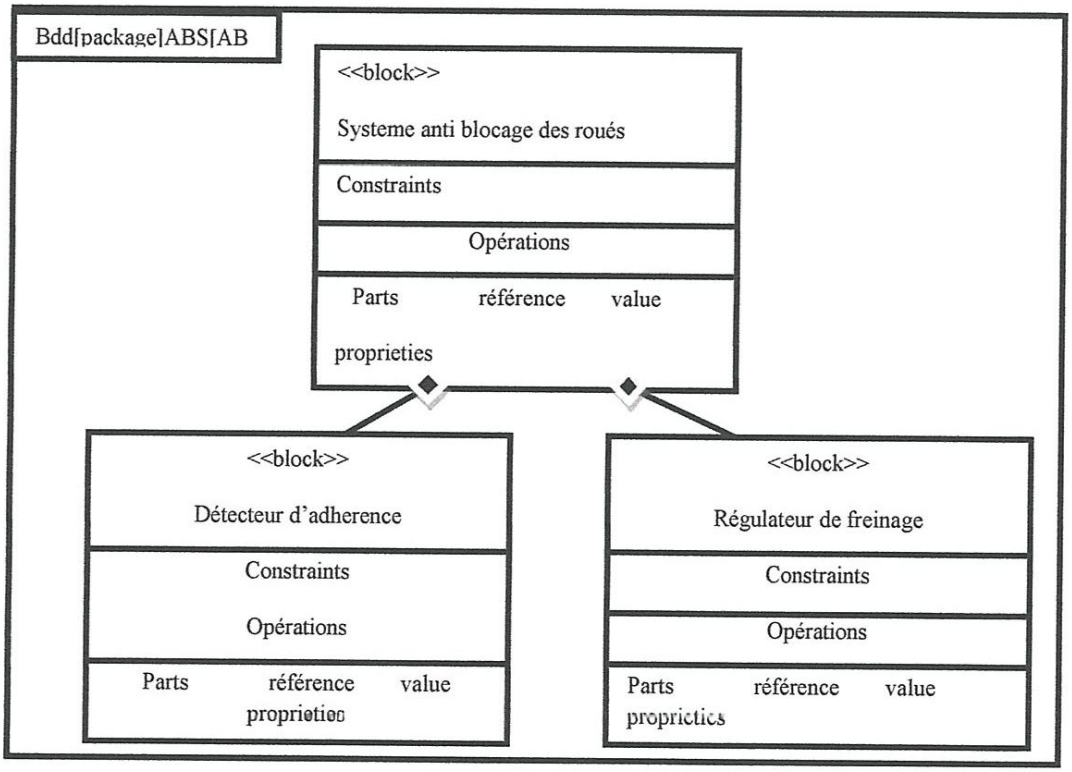


Figure 2.5 : *Exemple de modélisation du system ABS*

On retrouve bien dans cette représentation les principes d'UML mais adapté à un système qui combine de la mécanique, de la physique, de l'électronique et de la programmation.

La sémantique que nous allons donner aux diagrammes va ainsi évoluer puisque nous ne serons plus dans un contexte strictement informatique.

IV.1.8 Discussion sur SYSML

Vu les particularités des systèmes embarqués et des Socs, il existe des fortes similitudes entre les méthodes utilisées dans le domaine de l'ingénierie des systèmes et la conception de Socs complexes, telles que la gestion précise des exigences, la spécification de systèmes hétérogènes, la simulation, la vérification et la validation.

L'une des principales contributions de SysML dans le domaine de SOC est le support pour modéliser les exigences. Les principales limites de SysML sont surtout au niveau du raffinement du système vers des implémentations logiciels/matériel.

D'autre part, SysML ne résout pas le problème de l'absence de sémantique dans UML2.0 et ne dicte aucun processus de développement à être utilisé. Afin d'être en mesure d'intégrer les modèles d'exigences SysML dans les flots de conception des SOC, la formalisation de telles annotations informelles est indispensable. [9]

IV.2 SPT

En génie logiciel les spécialistes de performance, et les contributeurs ont la définition de la performance des sous profils du profil UML SPT.

Notre point de vue est que le profil doit être utilisable dans la pratique quotidienne par les développeurs de logiciels.

IV.2.1 Principes Généraux de Profil UML SPT

- Ne pas modifier UML sauf le cas inévitable
- Ne pas limiter la manière d'utilisation d'UML
- Fournir la capacité d'annoter les modèles UML en vue d'analyse [quantitative]
- Permettre aux utilisateurs de profiter de différentes techniques d'analyse sans avoir besoin d'une connaissance profonde

• Supporter toutes les technologies temps-réel, toutes les techniques d'analyse de modèle, toutes les paradigmes de conception

• Faciliter la construction automatique/systématique de modèles d'analyse à partir de modèles UML

• Fournir un Framework unifié qui capture les éléments communs de toutes les méthodes d'analyse modèle

• Supporter à la fois l'analyse et la synthèse de modèles [14]

IV.2.2 Le Point de Vue Fondamental du Profil SPT

Dans le profil SPT, un scénario est l'unité d'exploitation pour lesquels les spécifications de performance et de prédictions doivent être donnés, la durée du scénario définit ce qu'est un ingénieur delà performance pourrait appeler un de réponse. Spécifications de performance sont les étiquettes attachées son stéréotype étape, qui peut être un scénario, ou une étape en son sein. Paramètres d'intensité de charge de travail, et des demandes d'utilisation des ressources, qui sont utilisés dans la création prédictive modèles, et aussi attaché à étapes.

Scénarios utilisent les services des entités de ressources, qui ont des paramètres tels que la politique de service, la multiplicité et de temps de fonctionnement, et des mesures telles que l'utilisation.

Analyse de performance s'applique à des cas plutôt que des classes. Des instances d'objets sont déployées, et exécuter. Différentes instances de la même classe peuvent avoir des comportements différents en fonction de leur rôle, ou sur le des données qu'ils traitent, et la même instance d'objet peut avoir un comportement différent dans différents scénarios.

IV.2.3 Capacités et limites

Nous avons trouvé simple d'annoter une description monolithique d'une réponse donnée dans un seul diagramme de séquence ou d'activité. Des limitations peuvent survenir en raison delà sémantique du diagramme UML, par exemple diagrammes de séquence sont faibles dans la définition alternative et parallèle ramifications du scénario. Schémas de comportement en UML2 semblent mieux définir les scénarios. [26]

IV.3 UML-SOC

Il est développé par les laboratoires Fujitsu Limited et Fujitsu. Une soumission OMG a été préparée par un consortium constitué de Fujitsu Limited, IBM Corporation, CANON INC, CATS Co., Ltd métaboliques, RICOH COMPANY LTD., et Toshiba Corporation.

Ce profil vise à décrire les informations spécifiques aux Socs en utilisant UML. Il intègre les concepts de SOC et permet la génération automatique de code pour le matériel (par exemple : SystemC), couvrant les niveaux d'abstraction à partir de la modélisation de niveau transactionnel (TLM) au niveau transfert de registre (RTL). UML-SOC est appuyé sur le diagramme de structure d'UML2.0.

Il propose les stéréotypes qui permettent la modélisation structurelle, modélisation de la communication, et la modélisation des opérations et des propriétés. Le tableau 2.3 donne la correspondance entre certains stéréotypes de Socs et les constructeurs UML.

Dans cette approche, UML est utilisé comme un modèle formel pour la spécification de Socs afin de permettre la validation de la cohérence et l'exhaustivité de spécification.

Soc Model element	Stéreotype	UML métaclass
Module	**SoCModule**	**Class**
Process	**SoCProcess**	**Operation**
Data	**Data**	**Class**
Controller	**Controller**	**Class**
Protocol interface	**SoCInterface**	**Interface**
Channel	**SoCChannel**	**Class**
Protocol	**SoCprotocol**	**Collaboration**
Port	**SoCInterface**	**Port /Class**
Module part	**SoCModuleproperty**	**Property**
Channel part	**SoCChannelProperty**	**Property**
Connector	**SoCConnector**	**Connector**

Tableau 2.3 : *Un sous-ensemble des stéréotypes de profil UML-SOC*

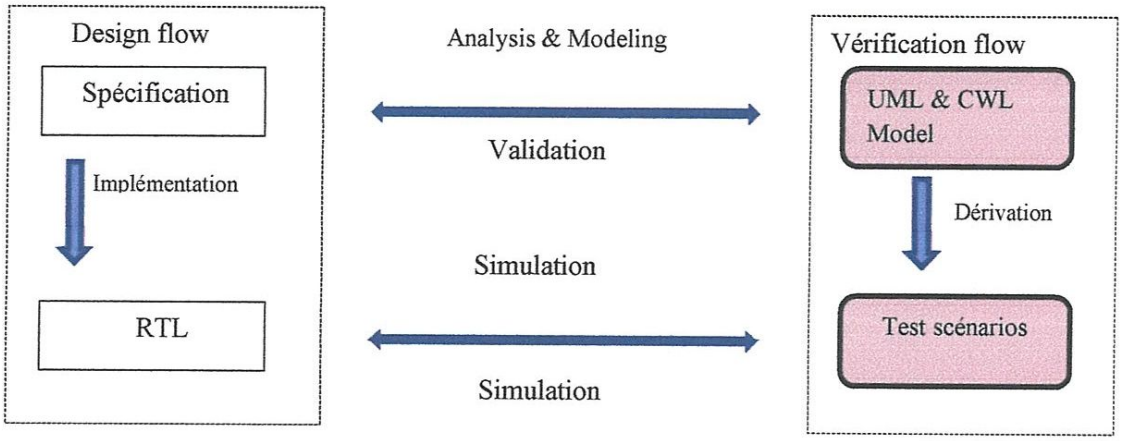


Figure 2.6 : *Le flot de profil UML- SOC*

IV.3.1 Discussion sur UML-Soc

Le profil UML-SOC peut être considéré comme une extension de processus conventionnel de conception de SOC.

Il ne traite que des aspects limités de développement de systèmes intégrés à savoir la formalisation des spécifications, et la dérivation de scénarios de test. Vu les aspects particuliers de systèmes embarqués complexes et les SOC, les principales limites de ce profil sont:

- Les aspects liés aux propriétés non fonctionnelles, ne sont pas abordés dans la spécification UML au niveau système.
- Le raffinement des interfaces est basé sur un langage propriétaire CWL entraînant un manque d'interopérabilité entre outils.
- La mise en œuvre est décrite au niveau RTL séparément, mais la vérification fonctionnelle utilise les mêmes scénarios de tests qu'au niveau UML.
- Certaines sémantiques des stéréotypes sont définies informellement (par exemple, le stéréotype protocole), d'autres nécessitent encore quelques précisions (sémantique de Synchronicité). [9]

IV.4 UML-SystemC

Ce profil est élaboré par l'université de Catane et STMicroelectronics. Il tire profit des avantages de deux standards : UML2.0 et SystemC en suivant les principes du MDA.

SystemC est bien adapté pour l'implémentation des modèles UML, car il supporte le paradigme orienté-objet et peut présenter de manière uniforme le matériel et le logiciel dans un seul langage.

En outre, SystemC est en train de devenir le langage standard pour la conception des Socs au niveau système. Selon, UML peut améliorer le flot de conception de Socs de trois façons:

1. UML peut être adopté au niveau système en tant qu'un langage fonctionnel exécutable pour décrire la spécification.

2. Le profil UML pour SystemC peut être utilisé pour la description du matériel aux niveaux abstraits au-delà de niveau RTL.

3. Les profils UML adaptés pour les langages de programmation comme C / C + +, Java,etc., peuvent être utilisés pour la partie logiciel.

Le profil UML-SystemC capture à la fois les caractéristiques structurelles et comportementales du langage SystemC et permet la modélisation de haut niveau de Socs avec une simple translation vers le code SystemC.

IV.4.1 Objectifs

- Environnement unifié de description matériel et logiciel.
- Explorer les solutions de conception en travaillant à un niveau système

=> Améliorer la productivité des concepteurs de systèmes électroniques

IV.4.2 Caractéristiques du langage SystemC

1. Spécification et conception à différents niveaux d'abstraction.
2. Intégration de portions de logiciel embarqué, à la fois sous la forme de modèles et de code : réutilisation.
3. Création de spécification exécutable.
4. Création de plateformes exécutables sur les quelles seront mappées les spécifications.
5. Simulations rapides pour permettre l'exploration de l'espace de conception (spécifications + plateformes)
6. Structuration des modèles autorisant la séparation de la fonctionnalité et de la communication

Adaptation flexible (IPs)

Besoin de réutilisation

IV.4.3 Les Structures SystemC

Les modules sont des SC_MODULE qui sont des classes. Les processus sont implémentés Sous forme de Co-routines. Il existe différents types de Co-routines en SystemC

Les SC_METHOD et les SC_THREAD (les SC_CTHREAD étant un cas particulier de SC_THREAD).

IV.4.4 Les Limites D'UML-SystemC

- Il ne traite, ni la capture des besoins, ni les propriétés non-fonctionnelles.
- Il ne prend pas en considération ni la partie logiciel, ni la génération des interfaces hardware / software.
- Le manque d'une sémantique claire conduisant à la traduction de la totalité du code SystemC à UML, donc plusieurs pages sont nécessaires pour capturer une simple fonction. [8]

IV.5 MARTE (Modeling and Analysis of Real Time Embedded Systems)

Il est défini par le Groupe de travail Pro MARTE et est voté à l'OMG pour le développement et l'analyse dirigée par les modèles des systèmes embarqués temps réel.

MARTE vise à remplacer le profil UML SPT. Il est basé sur le méta-modèle UML2.0,OCL2 et MOF 2.0 QVT. Comme illustrée en figure 2.8 l'architecture MARTE est fondée sur quatre paquetages: le paquetage de base, le modèle de conception, le modèle d'analyse, et les annexes.

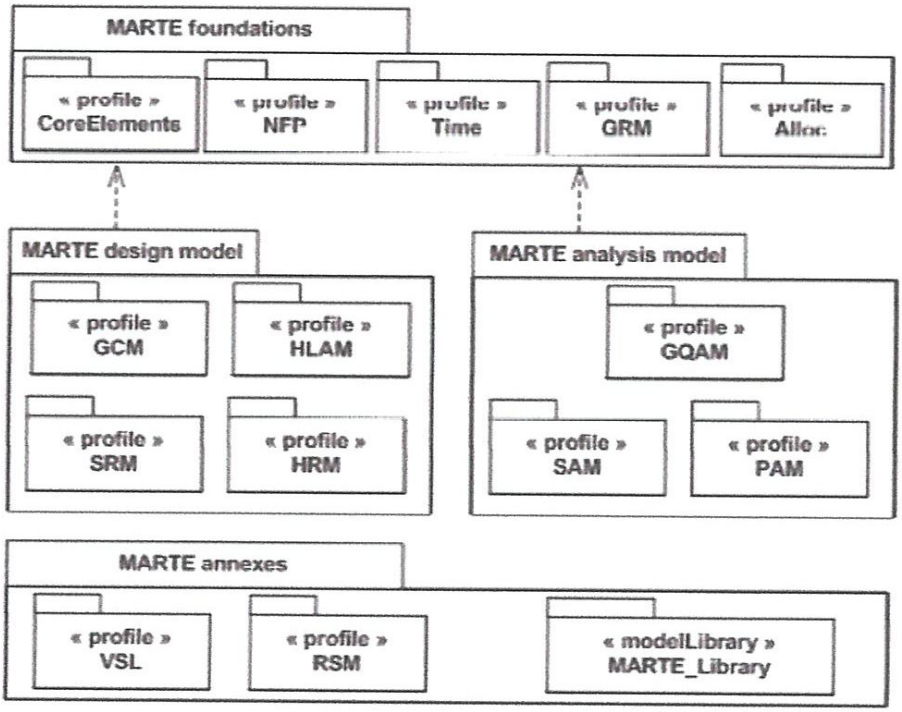


Figure 2.7 : *Architecture du profil MARTE*

- Le paquetage de base comprend le profil des NFPs pour la modélisation des propriétés non fonctionnelles. Une NFP, peut être soit élémentaire ou complexe, qualitative ou quantitative.
- Le profil GRM pour la modélisation générique de ressources, il est détaillé par le biais de DRM pour la modélisation détaillée de ressources.
- le profil GCM pour la modélisation de composants génériques et le profil ALLOC pour modéliser l'association matériel/logiciel.
- le profil SRM pour la modélisation de ressources logiciel et le profil HRM pour la modélisation des ressources matériel.
- Le paquetage d'analyse de MARTE introduit les éléments communs qui peuvent être utilisés pour contribuer à des nombreux types d'analyse quantitative. Trois types d'analyse sont pris en compte.
- La SAM pour l'analyse d'ordonnançabilité, la PAM pour l'analyse des performances et de temps d'exécution au pire des cas.
- Le sous-profil VSL (Value Specification Language) qui est un langage d'expression, utilisé pour spécifier les valeurs non-fonctionnelles. En fin, le sous profil RSM pour modéliser les structures répétitives.

MARTE apporte de nombreux avantages car il fournit un support pour la spécification, l'analyse, la conception, la vérification, et la validation, fournit un moyen de modélisation pour les aspects matériels et logiciels de systèmes embarqués temps réel en vue d'améliorer la communication entre les développeurs, et favorise la construction de modèles qui peuvent être utilisés pour faire des prédictions quantitatives.

IV.5.1 Discussion sur marte

MARTE vise principalement les systèmes embarqués temps réel. Ce profil offre une facilité de modélisation et d'analyse des applications temps réel, toutefois, dans le contexte de Co-conception, où les développements du matériel et de logiciel ont souvent lieu simultanément, le profil est moins utile: les problèmes liés au matériel, comme l'exploration de l'espace de conception, la synthèse, la génération d'interfaces matériel-logiciel ne sont pas suffisamment adressés.

Il souffre également d'un manque de modélisation des exigences, de l'analyse et une discussion plus profonde à propos de l'abstraction et de la hiérarchie de l'application et de la plate-forme matérielle serait nécessaire. [8]

V. Comparaison entre les profiles

	SYSML	UML-SOC	UML-SYSTEMC	MARTE
NC	**N**	**N**	**N**	**O**
RC	**O**	**N**	**N**	**N**
PA	**N**	**N**	**N**	**O**
HS	**N**	**O**	**O**	**N**
HSI	**N**	**N**	**N**	**N**
IPR	**N**	**O**	**O**	**N**
FA	**N**	**N**	**N**	**N**
PAR	**COM**	**PROC**	**PROC**	**COM PROC**
TD	**SYS**	**SOC**	**SOC**	**ERS**
AF	**?**	**MDA**	**MDA**	**?**

Tableau 2.4 : *Profils UML pour SOCS et systèmes embarqués*

NC: Capture de besoins non fonctionnels. **RC**: Capture de besoins fonctionnels. **PA**: Analyse de Performance. **HS**: Synthèse Matériel. **HSI**: Synthèse Interface Matériel/Logiciel. **IPR**: Réutilisation et Intégration des IPs. **FA**: Analyse Formelle. **PAR**: Paradigme. **TD**: Domaine cible. **AF**: Flot associé (Méthodologie). **COM**: Composant. **PROC**: Processus. **SR**: Réactive Synchrone. **ISP**: Traitement du signal intensif. **SOC**: Système mono puce. **ERS**: Systèmes embarqués Temps Réel. **ES** : Systèmes Embarqués. **WCP** : Protocoles de communication sans fil.

PBD: Conception basée plateforme. **MDA**: Architecture dirigée par les modèles. **MDD**: Développement dirigé par les modèles.

VI. Conclusion

Dans ce chapitre, nous avons présenté les profils UML les plus répandus dans le domaine de SOC et de systèmes embarqués. D'après l'analyse que nous avons faite, nous pouvons tirer les points suivants:

Depuis, que la plupart des profils UML s'appuient sur le paradigme processus (tâche), il leur manque les capacités du paradigme orienté objet. Bien que le paradigme processus soit plus approprié pour la synthèse, le partitionnement matériel/logiciel et l'analyse de performance, il manque des possibilités de réutilisation et de l'abstraction.

Selon nos connaissances, peu de travaux visent la synthèse de matériel à partir de spécifications purement orienté-objet. Les travaux de, ciblent la génération des architectures reconfigurables à partir de spécifications objets en exploitant les principes de paradigme objet comme le polymorphisme, l'encapsulation et l'héritage.

La plupart des profils manquent de support formel pour l'analyse, le raffinement et la validation.

Chapitre 3
EMF

Chapitre3 : EMF

I. Introduction

EMF est un projet de la communauté open source Eclipse. C'est un Framework de modélisation et de génération de code pour la construction des outils et d'autres applications basés sur une structure de modèle de donnée. Les modèles peuvent être spécifiés en utilisant des documents Java,UML,XML, puis sont importés dans EMF.

A partir d'une spécification du modèle décrit dans déférente formats, EMF fournit des outils et support d'exécution pour produire un ensemble de classes Java pour le modèle, un ensemble de classes d'adaptateur qui permet la visualisation et la commande basée sur l'édition du modèle.

Le projet EMF permet de créer deux types de modèles, d'un côté des modèles définissant des concepts, souvent nommé le méta-modèle, et de l'autre des modèles instanciant ces concepts. A titre d'exemple, on peut définir un méta-modèle définissant des concepts tel que "Classe", "Opération" et "Attribut" et ensuite utiliser ce méta-modèle pour définir les modèles contenant.

EMF permet non seulement de créer un méta-modèle représentant les concepts désirés par l'utilisateur mais il permet à l'utilisateur de créer des modèles issus de ce méta-modèle et de les manipuler avec un outillage adapté.

EMF propose son propre méta méta modèle, le méta méta modèle Ecore, Ecore est entièrement intégré à la plate-forme Eclipse.

EMF permet de traiter différents types de fichiers : conformes à des standards reconnus (XML, XMI) et aussi sous des formes spécifiques (code Java).EMF ne propose pas d'outil graphique (de dessin) pour la modélisation.[20]

II. L'objectif d'EMF

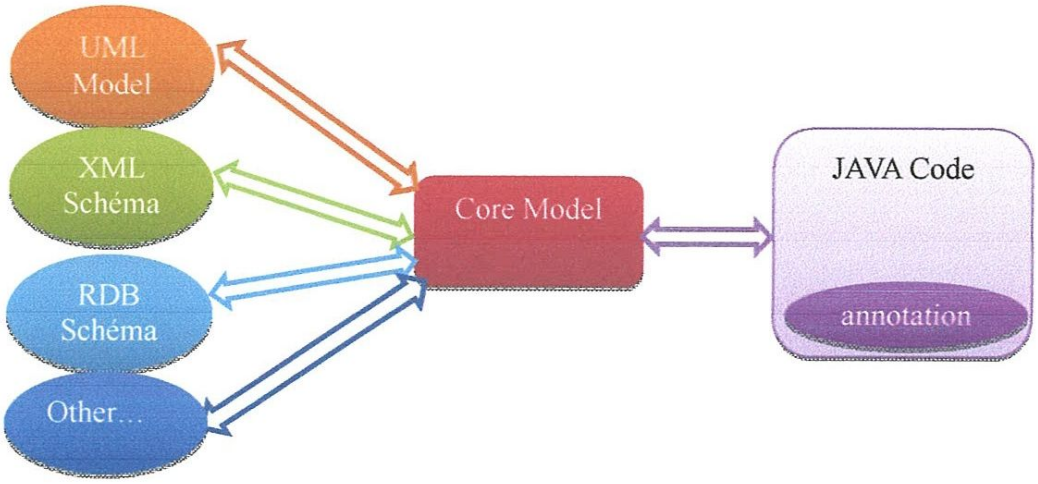


Figure 3.1 : *Organisation générale d'EMF*

L'objectif général d'EMF est de proposer un outillage qui permet de passer du modèle au code *Java* automatiquement. Pour cela le *Framework* 'articule autour d'un modèle (le *Core Model*).

EMF propose plusieurs services :

1. la transformation des modèles d'entrées, présentés sous diverse formes, en *Core Model*.

2. la gestion de la persistance du *Core Model*.

3. la transformation du *Core Model* en code Java.

Les doubles flèches symbolisent que les transformations inverses (ou les « imports/exports ») sont aussi gérées par EMF.

En cela, certains considèrent que EMF est à la fois :

• un outil de réunification de divers standards de modèles (XMI, UML, code Java - si du moins on considère ce dernier comme un modèle),

• un pont entre deux mondes du génie logiciel (celui des gourous de la modélisation et celui des partisans du code avant tout) en prenant une position médiane et en prenant le meilleur de chaque monde. [21]

III. Les Formats d'entrés Standard

EMF peut de base gérer en entrée des modèles présentés sous trois formats : UML, XMI et en code Java Annoté

III.1 XML

Pour cette option, il existe trois possibilités.

L'édition directe conformément au méta-modèle Ecore :

Il s'agit là, d'éditer des modèles graphiques UML conformément au méta-modèle *Ecore*. Cela sous-entend l'utilisation d'un outil de modélisation qui en soit capable. Par voie de conséquence, la transformation d'entrée du modèle n'est plus : on dispose du *Core Model* sous le bon format.

L'importation de modèles UML :

Il s'agit d'importer, à l'aide de la fonction ad hoc d'EMF, un modèle dans son format natif.

L'exportation de modèles UML :

C'est à peu près le même principe que l'option d'importation, sauf que la conversion du format natif en Ecore ne se fait pas avec EMF, mais avec l'outil de modélisation d'origine.

III.2 XMI

XMI se charge de formater ces contenus pour permettre de leur assurer une persistance standardisée.

III.3 J ava Annoté

Une des solutions tentantes pour modéliser les classes, qui vont être concrétisées par une application Java, est d'utiliser les interfaces Java :

• elles n'implémentent pas les méthodes : on s'abstrait donc de cette implémentation.

• les méthodes *get/set* peuvent être utilisées pour modéliser les attributs.

• une classe pourra implémenter plusieurs interfaces, ce qui est une manière détournée d'autoriser l'héritage multiple.

Les annotations sont des *tags model* placés dans la *java doc* des interfaces. Ces *tags*, et leurs attributs éventuels, sont détectés par EMF qui considère ainsi les entités concernées (interfaces, méthodes) comme des éléments de modélisation.

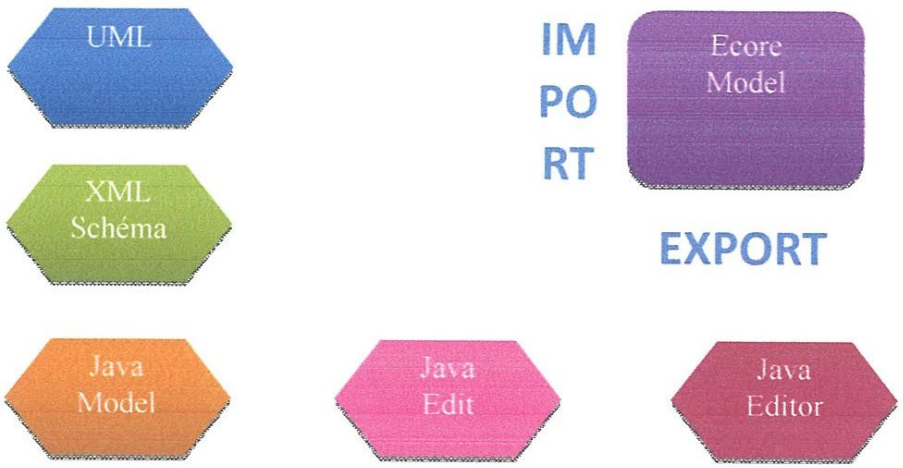


Figure 3.2 : *Vue générale d'EMF*

IV. Les étapes du parcours d'EMF

- Création d'un méta-modèle (extension *.ecore*).
- Création du genmodel associé (extension *.genmodel*).
- Génération du plugin d'édition et des classes java de parcours du modèle.
- Edition d'un modèle (conforme au méta-modèle).
- Parcours et modification de modèle par programme Java.

V. Avantages d'utiliser EMF

- ✓ EMF permet d'offrir une visibilité claire du modèle.
- ✓ EMF fournit également des fonctionnalités de notification de modification du modèle, en cas de modifications apportées au modèle.
- ✓ EMF va générer des interfaces pour créer vos objets, donc il vous aide à garder votre application propre à partir de l'implémentation individuelle des classes.
- ✓ Un autre avantage est que vous pouvez régénérer le code Java à partir du modèle à n'importe quel point dans le temps.

VI. L'architecture à quatre niveaux

Cette architecture est hiérarchisée en quatre niveaux. En partant du bas :

- ✓ Le niveau M0 (ou instance) correspond au monde réel. Ce sont les informations réelles de l'utilisateur, instance du modèle de M1.
- ✓ Le niveau M1 (ou modèle) est composé de modèles d'information. Il décrit les informations de M0. Les modèles M1 sont des instances de méta-modèle de M2.
- ✓ Le niveau M2 (ou méta-modèle), il définit le langage de modélisation et la grammaire de représentation des modèles M1. Le méta-modèle UML qui est décrit dans le standard UML, et qui définit la structure interne des modèles UML, fait partie de ce niveau. Les profils UML, qui étendent le méta-modèle UML, appartiennent aussi à ce niveau. Les méta-modèles sont des instances du MOF.
- ✓ Le niveau M3 (ou méta-méta-modèle) est composé d'une unique entité qui s'appelle le MOF. Le MOF permet de décrire la structure des méta-modèles, d'étendre ou de modifier les méta-modèles existants.[37]

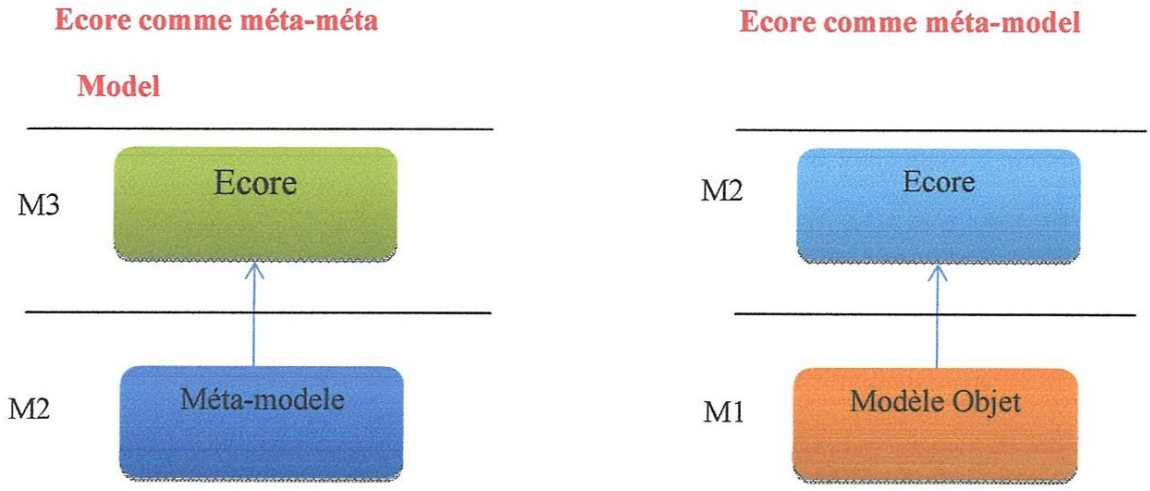


Figure 3.3 : *Deux niveaux de modélisation d'Ecore dans EMF*

VII. Génération du code

EMF répond bien à son objectif d'améliorer la productivité du développement d'application. Il y arrive en automatisant la génération du code à partir du modèle.

VII.1 Organisation du code généré

Un choix de conception a été fait par les concepteurs et est imposé par EMF : la séparation interface/implémentation dans le code généré.

Cela va se concrétiser par la génération de deux ensembles :

1. un ensemble d'interfaces Java,
2. un ensemble de classes implémentant ces interfaces.

Des classes correspondant au modèle d'entrée, EMF génère deux autres éléments importants : une interface *Factory* et une interface *Package* ainsi que leurs classes d'implémentation.

- ***La Factory***

Cette interfacc comprend une méthode *create* pour chacune des classes du modèle d'entrée. Cela va permettre de créer des instances (des objets) des classes de l'application.

Le modèle de programmation EMF incite fortement à utiliser ces méthodes pour créer les objets lors de l'utilisation de l'application, en lieu et place de l'opérateur *new*.

- ***Le Packetage***

Cette classe apporte des facilités pour accéder aux métadonnées Ecore du modèle. Il contient des accesseurs aux *EClasses*, EA*ttributes*et *EReferences* implémentées dans le

modèle (par exemple).

VII.2 La régénération et la fusion

Une des caractéristiques avantageuses d'EMF est qu'il va permettre de compléter manuellement le code obtenu automatiquement, de pouvoir régénérer le code à partir du modèle modifié sans perdre les ajouts faits manuellement.

En effet, il est indispensable de pourvoir ajouter des méthodes et des attributs au code généré automatiquement. Celui-ci s'occupe uniquement de générer le squelette des classes, les références aux autres classes (les associations) et les accesseurs aux attributs (les *get* et les *sets*).

VII.3 Le modèle générateur

En plus du modèle conforme au méta-méta-modèle Ecore, EMF utilise un modèle dit générateur (fichier d'extension *.genmodel*). Ce modèle, comme le modèle Ecore, est généré automatiquement lors de la transformation du modèle d'entrée en modèle Ecore.

La plupart des informations nécessaires sont contenues dans le modèle « core » : le nom des classes les attributs, les références,...

Mais un certain nombre d'informations n'y sont pas telles que : les règles de préfixation du nom des classes, où mettre le code généré.

VIII. Profil de Soc

Le terme System-on-Chip« système sur puce » (Soc) désigne l'intégration des composants d'un calcul ou un système de communication dans une seule puce. Les composants peuvent être des signaux numériques, analogiques.

Le système de conception Soc est très connu à base de composants de conception en génie logiciel. La motivation principale de Soc est la compacité et réutilisation de la conception.

System-on-Chip (Soc) de paradigme permet aux composants de fonctions multiples, composées à la fois hardware et software pour être intégré sur une seule puce de silicium.

Le paradigme soc permet la conception du système à base de composants. Bien que soc paradigme partage les mêmes principes de conception avec le développement de logiciels à base de composants, conception de soc diffère de génie logiciel en raison de la présence à la fois synchrone (cadencé) et les composants matériels asynchrones et des interfaces matériel-logiciel.

L'existants sémantique UML a un fort accent sur la spécification des exigences du logiciel et est donc insuffisante pour soutenir la modélisation et la spécification des conceptions soc.

Afin de soutenir la modélisation et la spécification des conceptions soc, nous proposons un minimum d'ensemble d'extensions pour un profil UML.

Pour augmenter la sémantique pour UML basé sur soc la conception, les stéréotypes pour chaque méta classe UML doivent être définis par l'introduction d'une mentions spéciale et la contrainte pour chaque élément soc. [32]

VIII.1 Les stéréotypes de profil Soc ML

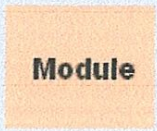
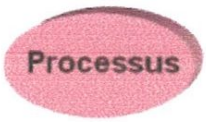
Chaque élément du modèle soc correspond au méta classes UML stéréotypés comme présentés dans le tableau ci-dessous. Contraintes définies par ces stéréotypes sont décrits dans documents d'accompagnement intitulé contraintes profil soc définies par OCL, le profil soc les stéréotypes, et les relations soc stéréotypes profil (diagramme de classes).[32]

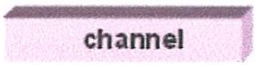
Section no	SoC Model element	Stereotype	Méta class
1	Module	SoCModule	Class
2	Process	SoCProcess	Opération
3	Data	Data	Class
4	Controller	Controller	Class
5	Protocol Interface	SoCInterface	Interface
6	Channel	SoCChannel	Class
7	Protocol	SoCProtocol	Collaboration
8	Port	SoCPort	Port/Class
9	Module Part	SoCModuleProperty	Property
10	Channel Part	SoCChannelProperty	Property
11	Connector	SoCConnector	Connector
12	Clock Port	SoCClock	Port
13	Reset Port	SoCReset	Port
14	Clock Channel	SoCClockChannel	Property

15	Reset Channel	SoCRestChannel	Property
16	Data Type	SoCDataType	Independency

Tableau 3.1 : *Méta classes UML étendu par Soc profile.*

IX. La validation du Modèle EMF

Elément du modèle SoC	Définition	Les contraintes
1. Module 	- Le module est la classe de base pour décrire la structure hiérarchique de classe SoC.	-chaque Module avoir un nom valide (non vide). - deux Modules n'ont pas le même nom.
2. Process 	- Le process est une fonction membre du module qui décrit son comportement.	- Il y a restriction sur le nombre de processus qui peuvent accéder à un canal ou d'un port en fonction de l'interface de protocole réalisant le canal. -un processus ***doit*** passer par un port pour accéder aux canaux qui lui sont extérieurs.
3. Data	- Seule la classe de données serait la cible de communication entre les modules utilisant la classe du canal.	
4. Controlleur	- La seule fonction membre de cette classe pourrait être le processus de module qui est	

	décrit plus loin.	
5. Protocol Interface 	- Il définit la communication entre les modules, et offre une abstraction pour la méthode de communication	-Tous communication entre les modules doit être réalisé à travers des interfaces de protocole. - L'interface doit être définie par le type de données.
6. Channel 	- un Channel est un moyen de communication entre les modules à travers les interfaces de protocole. - Il peut aussi contenir d'autres canaux.	- un Channel lit deux ports de deux modèles différents.
7. Protocol	- Protocol décrit la relation entre les interfaces de protocole et le canal.	- Chaque protocole doit avoir au moins une interface de protocole entrant, et au moins une interface de protocole sortant, et au moins un canal hérité directement ou indirectement à partir d'interfaces de protocole.
8. Port 	- Il définit l'interface de protocole utilisé pour la communication externe aux	- un port ne peut pas être connecté à rien - Un port d'un module est

	modules.	connecté à un Channel (sauf dans le cas particulier où il est directement connecté au port d'un module parent) -identifie par un nom unique
9. Module Part	- Il s'agit d'un type de propriété tapé par la classe du module. Il est généralement appelé " sub-module ".	
10. Channel Part	- Il s'agit d'un type de propriété tapé par la classe du canal. Typiquement, le canal doit être montré que cette propriété, et ne pas exister indépendamment.	
11. Connector	- Connecteur relie les ports ou relie les ports et un canal.	-identifie par un nom unique
12. Clock Port	- Cette classe est héritée du port, et il précise d'entrée d'horloge ou d'un port de sortie d'horloge.	
13. Reset Port	- Cette classe est héritée du port, et il précise d'entrée de réinitialisation ou reset port de sortie.	
14. Clock Channel	- Il s'agit d'une propriété définie l'instance de canal (s) qui	

	transmet les événements d'exécution des processus synchrones	
15. Reset Channel	- Il s'agit d'une propriété définie instance de canal (s) qui transmet les événements de réinitialisation des processus. - Il est généralement utilisé dans ou après la "conception structurale", qui	-exige la prise en compte des régimes de remise à zéro.

Tableau 3.2 : *Contraintes de la validation du modèle EMF*

Les contraintes servent à exprimer la sémantique du profile. Ils permettent de préciser les conditions d'emploi des éléments du modèle.[32]

X. Java Emitter Template (JET)

L'Eclipse Modeling Framework (EMF) contient un outil très puissant pour générer du code source. JET (Java Emitter Template). JET est un moteur de Template générique qui peut être utilisé pour générer du XML, code source Java et autre sortie à partir de modèles.

JET est généralement utilisé dans la mise en œuvre d'un "générateur de code".

En JET vous définir des modèles. Ces modèles seront utilisés pour créer des classes de mise en œuvre de Java. Cette étape du procédé est appelée «traduction».

Les classes Java peuvent ensuite être utilisées pour créer la sortie finale, cette étape du procédé est appelée «génération».

JET a trois différents types d'expressions, les directives, les expressions et scriplets. Scriplets sont démarrés par <% et se termine par%> et peut contenir tout code java. Expressions permettent d'insérer des valeurs de chaîne dans la sortie JET et les directives définissent les paramètres pour le modèle JET.

Un modèle de JET est un fichier texte avec un nom de fichier qui se termine par "jet". Le compilateur JET crée un fichier source Java pour chaque JET.

X.1 Librairies de commandes

JET contient principalement quatre librairies de commandes à savoir :

- **Commandes de Contrôle**: Utilisées pour accéder au modèle d'entrée et pour contrôler l'exécution du Template.
- **Commandes de format**: Utilisées pour modifier le format du texte dans les modelés selon certaines règles.
- **Commandes Java** : Ce sont des balises utiles pour générer du code Java.
- **Commandes de Workspace**: Utilisées pour la création de ressources dans l'espace de travail, telles que des Chiers, des dossiers et des projets.

En utilisant les Template de JET, le développeur peut créer différentes implémentations pour un même modèle. Il peut ensuite, paramétrer l'implémentation en fonction du contexte. Deux étapes intermédiaires sont nécessaires pendant la génération du code :

- **La translation** : Consiste à transformer les Template en une implémentation de classes Java.

La génération proprement dite : L'utilisation de ces classes Java en vue de l'obtention du résultat désire.[29]

XI. Conclusion

Nous avons étudié dans ce chapitre :

- La construction d'un modèle EMF
- La génération du code à partir d'un modèle
- Le stéréotype du profile soc ML
- La validation du modèle EMF
- La génération du code à partir du moteur de Template jet

Chapitre 4
SystemC

Chapitre 4 : SystemC

I. Introduction

Après une étude détaillée que nous avons faite sur un environnement qui permet une grande variété d'activités de développement, éclipse. Il possède une plate-forme ouverte pour le développement d'applications et extensible grâce à un mécanisme de plug-ins.

Au cours de ce chapitre nous avons présenté la définition des langages de description d'architecture et de modélisation SystemC, Le langage SystemC a été proposé comme solution pour la modélisation des composants logiciels et matériels d'un système.[22]

II. Les langages de Description du Matériel (HDL)

II.1 Définition

Il est un langage informatique permettant la description d'un circuit électronique. Celui-ci peut décrire les fonctions réalisées par le circuit (description comportcmcntalc) ou lcs portes logiques utilisées par le circuit (description structurelle).

À la différence d'un langage de programmations logicielles, la syntaxe et la sémantique d'un HDL incluent des notations explicites pour exprimer le temps et le parallélisme qui sont les attributs principaux du matériel.

Ces langages nous permettent d'observer le fonctionnent d'un circuit électronique modélisé dans un langage de description grâce à la simulation afin de la vérifier avant de faire un circuit prototype.

Les langages de description sont nombreux : SystemC, VHDL, Verilog, SpeC. Nous allons focaliser sur SystemC qui est le langage que l'on va utiliser par la suite.

II.2 SystemC

SystemC est un langage basé sur le C++. Il est soutenu par le consortium OSCI (Open SystemC Initiative) leur but été pas commercial mais c'été plutôt de promouvoir SystemC comme un standard pour la conception des systèmes sur puce.

L'environnement de développement du SystemC est le même que celui du C++ car il est une bibliothèque de classe de C++ avec une bibliothèque supplémentaire pour la simulation du matériel.

SystemC est un langage orienté objet, La librairie SystemC ne peut être téléchargée que depuis le site de l'OSCI. [33]

III. Pourquoi utiliser SystemC

- Orienté objet: réutilisation plus facile des blocs, ajout des structures nécessaires plutôt que définir un nouveau langage.
- Fournit un langage commun simple et une base de modèles pour faire les systèmes et les conceptions IP.
- SystemC est complètement source-ouvert (protégée par un accord de licence de source-ouvert).
- Il Permet le suivant
    - Une simulation plus rapide.
    - Une Co-simulation de matériel/logiciel.
    - Exploration architecturale.

IV. Caractéristiques du langage SystemC

- Spécification et conception à différents niveaux d'abstraction
- Intégration de portions de logiciel embarqué, à la fois sous la forme de modèles et de code : réutilisation,
- Création de spécification exécutable.
- Création de plateformes exécutables sur lesquelles seront mappées les spécifications.
- Simulations rapides pour permettre l'exploration de l'espace de conception (spécifications + plateformes)
- Structuration des modèles autorisant la séparation de la fonctionnalité et de la communication

□Adaptation flexible (IPs)

Besoin de réutilisation. [33]

V. Organisation de SystemC

L'architecture générale de l'environnement SystemC est organisée on plusieurs couche qui sont représentées dans la figure 4.1:

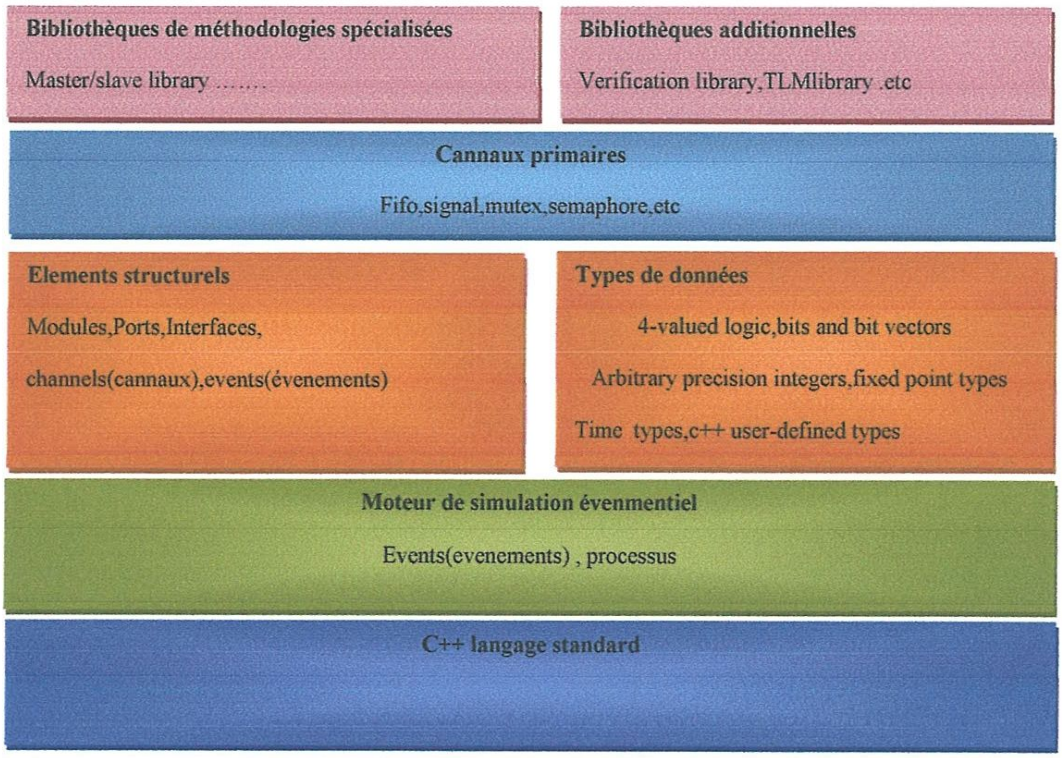


Figure 4.1 : Organisation de SystemC.

- Un modèle écrit en SystemC est donc un programme C++ utilisant les classes de la bibliothèque SystemC. Où C++ est un langage de programmation de haut niveau standardisé depuis 1998.
- les éléments structurels qui représentent la structure des systèmes matériels (modules, ports).
- Sachant que SystemC est une bibliothèque C++ qui elle comprend aussi en standard un moteur de simulation événementiel rapide, permettant de se passer d'un simulateur externe.
- Type de données : SystemC offre des type natifs (booléens, entier, réel) et les types définit par l'utilisateur (structure et classe) ainsi des types propre aux langages de description de matériel tels que les type pour la représentation des signaux [08].

VI. Structure d'un modèle SystemC

Un système est une hiérarchie d'objets. Généralement se sont des modules imbriqués communiquent entre eux par des canaux. Ces différents composants seront étudiés en détail Dans le chapitre précédant. Mais voici d'ores un aperçu rapide de la façon dont est construit un modèle SystemC d'un système numérique.

VI.1 Modules

Un *module* est une instance de la classe SC-module, le module encapsule un ou plusieurs ports d'entrée/sortie, un ou plusieurs processus, des variables partagées entre les processus et des instances d'autres modules. La déclaration d'un module se fait :

- ✓ soit « struct transmit : sc_module { } » cette forme de déclaration ressemble une déclaration typique de C+ + (struct ou class),
- ✓ soit on utilisant la macro SC_MODULE qui rend le code plus lisible. [33]

VI.2 Ports

Un module possède un ou plusieurs *ports*. Les ports sont juste des points d'entrée ou de sortie, qui ne font rien de particulier. Mais ils doivent déclarer les fonctions qui seront utilisées pour communiquer à travers eux. La déclaration des fonctions qu'il va utiliser est appelée *interface*.

La déclaration d'un port se fait : sc_+ in/ out/inout <Type de Port>Nom de Port.

Quand un port est relié à un signal ou une fifo, il existe des types de ports spécialisés, plus simples à instancier :

Type du canal	Entrée	Sortie	Bidirectionnel
sc_signal, sc_buffer	sc_in	sc_out	sc_inout
sc_signal_rv	sc_in_rv	sc_out_rv	sc_inout_rv
sc_signal_resolved	sc_in_resolved	sc_out_resolved	sc_inout_resolved
sc_fifo	sc_fifo_in	sc_fifo_out	(pas de sens)

Tableau 4.1: *Les Ports spécialisés.*

VI.3 Interfaces

Une interface est un objet C++ contenant des déclarations de méthode qui pourront être utilisées à travers les ports d'un module. Elle ne contient pas de code, c'est seulement une *déclaration de fonctions*. [33]

VI.4 Canaux

- Les canaux sont les moyens de communication entre les modules.

- Alors que les ports et interfaces définissent quelles fonctions sont disponibles, le canal décrit L'implémentation de ces fonctions.

La déclaration d'un Channel se fait : Nom de Channel <Type > Nom de port connecter à ce Channel. [33]

VI.5 Le Constructeur

Comme toute classe C++, un module possède un constructeur d'où le rôle de ce constructeur est de faire toutes les initialisations dont l'objet a besoin :

- donner une valeur par défaut aux variables
- appeler quelques fonctions (méthodes) pour pré-calculer ou s'enregistrer auprès d'autres objets, ...

VI.6 Processus

Les processus en SystemC décrivent une fonctionnalité et ils ne doit pas être appelés directement, c'est le moteur de simulation SystemC qui se charge de l'appeler (le déclencher).

La déclaration d'un Process se fait : Type de Process Nom de Process + (). [33]

La figure suivante illustre l'architecture d'un modèle en SystemC:

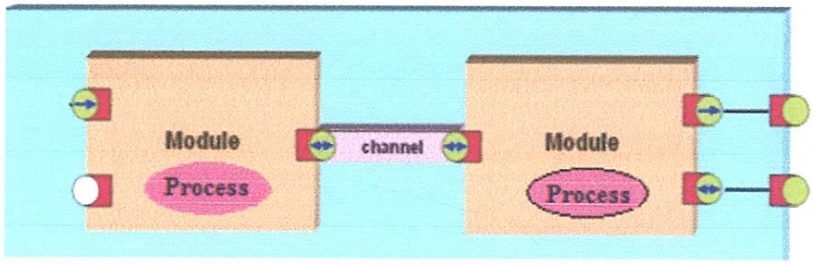


Figure 4.2: Organisation structurelle en SystemC.

VI.6.1 SC_METHOD

- ils sont *appelés par le scheduler* à chaque notification d'événement de leur liste de sensibilité
- ils s'exécutent en *entier*, dans le contexte du scheduler, puis exécutent un return(), qui *redonne la main* au scheduler.

Remarque importante: le scheduler SystemC est multithread. Mais les SC_METHOD ne sont *pas* des threads. Ce sont des fonctions normales, que le scheduler appelle les unes après les autres quand un événement de leur liste de sensibilité est notifié. Quand le SC_METHOD se termine (return()), il redonne la main au scheduler. Si, pour une raison ou pour une autre, le SC_METHOD ne retourne pas (s'il est bloqué, s'il est dans une boucle infinie), le scheduler est bloqué, la simulation n'avance plus !

- ***Création des SC_METHOD***

Comme tous les processus, leur création se fait en 4 étapes :

a. Déclaration

Un SC_METHOD est une méthode (fonction) du module qui ne prend pas d'argument et renvoie void. Il est déclaré comme toutes les autres méthodes.

Exemple :

```
SC_MODULE(mon_module) {
 // déclaration des ports, signaux, ...

// déclaration d'un processus (pour l'instant rien ne spécifie son type)
voidmon_sc_method();
 ....
}
```

Figure 4.7: La déclaration du processus SC_METHOD.

b. Enregistrement

C'est ici qu'on va indiquer au scheduler SystemC que cette fonction est un SC_METHOD, et qu'il faudra donc l'appeler quand ce sera nécessaire. On utilise pour cela la macro SC_METHOD, qui prend en argument le nom de la fonction.

L'enregistrement de la fonction auprès du scheduler se fait dans le *constructeur* du module.

Exemple:

```
SC_MODULE(mon_module) {
 // déclaration des ports, signaux, ...

 // déclaration d'un processus
voidmon_sc_method();

 // constructeur
 SC_CTOR(mon_module) {

// Enregistrement du processus comme un SC_METHOD
   SC_METHOD(mon_sc_method);

   // autres initialisations...
 }
...
}
```

Figure 4.8: L'enregistrement du processus SC_METHOD.

c. Déclaration de la liste de sensibilité par défaut

Il faut maintenant spécifier la liste de sensibilité du processus. Il pourra lui-même la modifier au coup par coup lors de la simulation s'il le désire. Cela se fait dans le *constructeur* du module.

Pour cela, on spécifie à quel *évènement* il est sensible à l'aide de la fonction sensitive. Cette fonction prend en argument un *évènement* et l'ajoute à la liste de sensibilité du *dernier processus enregistré*. Elle peut aussi prendre en argument un *signal*, et on extrait alors automatiquement *l'événement par défaut* (qui est notifié à chaque changement d'état du signal).

Il est possible de *filtrer les fronts montants* ou descendants, uniquement *pour les signaux sur 1 bit* bien évidemment (un bus n'a pas de "front montant"...).

Remarque:

- Les fonctions sensitive_pos et sensitive_neg ne sont plus supportées. Mais on peut les utiliser avec les SC_METHOD uniquement. Il vaut mieux les remplacer par: sensitive <<mon_signal.pos() et sensitive <<mon_signal.neg().

Si un processus doit être sensible à plusieurs évènement, on peut soit appeler plusieurs fois sensitive, soit profiter de la notation de flux << pour enchaîner les signaux.

Par défaut, les processus sont tous exécutés une fois au début de la simulation. Si on ne le souhaite pas, il suffit d'appeler la fonction dont_initialize () après la déclaration de la liste de sensibilité (toujours dans le constructeur).

Exemple :

```
SC_MODULE(mon_module) {

 // déclaration des ports, signaux, variables et d'une fifo
sc_in<bool>enable;
sc_fifo<int>fifo;
sc_event ev1;
sc_port<sc_signal_in_if<bool>> reset;

 // déclaration d'un processus
voidmon_sc_method();

 // constructeur

SC_CTOR(mon_module) {

  // Enregistrement du processus comme un SC_METHOD
  SC_METHOD(mon_sc_method);

// Déclaration de sa liste de sensibilité
  sensitive << ev1 <<enable<< reset;
  sensitive <<fifo.data_read_event();

  // Si on ne veut pas l'exécuter au début de la simulation
dont_initialize();

  // Enregistrement des autres processus
  ...
 }
...
}
```

Figure 4.9: La déclaration de la liste de sensibilité d'un SC_METHOD.

d. Implémentation

L'implémentation suit les mêmes règles que les fonctions normales. Quand le processus a fini d'exécuter ce qu'il a à faire, il effectue un return () pour redonner la main au simulateur.

VI.6.2 SC_THREAD

- Ils sont des threads indépendants du scheduler.
- ils sont *lancés une seule fois*, au début de la simulation, et plus jamais après.
- ce sont des *boucles infinies*, qui peuvent et *doivent* être *mises en veille* par la fonction wait(). Dans un intervalle régulier. Les SC_THREAD sont *réveillés* par leur liste de sensibilité.

❖ **Création des SC_THREAD**

Le processus de création des SC_THREAD est exactement le même que celui des SC_METHOD, en utilisant la macro SC_THREAD.

VI.6.3 SC_CTHREAD

Les SC_CTHREAD sont des cas particuliers des SC_THREAD .Leur nom signifier *C*locked *THREAD*: threads synchrones. Ils sont utilisés pour modéliser des threads sensibles *uniquement* à un front d'horloge.

❖ **Création des SC_CTHREAD**

Un SC_CTHREAD est, comme les autres processus, une fonction de type voidf().

❖ **Enregistrement des SC_CTHREAD**

Un SC_CTHREAD est enregistré auprès du moteur de simulation par la macro SC_CTHREAD, en lui passant en argument le nom de la fonction associée ainsi que l'évènement correspondant au front d'horloge sur lequel le processus est synchrone.[33]

```
SC_MODULE(mon_module) {
 // déclaration des ports, signaux, ...
sc_in<bool> clock;

// déclaration d'un processus (pour l'instant rien ne spécifie son type)
voidmon_sc_cthread();

 // constructeur du module
 SC_CTOR(mon_module) {
   // Enregistrement de la fonction comme étant un SC_CTHREAD sensible au front montant de
l'horloge
   SC_CTHREAD(mon_sc_cthread, clock.pos());
 }
... } ;
```

Figure 4.10: La déclaration et l'enregistrement d'un SC_CTHREAD.

VII. Evènement

- ✓ Un processus dispose d'une liste de sensibilité décrivant les événements auxquels il doit réagir.
- ✓ Lorsque l'exécution d'un processus rencontre un wait(), il est interrompu.
- ✓ Il attend le déclenchement (implicite) d'un des évènements de sa liste de sensibilité statique.
- ✓ Lorsqu'un de ses événements est déclenché, le scheduler place le Thread dans la liste des Processus READY.

VII.1 SC_event

- ✓ Un event est un objet de classe sc_event.
- ✓ Il n'a ni durée, ni valeur.
- ✓ L'acte d'indiquer une modification d'un évènement est appelé notification
  - e.notify();
  - notify(e);
- ✓ L'évènement garde une liste des processus qui lui sont sensibles (par wait(e)).
- ✓ Lorsqu'un event est notifié, il informe le scheduler des processus à (re) lancé.

VII.2 Notification

- ✓ Immédiate : e.notify();
- ✓ Notification immédiate, le processus en attente est replacé immédiatement dans la liste Ready.
- ✓ Notification après un cycle d'évaluation appelé Delta-cycle, lorsque tous les processus en attente se seront exécutés.
- ✓ Place l'évènement en fil d'attente jusqu'à la durée indiquée. [33]

VIII. Sensitive

a. Sensibilité statique –

□La liste des évènements d'activation d'un process est déterminée avant la simulation (dans les constructeurs).

□Sensitive<< a << b;

□b. Sensibilité dynamique =

□Permet de remplacer la liste statique par un évènement désigné lors de la simulation.

□Le processus n'est alors plus sensible aux évènements de sa liste statique.

□wait(e);

IX. Les étapes d'installation du visuel c++ pour Windows

- Le répertoire de téléchargement contient deux sous-répertoires: «msvc71» et des exemples ».
- Le «msvc71 'répertoire contient les fichiers de projet et l'espace de travail à compiler le "systemc.lib« bibliothèque. Double-cliquez sur le SystemC.vcproj '
- fichier pour lancer Visual C + + 7.1 avec le fichier d'espace de travail. Le fichier espace de travail auront les commutateurs appropriés fixés à compiler pour Visual C + + 7.1.
- Sélectionnez `Construire SystemC» sous le menu Générer ou appuyez sur F7 pour construire`systemc.lib '.
- Les exemples répertoire `'contient les fichiers de projet et l'espace de travail à compiler les exemples SystemC. Allez à l'un des sous-répertoires d'exemples et double-cliquez sur le fichier. vcproj pour lancer Visual C + + avec le fichier espace de travail. Le fichier espace de travail aura les commutateurs appropriés mis en pour compiler pour Visual C + + 7.1. Sélectionnez «Construire <exemple>. Exe" dans le cadre du Dans le menu Générer ou appuyez sur F7 pour générer l'exécutable par exemple.

X. Conclusion

Toute au long de ce chapitre, nous avons présenté l'architecture générale de l'environnement SystemC et l'organisation structurelle des modèles en SystemC, par la suite nous avons étudié en détaille chaque composant les modules, les ports, les interfaces et les processus.

Finalement et après cette étude on éclaire le début de chemin pour développer notre système qui est construit et illustré dans le chapitre prochain.

Chapitre 5
Conception et implémentation

Chapitre 5 : Conception et implémentation

I. Introduction

Après une étude approfondie dans les chapitres précédents des systèmes embarqués qui contiennent un système complet sur puce, ces derniers deviennent de plus en plus complexes et inclut presque toujours une partie matérielle et une partie logicielle.

Nous allons utiliser les outils d'Eclipse qu'on vu dans les chapitre précédents pour maitre en œuvre une approche MDA pour modéliser ces système avec SystemC.

Notre objectif dans ce chapitre est de développer une application pour la génération automatique des modèles de simulation SystemC à Partir des Profils UML. Nous allons donner une vision plus détaillées aux composantes principales de notre application ainsi que les outils de modélisation, ensuite nous allons faire une étude de cas pour tester les différents mécanismes de fonctionnement de notre application.

II. Les Outils de Modélisation

II.1 Eclipse Platform

II.1.1 Présentation

Initialement, Eclipse était un environnement de développement dédié au langage Java. Actuellement, Eclipse possède un grand nombre d'extensions (ou *plugin*) qui ont des fonctions très diverses telles que l'export d'un projet en archive compressée Zip ou la représentation d'un projet sous forme de diagrammes UML.

Eclipse est en réalité constitué d'une multitude d'extensions permettant le codage d'un projet ou le codage d'extension dans le but de toujours augmenter les fonctionnalités fournies par Eclipse. A partir de ce point de vu, nous avons décidé d'utiliser cette plateforme parce qu'elle nous détache de plusieurs options qu'on trouve disponible directement dans Eclipse sous forme des plugins. Et au même temps, nous offrons la possibilité d'étendre facilement notre application pour qu'elle réponde à des nouvelles besoins.[38]

II.1.2 Architecture d'Eclipse

L'architecture de la plateforme Eclipse est présentée dans la figure la suivante :

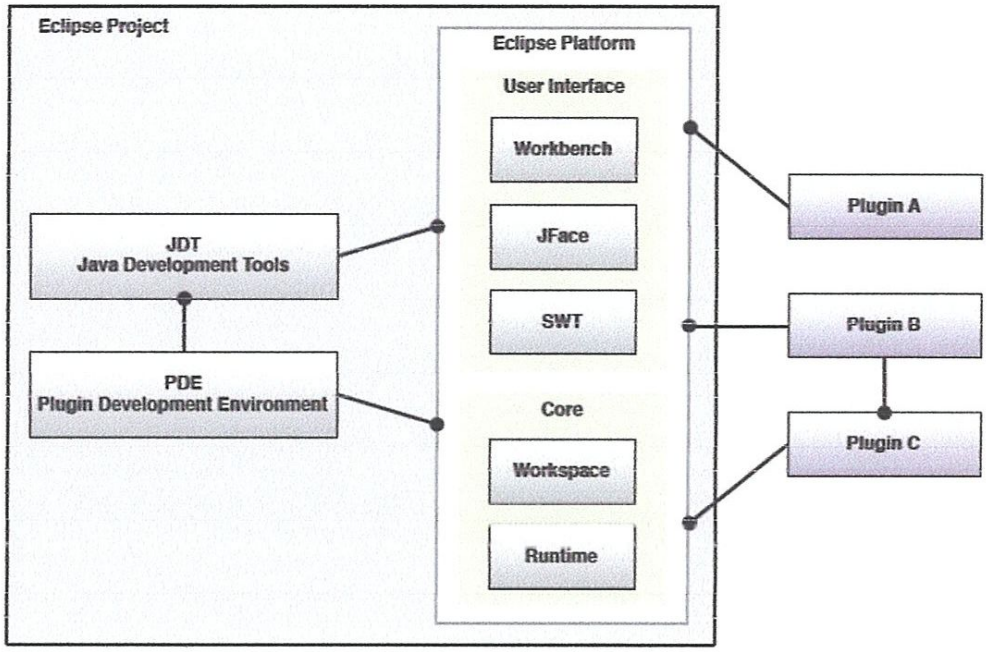


Figure 5.1 : *Architecture d'Eclipse*

Plugin	Rôle
Eclipse Runtime	Noyau principal d'Eclipse. Cette partie contient la gestion des extensions. C'est cette partie qui rend possible l'architecture modulaire d'Eclipse.
Workspace	Gestion des fichiers et des répertoires de travail.
Team	Module permettant les outils tels que CVS[1]. Cette partie définit la possibilité de travailler sur un projet en groupe et que ce soit géré dans Eclipse.

[1] CVS (Control Version System) une architecture client-serveur pour le développement des applications en équipe

Help	Gestion de l'aide, cet outil permet d'avoir à tout moment l'aide relative à l'action en cours.
SWT	Librairie graphique très rapide car elle est compilé en fonction de la machine exécutant le code. Elle est plus rapide mais moins souple et moins facile d'utilisation que SWING.
JFace	Une extension de SWT permettant une gestion des composants graphiques complexes tels que les listes.
Workbench	extension permettant une abstraction supplémentaire dans la définition de l'interface graphique. Cette partie gère les vues et perspectives d'éclipse.

Table 5.1 : *Rôle de plugin Eclipse*

II.2 Plugin Eclipse

Le composant le plus petit dans la plateforme Eclipse est les plug-ins. Eclipse permet au développeur d'étendre ses fonctionnalités via des plug-ins. Par exemple, nous pouvons ajouter à l'interface graphique standard d'Eclipse de nouvelles menues ou nouveaux éditeurs via des plug-ins. Dans notre application, nous avons étendu Eclipse par l'ajout des éditeurs, des vues, et des commandes de plus nous avons envisagé des messages qui s'affichent dans des différentes circonstances et qui sont gérés par l'environnement Eclipse. Nous pouvons alors dire que notre application est une extension d'Eclipse et elle ne s'agit guère d'une application autonome.

Dans les sous paragraphes suivants nous allons donner les principales composantes graphique fournis sous forme des plugins Eclipse. [3]

II.2.1 Les Vues

Les vues permettent de présenter des informations sur des ressources de n'importe quel type (objet, répertoire, fichier, dessin, etc.). Parmi les propriétés fondamentales des vues c'est qu'ils se réagissent aux événements produits par les changements d'état des ressources.

Les principales caractéristiques d'une vue sont :

- Les vues fournissent de l'information sur certains objets
- Certaines vues complètent les éditeurs
- Certaines vues complètent d'autres vues
- Il existe un point d'extension pour définir de nouvelles vues
- La Plateforme Eclipse possède de nombreux vues standards
- La Plateforme Eclipse offre API[2] pour les vues

- Les vues peuvent être implémentées avec les viewers JFace[3] [6]

II.2.1.1 Construction d'une Vue par extension

Les étapes suivantes donnent une démarche pratique pour l'utilisation des vues

- Sélection du point d'extension *org.eclipse.ui.views*

[2] API : Application Programming Interface. Une bibliothèque pour la gestion des vues
[3] JFace est une bibliothèque graphique équivalente à Swing

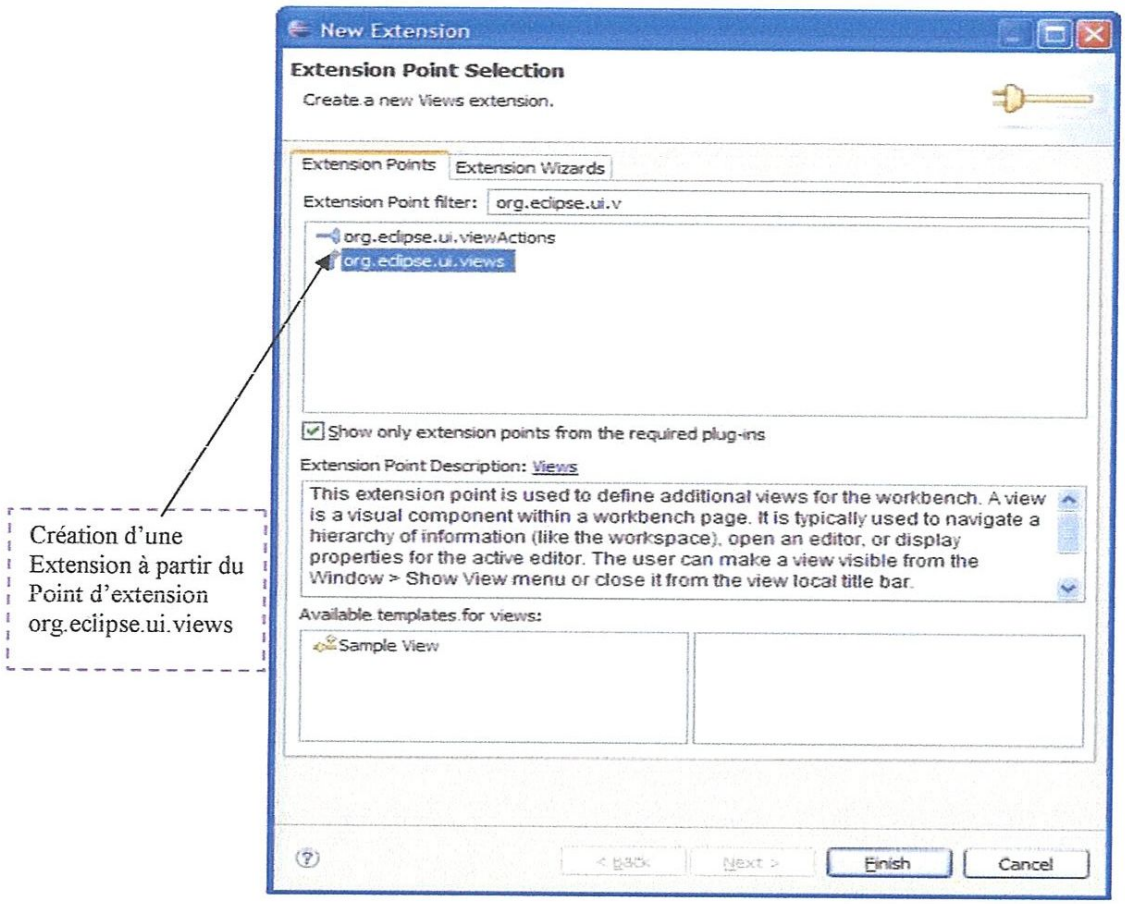


Figure 5.2 : *Construction d'une vue par extension*

- Définition des attributs de l'extension

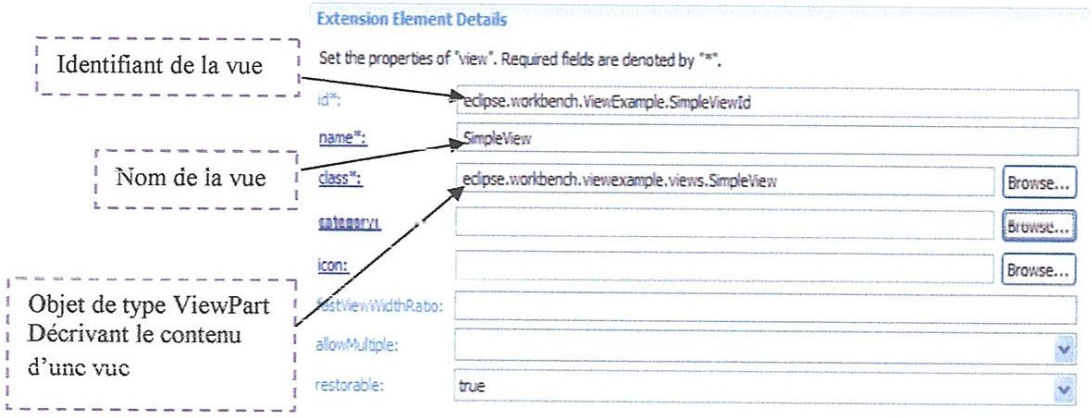


Figure 5.3 : *Définition des attributs de l'extension*

II.2.2 Les Editeurs

Eclipse utilise éditeurs et des vues de conserver les données. Un éditeur exige généralement que l'utilisateur explicitement sélectionnez "enregistrer" pour appliquer les modifications aux données tout en vue change généralement les données immédiatement.

Tous les éditeurs sont ouverts dans la même zone. Via la perspective que vous pouvez configurer si la zone de l'éditeur est visible ou non.

- Un éditeur est commun à toutes les perspectives d'une Fenêtre
- Si l'éditeur est fermé à partir d'une perspective il est fermé pour Toutes les perspectives
- Il n'est pas possible d'empiler une vue avec un éditeur
- Un éditeur n'est pas détachable
- Un éditeur a obligatoirement une barre de titre
- Un éditeur n'a pas de barre de menus et de barre d'outils localisées, il partage avec les barres de la fenêtre
- Un éditeur peut être associé à un nom de fichier ou à une extension et cette association peut être modifiée par l'utilisateur. [30]

II.2.2.1 Construction d'un éditeur interne par extension

- Sélectionner le point d'extension org.eclipse.ui.editors. [7]

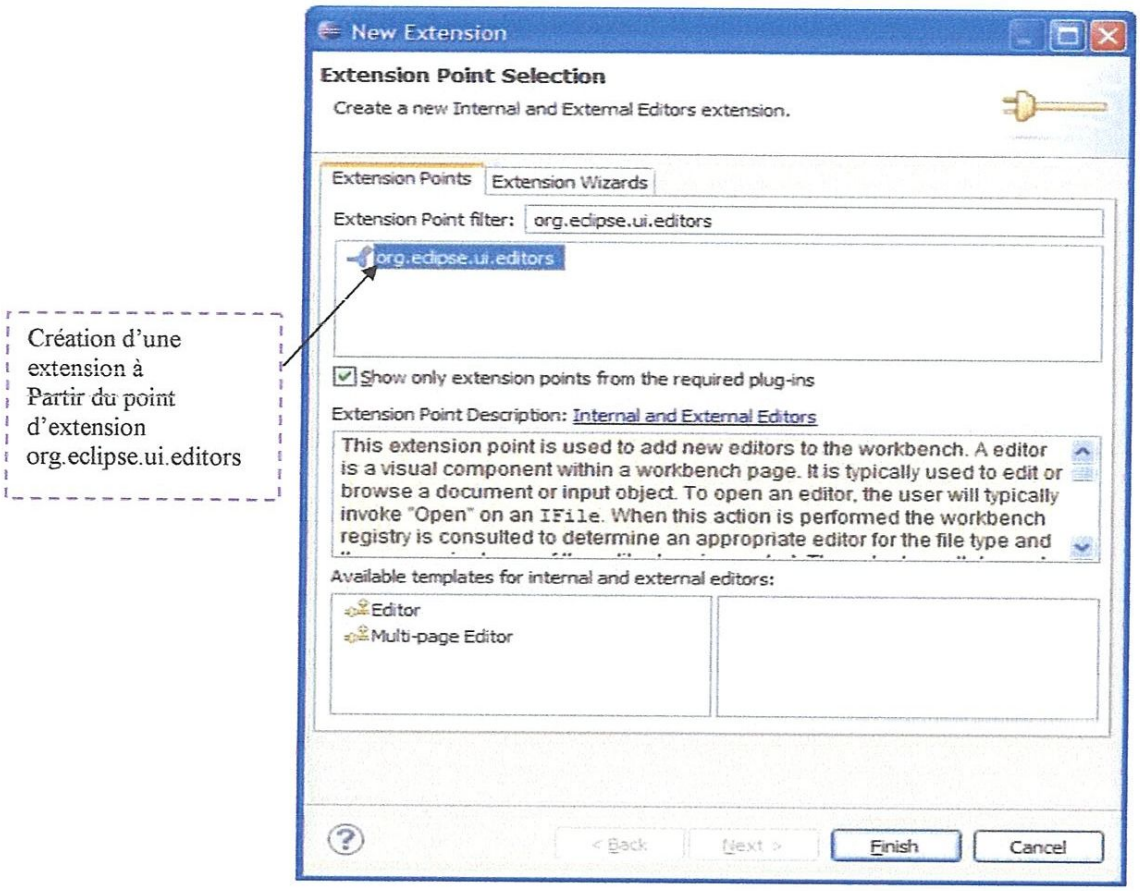


Figure 5.4 : *Création d'une nouvelle extension org.eclipse.ui.editors*

- Définir les attributs de l'extension

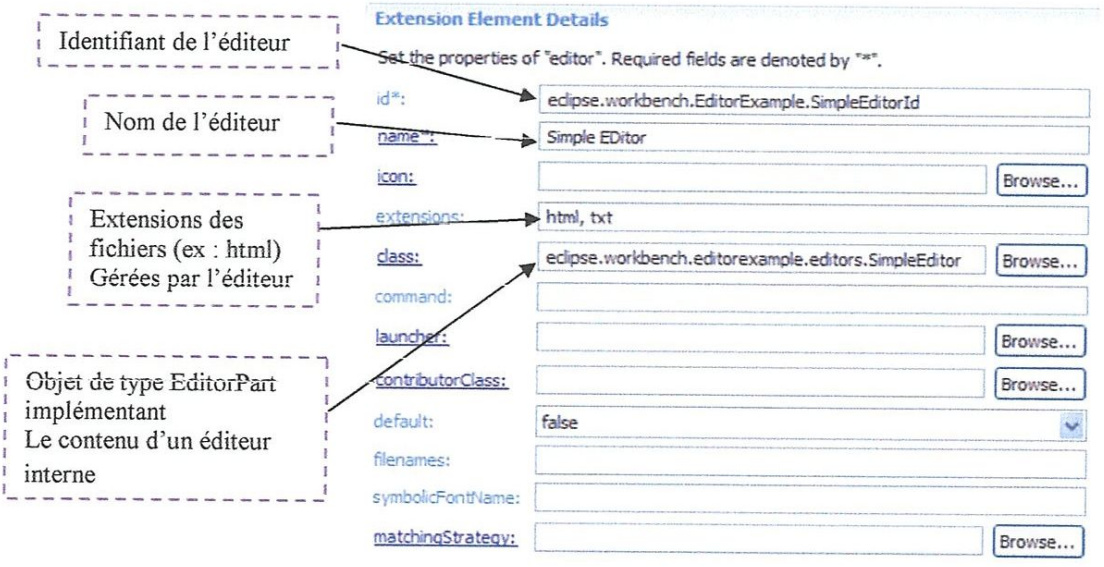


Figure 5.5 : *Définition des attributs de l'extension*

II.2.3 Les Commandes

Une commande en Eclipse est une description déclarative d'un composant et est indépendante des détails d'implémentation. Les commandes sont définies par l'org.eclipse.ui.commands point d'extension.

Une commande peut être classée, affectée à l'interface utilisateur et un raccourci clavier peut être défini pour la commande. Le comportement d'une commande est défini par un gestionnaire.

II.2.3.1 Définition des Commandes

Dans notre travail on a définir beaucoup de commande on prend comme exemple la commande ouvrir. Cliquez sur le fichier plugin.xml et sélectionnez l'onglet Extensions.

Appuyer sur le bouton « ajouter »

Rechercher l'extension "org.eclipse.ui.commands". Sélectionnez-le et appuyez sur Terminer.

Créer une nouvelle commande en cliquant-droit sur votre point d'extension et en sélectionnant Nouveau -> commande.

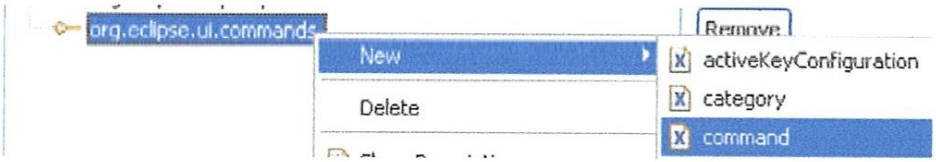


Figure 5.6 : *Création d'une nouvelle commande*

Définissez l'ID à "new.editor.ouvrirFichier" et le nom de "ouvrir". Entrez la classe "comportement" comme DefaultHandler. [30]

II.2.3.2 Utilisation des Ccommandes dans les Menus

La commande que nous avons définie doit être utilisée dans un menu. Ajouter une extension à l'org.eclipse.ui.menus point d'extension. Faites un clic droit sur le point d'extension et sélectionnez menuContribution → Nouveau.

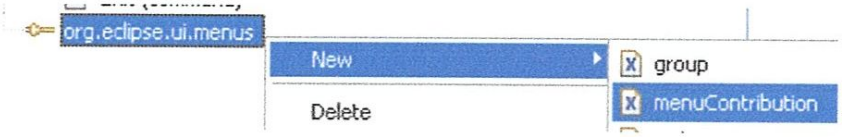


Figure 5.7 : *Utilisation des commandes dans les Menus*

Créer un nouveau menu avec la contribution du "menu: org.eclipse.ui.main.menu" locationURI. Assurez-vous que l'URL est correcte orthographié, sinon votre menu ne sera pas montré.

Faites un clic droit de votre menucontribution et sélectionnez Menu → Nouveau.Ajouter un menu avec le menu « fichier » étuquette et la « FileMenu » ID.[30]

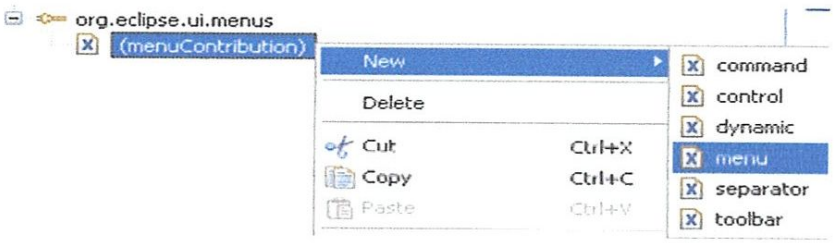


Figure 5.8 : *Création du Menu*

Il devrait en résulter une "plugin.xml" fichier qui ressemble à la suivante.

```
<plugin>
<extension
        point="org.eclipse.ui.newWizards">
</extension>
<extensionpoint="org.eclipse.ui.editors">
<editor
id="model.presentation.ModelEditorID"
name="%_UI_ModelEditor_label"
icon="icons/full/obj16/ModelModelFile.gif"
extensions="model"
class="model.presentation.ModelEditor"
contributorClass="model.presentation.ModelActionBarContributor">
</editor>
</extension>
<extension
point="org.eclipse.ui.commands">
<category
id="new.editor.SocCategory"
name="ouvrir">
</category>
<command
categoryId="new.editor.SocCategory"
defaultHandler="comportement"
id="new.editor.ouvrirFichier"
name="Ouvrir">
</command>
```

Figure 5.9 : *Fichier plugin.XML*

II.2.4 Dialogues dans Eclipse

Les Dialogues permettent de demander à l'utilisateur pour obtenir des informations supplémentaires ou fournir à l'utilisateur une rétroaction. La plate-forme Eclipse propose plusieurs SWT et JFace`Dialogues`.

Dans notre application on a utilisé les dialogues pour définir des messages d'alerte comme la suppression si le fichier existe (voir le code). [3]

```
JOptionPane jop = new JOptionPane();
                                        ImageIcon img = new
ImageIcon("C:/Users/winseven/workspace/behavior_modeling/img/atten.gif");
                                        int option = jop.showConfirmDialog(null, "Le projet existe ! Voulez-
vous supprimer ", "Information", JOptionPane.OK_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE,
img);

                                        if(option == JOptionPane.OK_OPTION) project.delete(true, null);
```

Figure 5.10 : *Déclaration d'un message de dialogue*

II.3 Eclipse Modeling Framework (EMF)

Nous avons utilisé EMF pour modéliser notre modèle de domaine (Les systèmes embarqués). EMF permet de créer le méta-modèle par différents moyens, par exemple XMI, annotations Java, UML ou un schéma XML. La description qui suit va utiliser les outils EMF directement pour créer un modèle EMF. [5]

II.3.1 Définition d'un Modèle EMF

Dans cette section nous allons présenter les étapes de réalisation d'un modèle EMF.

II.3.2 Création d'un Projet

Créez un nouveau projet via le menu Fichier / Nouveau / Projet ... / Eclipse Modeling Framework / projet EMF vide. [28]

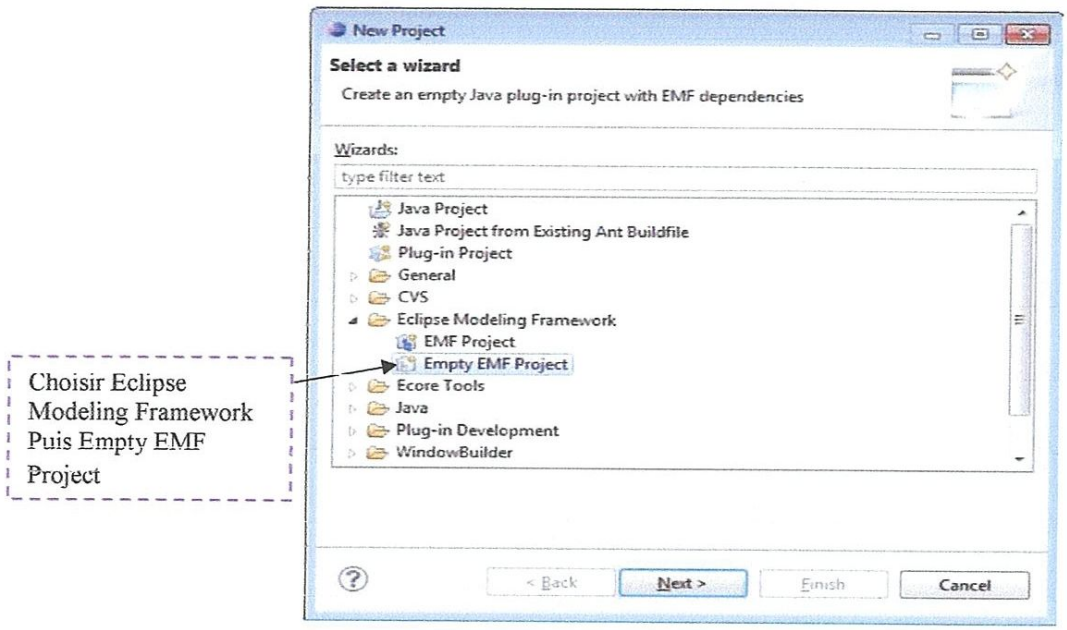


Figure 5.11 : *Création d'un projet EMF vide*

II.3.3 Création d'un Diagramme Ecore

Avec l'outil graphique « Ecore Tool » nous pouvons construire des modèles EMF d'une manière très simple. Sélectionnez le dossier «modèle», faites un clic droit sur le dossier et sélectionnez Nouveau / Autre ... Ou Outils / Diagramme de Ecore [5]

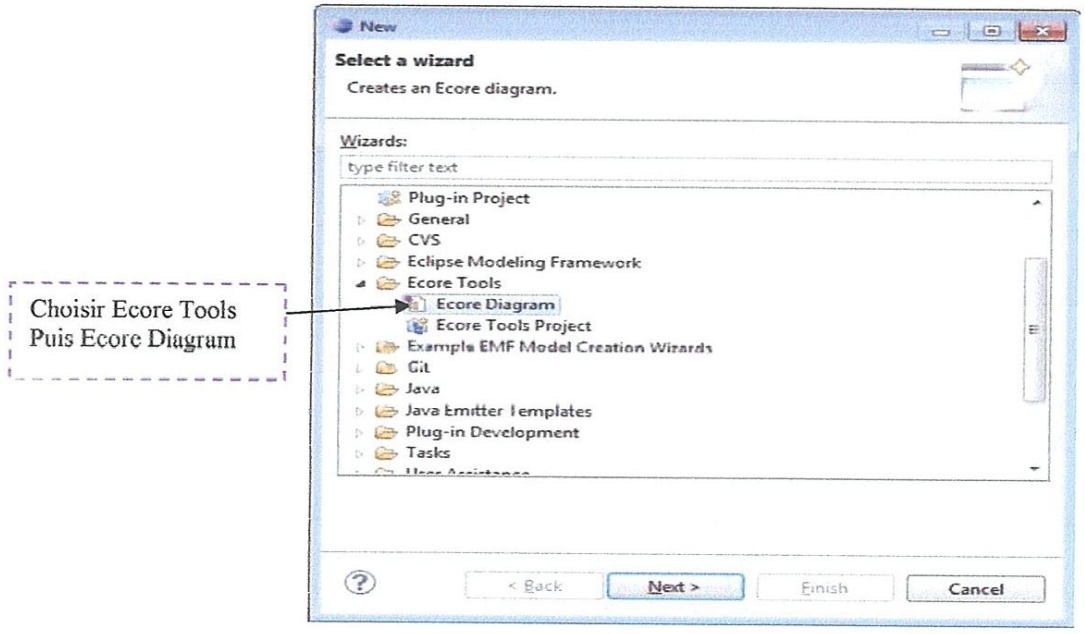


Figure 5.12 : *Création d'un Ecore Diagram*

Ceci devrait ouvrir un éditeur visuel pour créer des modèles EMF.

Nous allons utiliser les différents éléments (*EClass,EAttribute,EReference,EDataType)* pour créer le modèle

II.3.4 Création du Modèle

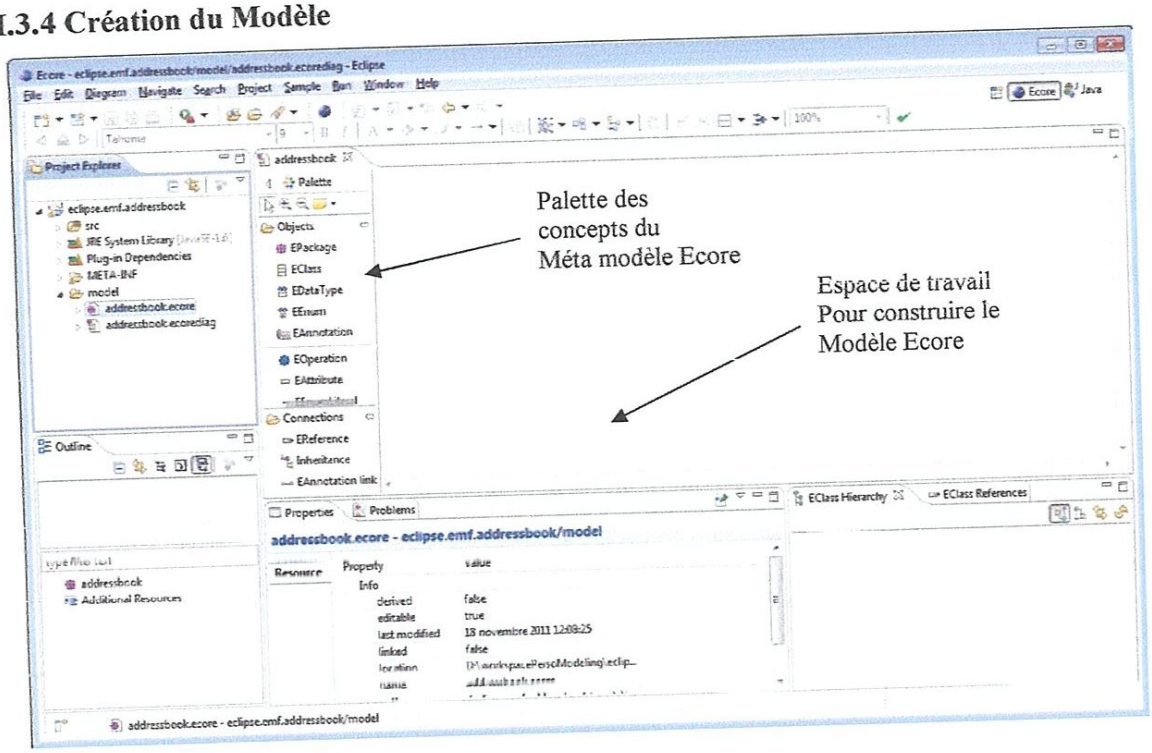


Figure 5.13 : *Espace de travail de la Plateforme Eclipse*

II.3.5 Création d'un Modèle Générateur EMF

Le modèle générateur permet de décorer le modèle EMF en précisant des détails concernant la génération du code.

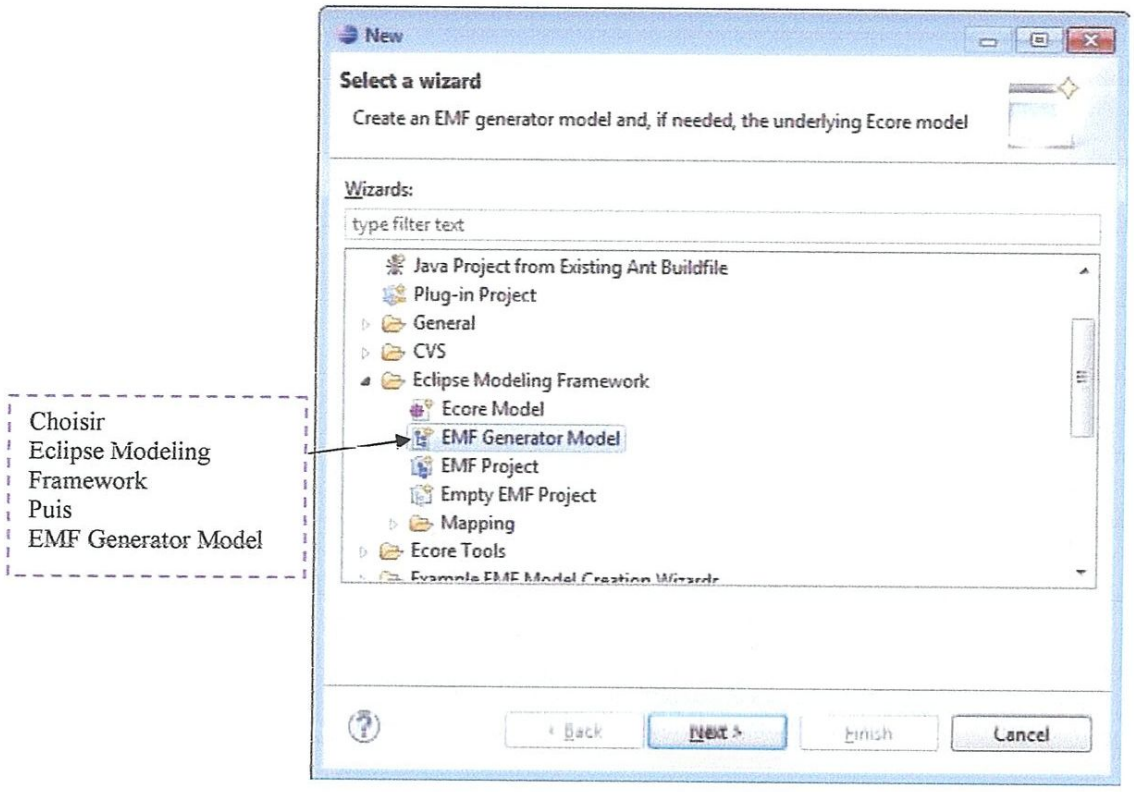


Figure 5.14 : *Création d'un modèle générateur EMF*

Sélectionnez notre modèle et appuyez sur la charge. [28]

II.3.6 Génération de Code Java

Nous avons créé deux modèles, le modèle EMF avec l'extension ". Ecore" et le modèle de génération avec une extension ". Genmodel" Sur la base de ces deux modèles, nous pouvons générer du code Java relatif au domaine modélisé.

De plus, la plateforme EMF permet de générer deux autres plugins :

« .edit » : permet de faire la liaison entre le code java du domaine avec l'éditeur visuel du deuxième plugin (.editor)

« .editor » : un plugin qui offre un environnement graphique (avec pleins d'options pour l'édition la persistance l'exportation, etc.) pour la définition des modèles instances du modèle EMF défini auparavant.

Cliquez-droit sur le nœud racine de l'Genmodel et sélectionnez "Générer le code du modèle". Cela va créer la mise en œuvre de Java du modèle EMF dans le projet actuel.

II.4 Java Emitter Template(JET)

JET est une plateforme fournie sous forme d'un plugin intégré dans Eclipse pour les transformations des modèles EMF vers des modèles textuel (exemple : code java, code C++, XML etc.). JET utilise une technologie de Template très proche de la syntaxe de Java Server Pages (JSP). [29]

La figure suivante explique le schéma de fonctionnement de l'outil JET

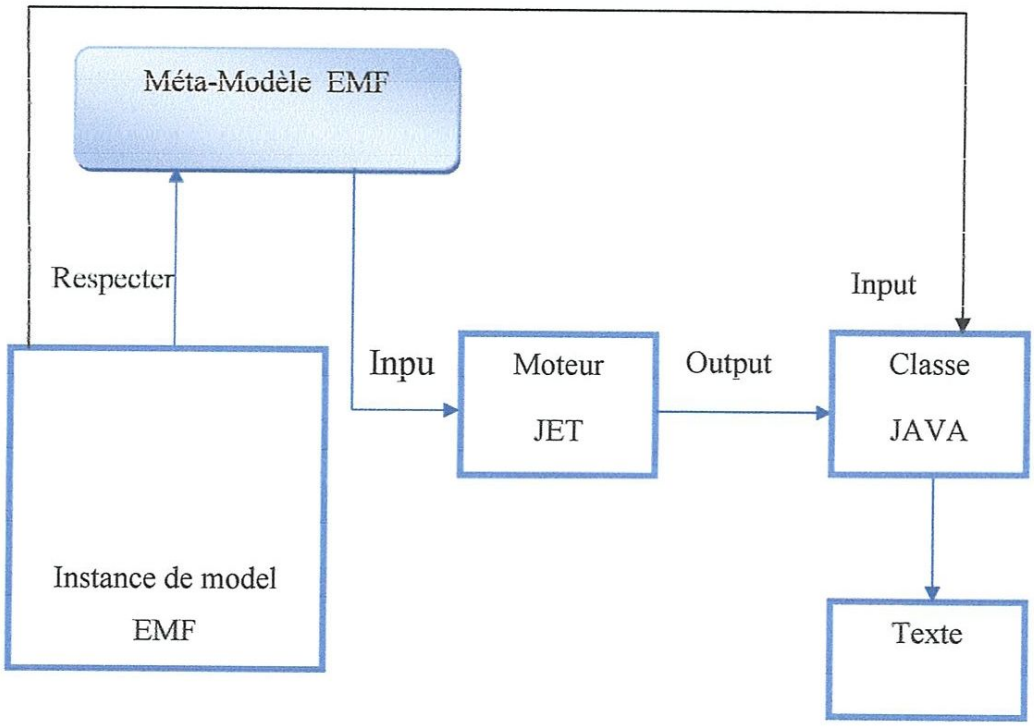


Figure 5.15 : *Schéma d fonctionnement de l'outil JET*

Les classes Java peuvent ensuite être utilisées pour créer la sortie finale, dans notre exemple, ils'agit d'un code SystemC. Cette classe générée peut être initialisé et créer le résultat souhaité en tant que chaîne avec la méthode "generate ()". Cette étape du procédé est appelée «génération».

II.4.1 Les Etapes de Réalisation d'une Transformation avec JET

II.4.1.1 Conversion d'un Projet en Projet JET

Pour utiliser JET au sein d'un projet, nous devrons convertir le projet à un projet JET. Sélectionnez Nouveau -> Autre, puis l'entrée suivante.

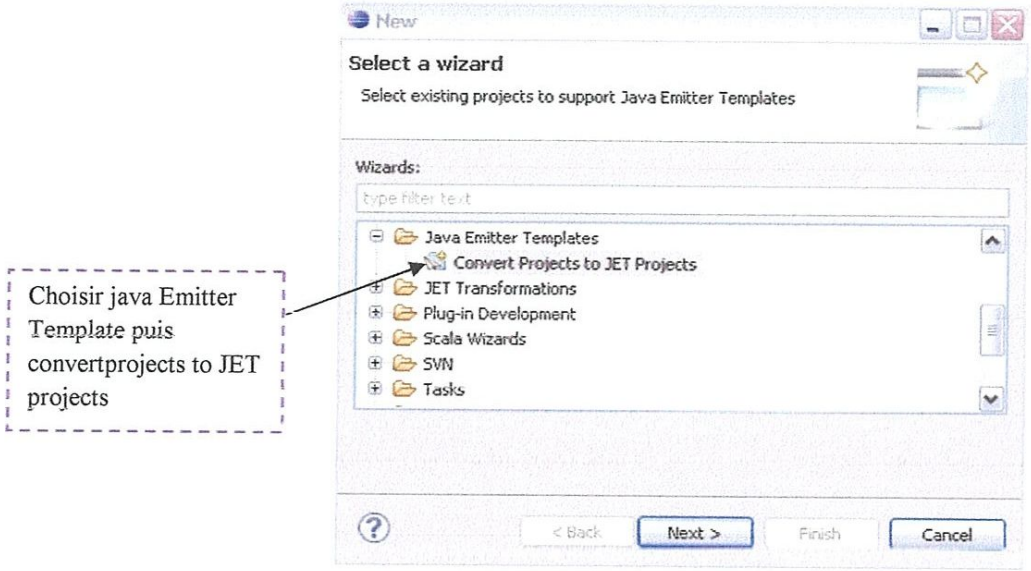


Figure 5.16 : *Conversion du projet en JET*

Sélectionnez le projet qui doit être converti. Appuyez sur Terminer.

Un nouveau dossier "templates" sera créé. Tous les modèles qui seront créés / sauvegardés, seront automatiquement convertis par le moteur JET. [29]

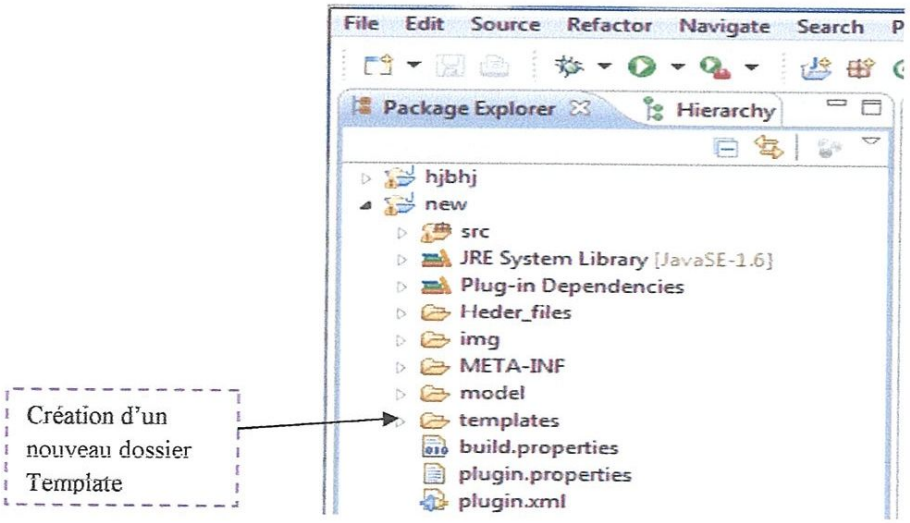


Figure 17 : *La création d'un nouveau dossier template*

Sélectionnez votre projet, faites un clic droit, sélectionnez Propriétés, sélectionnez Paramètres Jet et maintenir le «Container Source" comme "src".

II.4.1.2 L'Utilisation De JET

Sélectionnez le dossier Template faites un clic droit, Créez un fichier qui doit avoir le "jet" suffixe.

Exemple :

```
<%@ jet package="pfe"
class="c1" imports ="java.util.Iteratororg.eclipse.emf.common.util.EList model.* model.impl.* model.util.*
model.Process"
%>

<%Soc soc =(Soc)argument;%>

<%EList <Module> list =soc.getModules();%> // créer liste de Modules
<% for (Iterator <Module> I = list.iterator();I.hasNext();) {%>
<% Module mod =(Module) I.next();%>// pour chaque module affiche le nom
<%=mod.getNom()%>( ){

}

<%}%>
```

Figure 5.18 : *Exemple sur l'utilisation du JET dans les SoCs*

Après avoir enregistré le fichier une classe sera créée dans votre dossier source.

III. Le Méta-Modèle

Le cœur de notre application est les méta-modèles décrivant les différents concepts des systèmes embarqués.

Nous avons défini deux méta-modèles pour capturer les propriétés structurels et comportementales de ce type des systèmes. Nos méta-modèles sont basés sur des standards bien connus de l'OMG : les profils SoCML et les Diagrammes d'états/transition UML.

La figure suivante présente les deux méta-modèles, et la relation qui les associe

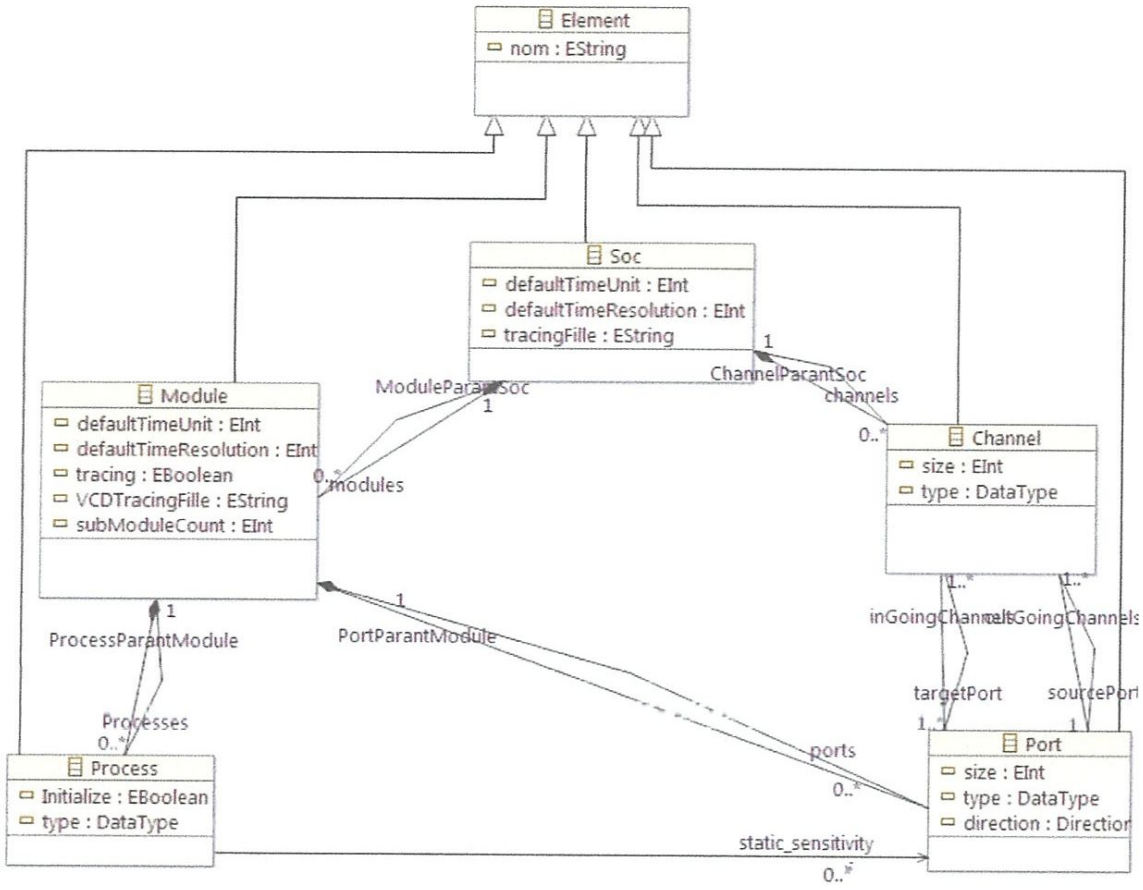


Figure 5.19 : *Méta-Modèle SocML*

Notre modèle fait référence à d'autres modèles (voir la figure)

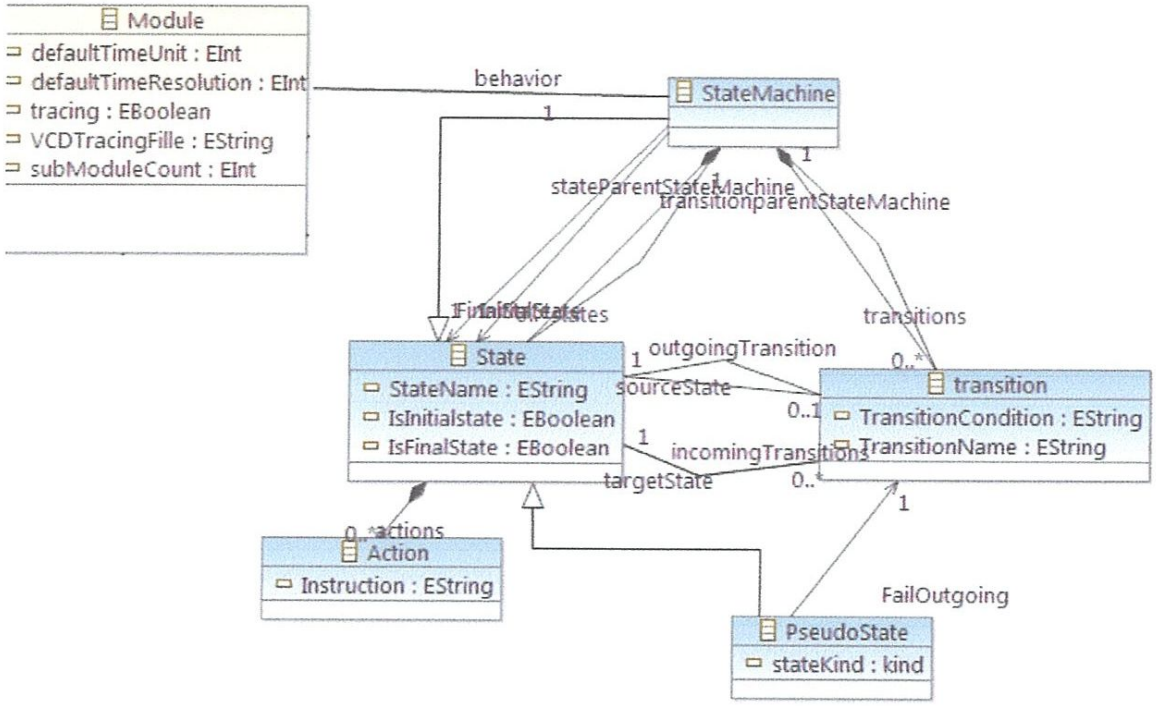


Figure 5.20 : *Méta-model pour les modules d'état transition UML*

IV. Validation des Contraintes d'Intégrités

Pour concevoir un formalisme pour un domaine spécifique, il ne suffit pas des définir uniquement le méta-modèles mais aussi nous avons besoin d'avoir un ensemble des règles qui permettent de valider les modèles instances. Ces règles sont inspirées du domaine modélisé.

La plateforme EMF offre entre autre fonctionnalité l'option de validation des modèles avec

- Le langage OCL[4].
- des méthodes Java.

Il existe en effet deux type de validation :

- La validation en Patch : la validation de toutes les contraintes se fait simultanément suite à la demande de concepteur du modèle.
- La validation Live : un événement signalant la présence d'une violation d'une contrainte est généré lorsque la violation aura lieu.

[4] Object Constraint Language

Dans notre travail, nous avons exprimé les contraintes sur nos méta-modèles avec le langage Java et nous avons opté pour la validation en patch.

Exemple 1:

Contrainte : le nom d'un Module ne doit pas être vide

Réalisation :

```
Int compteur=1;
boolean validateModule_nom_vide(Module module, DiagnosticChain diagnostics, Map<Object, Object> context) {
        boolean erreur = false;
        String nom = module.getNom();
        if (nom!="") i++;
        else {
                erreur= true;
        }
        if (erreur) {
                if (diagnostics != null) {
diagnostics.add (createDiagnostic (Diagnostic.ERROR,DIAGNOSTIC_SOURCE, 0,"_UI_GenericConstraint_diagnostic",new Object[] {
"nom_vide de module N°:"+i, getObjectLabel(module, context) },new Object[] { module }, context));
                        i++;
                }
                Return false;
        }
        Return true;
}
```

Figure 5.21 : *Scripte de vérification de « nom vide »*

Exemple 2 :

Contrainte : La vérification que le nom doit être unique

```
intj=1;
        String tab[]= new String [5];
        intk=1;
        intr;
                booleanerreur = false;
                String nom=module.getNom();
                if (nom!=null){
                        tab[j]=nom;
                        j++;
                }
                else{
                for(k=1; k<tab.length; k++){
          for(r=k+1; r<tab.length; r++){
           if (tab[k] contentEquals(tab[r])){  erreur=true;
           else erreur=false;
 if(erreur==true) break;
if(erreur==true) break;
                }
                }
if (erreur) {
on affiche ce message : "le même nom existe dans module "+k+"et module "+(k+1)
```

Figure 5.22 : *Scripte de vérification de « nom unique »*

Aussi d'autre algorithme de validation qui fait la vérification que le Channel ne lie pas deux ports du même module :

```
if(channel.getSourcePort().getPortParantModule().getNom()==
channel.getTargetPort().get(DIAGNOSTIC_CODE_COUNT).getPortParantModule().getNom()) err=true;

if (err) {
on affiche"channel : "+channel.getNom()+" lie deux ports de la même module"
```

Figure 5.23 : *Scripte de vérification de connexion de Channel*

V. Génération de Code SystemC

Pour générer le fichier header files on a créé une commande s'appelle « génération » qui permit de générer du code en systemC à l'aide d'algorithme ci-dessous :

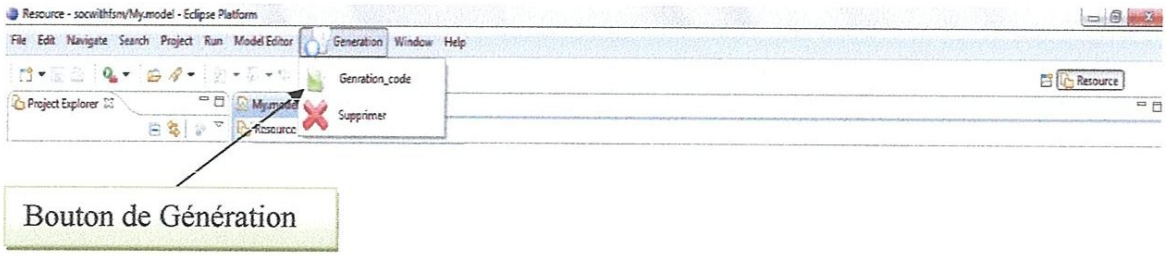


Figure 5.24 : *La commande de génération des header files*

Exemple :

```
Algorithmegeneration_heder ;
VarSocsoc;
e : char ;
mod :Module ;

p : Port ;

r : Process ;

i ,j,u:entier ;

debut
e='' ;
pour(i=1 , i<= nb_module , i++){
        e =e+' ' #include "systemc.h"
    '#ifndef' + mod.getNom()+'_H
     #define '+mod.getNom()+'_H' ;

E=e+'sc_module'+mod.getNom()+'{';
pour (j=1,j<=nb_port,j++){
e=e+p.getDirection()+'< '+p.getType()+'>'+p.getNom();
}
pour (u=1,u<=nb_porcess,u++){
e=e+r.getType+r.getNom();
}
E=e+'}
#endif' ;
}
```

Figure 5.25 : *Algorithme de génération des header files*

Exemple en JET :

Le fichier **hader.jet** pour la génération des fichiers **nom_de_module.h**

```
<%@ jet package="pfe"
class="hader" imports ="java.util.Iteratororg.eclipse.emf.common.util.EList model.* model.impl.* model.util.*
model.Process"
%>
/* declaration */
<%Socsoc =(Soc)argument;%>
<%EList<Module>list =soc.getModules();%>
/* pour chaque module de la liste « list » en affiche la suite */
<% for (Iterator <Module> I = list.iterator();I.hasNext();)
{%>
<% Module mod=(Module) I.next();%>
#ifndef<%=mod.getNom()%>_H/* l'entête de module*/
#define<%=mod.getNom()%>_H /* l'entête de module*/
sc_module(<%=mod.getNom()%>){
/* traitement sur les ports de module*/
<%EList<Port>listt =mod.getPorts();%> //ports
/* pour chaque port de module en affiche la direction, le type et le nom*/
<% for (Iterator<Port> J = listt.iterator();J.hasNext();){%><% Port p =(Port)
J.next();%>SC_ <%=p.getDirection()%><<%=p.getType()%>><%=p.getNom()%>; <%}%>
/* traitement sur les process de module*/
<%EList<Process>listts =mod.getProcesses();%>//processes
/* pour chaque process de module en affiche le type et le nom*/
<% for (Iterator <Process> U = listts.iterator();U.hasNext();){%><% Process r =(Process)
U.next();%><%=r.getType()%><%=r.getNom()%>();

<%}%>
}
#endif/* fin de module*/
<%}%>
```

Figure 5.26 : *Scripte de génération du fichier* ***nom_de_module.h*** *en* ***.JET***

Exemple :

Le fichier **sc_main.jet** pour la génération de fichier **sc_main.cpp**

```
<%@ jet package="pfe"
class="sc_main" imports ="java.util.Iteratororg.eclipse.emf.common.util.EList model.* model.impl.*
model.util.* model.Process" %>
intsc_main()/* l'entête */
{
//channels
<%Socsoc =(Soc)argument;%>
<%EList<Channel>list =soc.getChannels();%>
/* pour chaque channel en affiche le nom et le type*/
<% for (Iterator <Channel> I = list.iterator();I.hasNext();){%><% Channel ch =(Channel)
J.next();%><%=ch.getNom()%><<%=ch.getType()%>>
<%EList<Port>listt =ch.getTargetPort();%><% for (Iterator <Port> Y =
listt.iterator();Y.hasNext();){%><% Port p =(Port) Y.next();%><%=p.getNom()%>;<%}%>
<%}%>

// modules
/* déclaration de la liste des modules*/
<%EList<Module>listr =soc.getModules();%>
<%int a=1;%>
<%int h=1;%>
<%int z;%>
<%String u= null;%>
/* la récupération de nom de chaque channel*/
<% for (Iterator <Channel> S = list.iterator();S.hasNext();){%><% Channel ch =(Channel)
S.next();%><%u=ch.getNom();%>
<%}%>
/* pour chaque module le nom, le port et le channel que relit ce port*/
<% for (Iterator <Module> I = listr.iterator();I.hasNext();)
{%>
<% Module mod=(Module) I.next();%><%=mod.getNom()%> M<%=a%>
/*debut de module*/
(<%=mod.getNom()%>){/*le nom de module*/
M<%=a%>.<%EList<Port>listtr =mod.getPorts();%><% for (Iterator<Port> N =
listtr.iterator();N.hasNext();){%><% Port p =(Port)
N.next();%><%=p.getNom()%>(<%=u%>)<%}%>/*le nom de port + (nom de channel)*/
<% z=a++;%>
}
/*fin de module*/
<%}%>
 }/*fin*/
```

Figure 5.27 : *Scripte de Génération du fichier* ***sc_main.cpp*** *en* ***JET***

VI. Etude De Cas

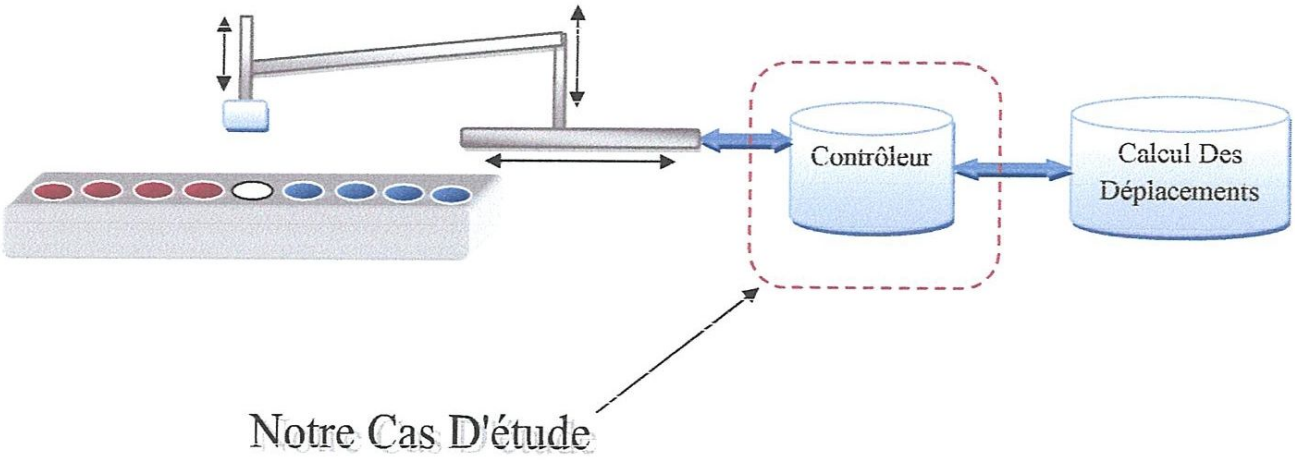


Figure 5.28 : *Le fonctionnement du robot*

VI.1 Contrôleur du Robot l'exemple de RTI

Pour mettre en évidence l'importance de notre travail. Nous illustrons la construction d'un contrôleur du robot qui a été déployé dans un robot qui résout le jeu de navette.

Dans ce jeu, il y a neuf places et huit pions (4 rouges initialement à gauche et 4 noirs à droite) dans les ces places. Avec une place vide dans le milieu. Le but de jeu consiste à déplacer les pions d'une coté à l'autre. Le gagnant c'est lui qui réussit à déplacer tous ses pions vers l'autre côté. Tout en respectant les règles suivantes :

(1) les pions rouges peuvent bouger seulement à droite, et les pions noirs peuvent bouger seulement à gauche.

(2) un pion peut bouger ou par un pas quand la place vide est adjacente à lui, ou par un saut sur un pion de la couleur opposée à la place vide adjacente.

Les calculs des déplacements sont faits par un module de spécifique qui transmet ses instructions au contrôleur du robot. Ce dernier s'occupe d'agir sur le robot pour effectuer les déplacements.[4]

VI.2 Description du Contrôleur

Le robot possède 2 degrés de liberté de mouvement : horizontale(X) et verticale(Y). En plus d'un électro-aimant en haut qui permet de récupérer/lâcher les pions. Les

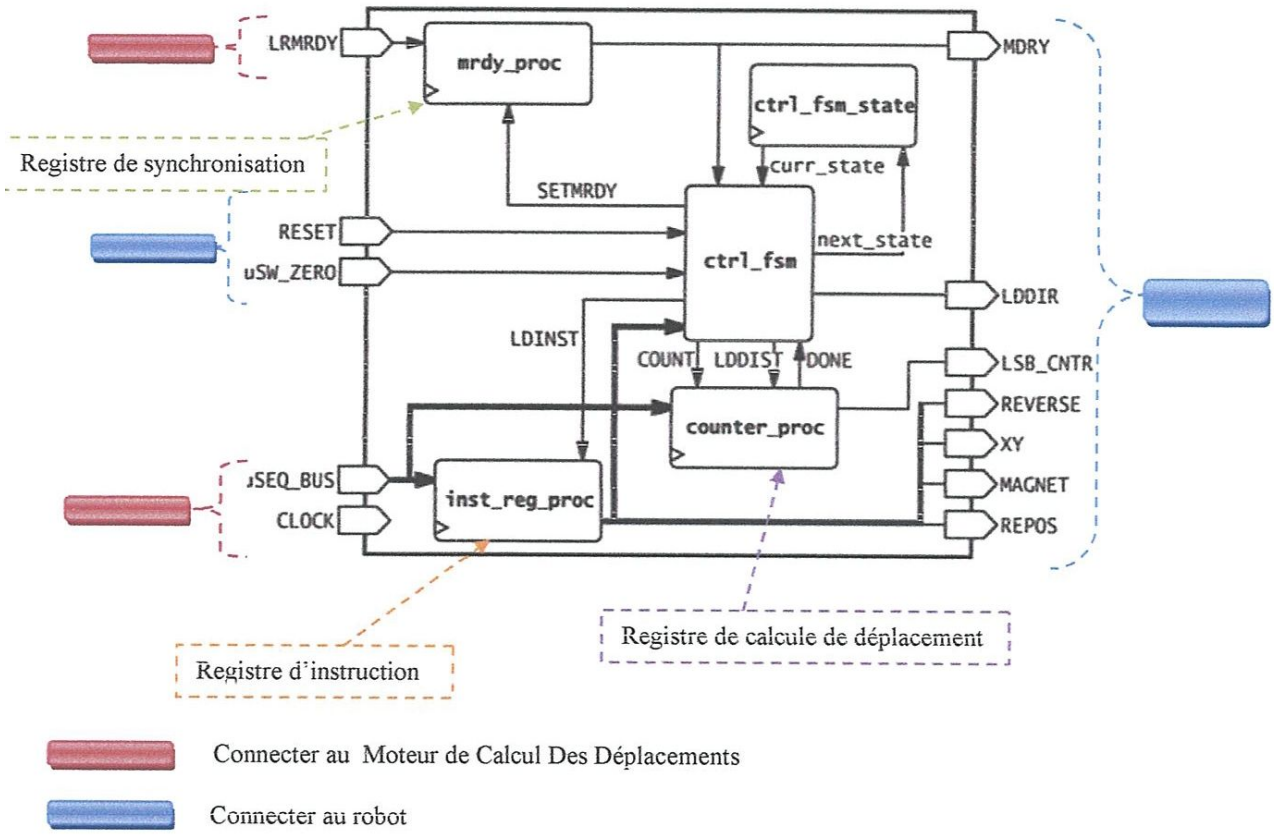
déplacements (horizontale et verticale) sont assurés par moteurs pas à pas, et dans chaque extrémité il y a un micro-switch qui fournit un point de la référence absolu indiquant la ternissant de la course.

Le contrôleur du robot émis des instructions au robot pour contrôler ses mouvements. Chaque instruction est composée du quatre bits. Le rôle de chaque bit ainsi que les combinaisons possible sont décrits dans la table suivant :

Moteur en Repos	Electro-aiment	Dépacement XY	Déplacement Reverse	Actions
0	0	0	0	**Déplacer X pas en avance**
0	0	0	1	**Déplacer X pas en arrière**
0	0	1	0	**Déplacer Y pas en avance**
0	0	1	1	**Déplacer Y pas en arrière**
0	1	-	-	**Alémenter l'electro-aiment**
1	1	-	-	**couper l'electro-aiment**
1	0	0	-	**Réinitialiser Moteur à X**
1	0	1	-	**Réinitialiser Moteur à Y**

Tableau 5.2 : *Instruction qui chiffre du contrôleur du moteur*

Le fonctionnement du contrôleur est achevé à l'aide des 3 registres: un registre d'instruction à 4 bit, un registre à 8 bits servant comme comptoir pour comptabiliser le déplacement du moteur pas à pas, et un 1 registre un seul bit pour la synchronisation entre le contrôleur et les module de calcul des déplacements.[4] La figure suivante donne la structure du contrôleur



Figures 5.29 : *Diagramme structurel du contrôleur du robot*

Le fonctionnement du controleur est dérit à l'aide de la machine d'étas finis suivante :

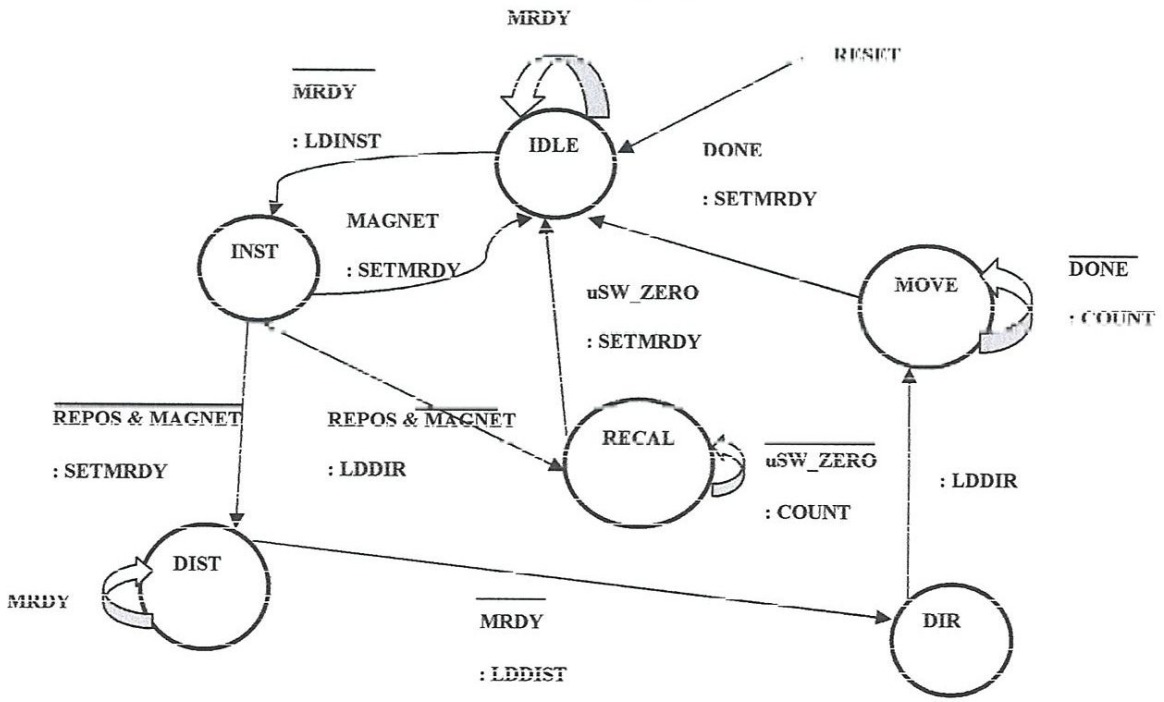


Figure 5.30 : *Machine de l'état finie pour le contrôleur du robot.*

Les nœuds précisent les différents états des contrôleurs et les transitions sont conditionnées par l'apparition de certains signaux et à leurs tours provoques des actions. Les conditions et les actions de chaque transition sont séparées par deux points verticaux.

	Nom	Signification
Etats	IDLE	Le contrôleur est à l'état repos
	INST	Le contrôleur passe à cet état une fois le signal MRDY passe à 0. Le signal MRDY est passé à 0 par le module de calcul de déplacement pour indiquer au contrôleur qu'une instruction est disponible au niveau du bus uSEQ_BUS
	MOVE	Le contrôleur passe à cet état une fois la direction est chargée
	DIR	Le contrôleur passe à l'état direction si le signal MRDY est passé à 0 pour indiquer au contrôleur le chargement de la destination
	DIST	Le contrôleur passe à l'état destination si les signaux REPOS et MAGNET sont mis a 0
	RECAL	Le contrôleur passe à l'état RECAL une fois le signal REPOS passe à 1 et le signal MAGNET passe à 0
Conditions	MRDY	Signal pour la synchronisation il prend la valeur 0 ou 1 pour indiquer au contrôleur qu'une instruction est disponible au niveau du bus uSEQ_BUS
	DONE	Il prend la valeur 1 une fois le compteur de déplacement atteint ZERO
	MAGNET	Il prend la valeur 1 si l'électro-aimant est chargé
	REPOS	Il prend la valeur 1 lorsque le moteur

		est au repos
	uSW_ZERO	Mettre le micro-Switch à zéro
Actions	COUNT	Décrémente le compteur de déplacement
	RESET	Réinitialiser le controleur
	SETMRDY	Permet une synchronisation entre le contrôleur et le module de calcul de déplacement.
	LDINST	Ordonne le processus contrôlant le registre d'instruction de charger l'instruction courante à partir du uSEQ_BUS
	LDDIR	Ordonner le robot pour changer la direction du mouvement (dans les 2 dimensions)
	LDDIST	Un sinal présignant la demensuon du mouvement (x : horzontal, y :vertical)

Tableau 5.3 : *signification des composants de l'automate*

VI.3 Modèle SystemC du contrôleur

Ayant examiné les fonctionnalités du contrôleur du robot, maintenant laissez-nous se mettre à écrire une description SystemC pour lui.

La description du modèle en SystemC se génère automatiquement à partir de 2 descriptions : structurelle et comportementale.

La partie structurelle précise les modules, les ports et leurs connexions, les processus et leurs listes de sensibilité. La partie comportementale augmente tout d'abord la partie structurelle en ajoutant 2 ports au module concerné[5] : un port pour l'horloge « *clock* » qui va etre utilise pour synchroniser les différents processus du module et le port *reset* qui permet de réinitialiser le module. Et après il définit 2 autres processus :

Ctrl_FSM_State : pour calculer l'état courant du Module après l'occurrence de n'importe quel évènement.

Ctrl_fsm : pour contrôler l'exécution des différents processus du module en fonction de l'état courant et les évènements produit au niveau des ports.

[5] Le Module qui est l'objet de la description comportementale.

Des liens internes sont insérées entre ce processus et les autre processus pour assurer la communication.[4]

Le code du processus Ctrl_fsm est consitué essentiellement par une structure à choix multiple **switch … case** où les clauses **case** correspondent aux états de l'automate. La figure suivante donne un aperçu sur le code du processus Ctrl_fsm de notre cas

```
If (RESET.read()) {
Ns = IDLE ;
} else {
Switch (curr_state.read ()) {
Case IDLE :
    If ( !MRDY.read()) {
        LDINST.write(true) ;
        Ns =INST ;
    } break ;
Case INST :
    If ( MAGNET.read()) {
        SETMREDY.write(true) ;
        Ns =IDLE ; }
  Else If ( ROPOS.read()) {
        LDDIR.write(true) ;
        Ns = RECAL ;}
    Else {  SETMRDY.write(true) ;
        Ns = DIST ; }
  }
  break ;
Case RECAL :
  If ( uSW_ZERO.read()) {
      SETMRDY.write(true) ;
      Ns =IDLE ;
  } else {
      COUNT.write(true) ,
   }
  break ;
```

```
Case DIST :
      If ( !MRDY.read()) {
           LDDIST.write(true) ;
           Ns =DIR ;
       }
      break ;
Case DIR :
           LDDIR.write(true) ;
           Ns =MOVE ;
           break ;
Case MOVE :
       If ( DONE.read()) {
            SETMREDY.write(true) ;
            Ns =IDLE ; }
      Else {
            COUNT.write(true) ;
             }
     break ;
} /* switch */
```

Figure 5.31 : *aperçus sur le code du processus Ctrl_fsm.*

Et pour réaliser cette structure, nous avons défini un algorithme qui permet de générer le corps du processus *ctrl_fsm* du module qui fait l'objet d'étude. La figure suivante présente L'algorithme de génération de comportement de module.

```
Algorithme SocWithFsm ;
Var e : char ;
soc : Soc ;  mod :Module ;
p : Port ;  smachine :StateMachine ;
sta : State ;  r : Process ;
trans : Transition ;
i ,j,u,o,a,t :entier ;
Debut
e= '' ;
e =e+' #include "systemc.h"' ;
```

```
pour (i=1 ; i<=nb_module ; i++)
{
e= e+mod.Nom() +'( ){'

    e=e+ 'sc_in<bool>CLOCK ;
sc_in<bool>RESET ;' ;

pour(j=1 ;j<=nb_port ;j++){
e=e+ 'sc_'+p.Direction() +'<'+p.Type()+'>'+p.Nom();

}

e=e+ '/* Internal variables and signals */
 /* Member function prototypes */
/* Constructor */'
};
e=e+smachine.getName()+ '{' ;
pour (u=1 ;u<=nb_state ; u++){
e=e+ '{'+sta.Name() +'}' ;
sc_signal<ctrl_state>curr_state,next_state;
}

pour (t=1 ;t<=nb_process ;t++){
e=e+'void '+r.Nom()+'()';
}
e=e+ 'void ctrl_fsm_state();
voidctrl_fsm();
/* Constructor */
SC_CTOR(robot_controller){' ;
pour  (a=1 ;a<=nb_process ;a++){
      e=e+ 'SC_METHOD ('+r.Nom()+')    sensitive <<CLOCK.pos>> ';
}
e=e+ 'SC_METHOD(ctrl_fsm_state),      sensitive <<CLOCK.pos>>';
SC_METHOD(ctrl_fsm);          sensitive <<CLOCK.pos>>' ;

e=e+'sensitive<<RESET' ;
pour (o=1 ;o<=nb_transition ; o++){
       e=e+ trans.Events()+ '<< ' ;
}
e=e+'curr_state ';
'}' ;
Void
Robot_controller :: ctrl_fsm
{
Control_state ns =curr_state ;
}
If (RESET .read()) {
Ns = IDLE ;
} else {
Switch (curr_state.read()) ';
pour(tr=1, t<= nb_transition, tr++){
e=e+ 'case '+trans.getTransitionGuard().getName()+ ' :
if (! '+trans.getTransitionGuard().getName()+'.read()) {
```

Bibliographie

[1]- Dave Steinberg, Frank Dubinsky, Marcelo Paternostro, Ed Merks, «EMF Eclipse Modeling Framework». Series Editors. Erich Gamma, Lee Nackman, John Weigand.16 December 2008,744 pages, Print ISBN-13: 978-0-321-33188-5.

[2]- Berthold Daum, « Professional Eclipse 3 for Java™ Developers». WILEY, 2005,602 pages, ISBN: 3-89864-281-X, England

[3]- Eric Clayberg, Dan Rubel, «Best-selling plug-in guid updated for Eclipse 3.4: Eclipse Plug-ins» Addison Wesley. Boston, San Francisco, New York. Erich Gamma, Lee Nackman, John Weigand, 2009,857 pages, ISBN 0-321-55346-2 (pbk.:alk. paper)

[4]- Thorsten Grotker, Stan Liao, Grant Martin, Stuart Swan, «System Design With SystemC». 2002,236pages, ISBN: 1-4020-7072-1.USA

[5]- Mickaël BARON, «Développement de clients riches : Plateforme Eclipse RCP : Modélisation via EMF».2012 ,81 pages, France

[6]- Mickaël BARON, «Développement de clients riches : Plateforme Eclipse : Conception de plug-ins Workbench : Views». 2009 ,95 pages, France

[7]- Mickaël BARON, «Développement de clients riches : Plateforme Eclipse : Conception de plug-ins Workbench : Editors», 2010 ,207 pages, France

[8]- Adolf Samir ABDALLAH, « Conception de SoC à Base d'Horloges Abstraites : Vers l'Exploration d'Architectures en MARTE ». Thèse de Doctorat. UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES, LILLE - FRANCE

[9]- Boutekkouk Fateh, « Aide à la Conception des Systèmes Multiprocesseurs Mono-puce à partir de Spécification de haut niveau ».Thèse de Doctorat, Université de Constantine, 27/01/2010

[10] - Sylvain ANDRE, MDA (Model Driven Architecture) principes et états de l'art, 05 novembre 2004.

[11]- LAURENT AUDIBERT, « UML 2 - de l'apprentissage à la pratique ».
FNAC, amazon.fr, internet :http://laurent-audibert.developpez.com/Cours-UML/html/Cours-UML.html

[12]- Philippe Marquet, Samy Meftali « Modélisation d'IP pour la simulation SystemC d'OS de type Linux embarqué ».Mémoire de Master, Université des Sciences et Technologies de Lille, 1er Juillet 2005, France

[13]- E. Riccobene , P. Scandurra , A. Rosti , S. Bocchio «A UML 2.0 Profile for SystemC: Toward High level SoC Design». EMSOFT '05, 2005, page 138-141.

[14]- Murray Woodside, Dorina Petriu, « Capabilities of the UML Profile for Schedulability Performance and Time (SPT) ». Dept of Systems and Computer Engineering, Carleton University, April 2004, Ottawa Canada

[15]- Philippe DESFRAY, « Profiles UML et langage J : Contrôlez totalement le développement d'applications avec UML ». Softeam ,1999

[16]- JAMAL ADD-ALI, « Méta Modélisation et Transformation Automatique du PSM dans une Approche MDA ». Université du Québec en Outaouais, MAI 2006.

[17]-Graiet Mohamed, Bhiri Mohamed Tahar, Dammak Faïza, Jean-Pierre Giraudin, « Adaptation d'UML2.0 à l'ADL Wright ». (Faculté des Sciences de Sfax), 2001.

[18]- *Xavier Le Pallec,* « **Gestion de méta-données avec le Meta Object Facility**». Actes de la conférence OCM'2000 ,7 juin 1999, FRANCE

[19]- Nicolas Palix, « Modélisation et traduction de systèmes sur puce à base de composants ».*PFE – Filière Informatique et Réseaux,* INRIA, ESISAR – Valence.

[20]- Chantal Taconet, « EMF Eclipse Modeling Framework Startup ». Aide-mémoire, Février 2008.

[21]- Jacques Barzic, « Eclipse et ses plugins de modélisation (EMF – GEF – GMF)». 2ème partie : *Eclipse* Modeling Framework *(EMF),* 12 janvier 2008.

[22]- B. Miramond, «SystemC, Joined design at system level (based on SystemC 2.2) ». Page 1-100.

[23]- DJOUDI Adel, « De la relation entre méta-modélisation et la réflexion ». STL. APR ALADYN 2011.

[24]- Naoui Jihène, Daknou Amani « UML2.0 Profile pour la conception des systèmes embarqués ».

[25]- Finn Overgaard Hansen, Peter Gorm Larsen «An Introduction to SysML».

[26]-HungLe-Dang, «Profil UML SPT» http://tacos.loria.fr/drupal/?q=system/files/ProfileUMLSPT.pdf.Dernière mise à jour Juin2007

[27]-Draft RFC Submission to OMG «A UML Extension Profile for SoC».Aout 2004. http://www.uml-sysml.org/documentation/uml-profile-for-soc-319-ko/view

[28]- Lars Vogel, «Eclipse Modeling Framework (EMF) ».Tutorial, 27-06-2011, disponible en ligne : http://www.vogella.com

[29]- Lars Vogel, «Java Emitter Template(JET) ».Tutoriel, 01-02-2011, en ligne http://www.vogella.com/articles/EclipseJET

[30]- Lars Vogel, « Plugin Eclipse Editeur ». Tutoriel, 01-09-2011, en ligne http://www.vogella.com/articles/EclipsePlugIn

[31]- LAURENT AUDIBERT, « UML 2 - de l'apprentissage à la pratique » ELLIPSES, le 11/2009

[32]- Object Management Group « A UML Extension Profile for SoC ». Revision 1.0a, Date: 1 August 2004.

[33]- Cour en ligne SystemC. : http://comelec.enst.fr/hdl/sc_intro.html.

[34]- E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio. *A UML 2.0 profile for SystemC: Toward High-level SoC design.* [ed.] ACM. Jersey City, New jersey, USA : s.n., September 19-22 2005. 1-59593-091-4/05/0009.

[35]- N.Berrehouma, E.Bourennane, A.Chaoui, O.Labbani. *Visual Tool for SystemC functionnal MPSoC Design. To appear in Proceedings of the 24th Compteer Application in Industry and Engineering (CAINE'11), 16-17 November 2011, Hawaii, USA*

[36]- N. Berrehouma. *Rapport Sur L'état D'avancement De La These De Doctorat De Science En Informatique, Octobre 2010*

[37]-Hadded Sara, Benchaàbene Hadda « Environnement visuel pour la modélisation des systemes embarqués » mémoire Master 2, Univ-guelma, Juin 2011.

[38]- Pierre-Arnaud Marcelot. « Cours Plugin Eclipse », Master I, Université Paris VI / Parcours STL.