Université de Montréal

Nouvelles approches pour la conception d'outils CAO pour le domaine des systèmes embarqués

Par

James Lapalme

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences - Secteur des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de Doctorat
en Informatique

Février, 2009

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée :

Nouvelles approches pour la conception d'outils CAO pour le domaine des systèmes embarqués

présenté par :

James Lapalme

a été évaluée par un jury composé des personnes suivantes :

Dr. Houari Sahraoui
président-rapporteur

Dr. El Mostapha Aboulhamid
directeur de recherche

Dre. Gabriela Nicolescu
codirectrice de recherche

Dr. Claude Frasson
membre du jury

Dr. Ahmed Jerraya
examinateur externe

Dr. Thierry Bardini
représentant du doyen de la FES

Résumé

Notre société de consommation exerce des pressions énormes sur l'industrie des systèmes embarqués. Ces pressions sont causées en grande partie par des attentes souvent contradictoires et difficilement conciliables. Afin de répondre à la situation, l'industrie fait évoluer de manière fulgurante les techniques et les processus de fabrication des systèmes électroniques. Malgré cette évolution, un gouffre important sépare notre capacité d'intégrer des transistors et notre habileté pour la conception de systèmes les utilisant de façon efficace. Ce gouffre ne fait qu'exacerber les difficultés de l'industrie à répondre aux exigences du marché. Les outils CAO sont un moyen prometteur pour combler ce gouffre. L'efficacité de ces outils est un critère important pour leur réussite et peut être grandement influencée par les technologies et les approches de conception utilisées pour leur réalisation. La communauté du logiciel a développé plusieurs technologies et approches pouvant servir à l'élaboration des outils CAO.

Ce travail présente des idées innovatrices pour la conception et la réalisation d'outils CAO. Ces idées s'appuient sur l'utilisation de technologies modernes provenant de la communauté du logiciel. Les technologies principales qui seront abordées dans le cadre de ce travail sont la plateforme .Net, le Web Sémantique, la conception dirigée par les modèles et les patrons de conception. Ce travail présente aussi plusieurs réalisations afin d'appuyer les idées proposées.

Mots clés : système embarqué, .Net, Web Sémantique, CAO, conception dirigée par les modèles, Y-Chart

Summary

Our society of consumption exerts enormous pressures on the embedded systems industry. These pressures are mainly due to expectations which are often contradictory and not easily reconcilable. In order to cope with the situation, the industry has rapidly developed and evolved techniques and manufacturing processes for electronic systems. In spite of this evolution, an important gap exists between our capacity to integrate transistors and our systems design capabilities to use these transistors effectively. This gap does nothing but exacerbate the difficulties of industries to fulfill the expectations of the market. CAD tools are a promising means to cross this gap. The effectiveness of these tools is an important criterion for their success. Their effectiveness is largely influenced by the technologies and the design approaches used for their implementation. The software community has developed several technologies and approaches which can be greatly beneficial to CAD tools development. This thesis presents innovative ideas for the design and the realization of CAD tools. These ideas are based on the use of modern software technologies. Several accomplishments will also be presented which support these ideas. The main technologies that will be discussed within this work are the Net platform, the Semantic Web, model-driven design\development and design patterns.

Key words: embedded systems. Net, Semantic Web, CAD, model-driven design, Y-Chart

Table des matières

Liste des figures

Liste des abréviations

CAD	Computer Assisted Design
CAO	Conception assistée par ordinateur
CIL	Common Intermediate Language
CLI	Common Language Infrastructure
CLS	Common Language Specification
CTS	Common Type System
ECMA	European Computer Manufacturers Association
EDA	Electronic Design Automation
EDIF	Electronic Design Interchange Format
FIFO	First In, First Out
FPGA	Field-Programmable Gate Arrays
HDL	Hardware Description Language
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
ISO	International Organization for Standardization
JVM	Java Virtual Machine
NOC	Network On Chip
OOP	Object-Oriented Programming
OSCI	Open SystemC Initiative
PCB	Printed Circuit Board
RTL	Register Transfer Level
SCV	SystemC Verification Library
SDL	System Description Language
STOC	SpecC Technology Open Consortium
UCI	University of California Irvin
VES	Virtual Execution system
VHDL	Very High Speed Integration circuit Hardware Description Language
XML	Extensible Markup Language

Remerciements

Je voudrais remercier Dr. El Mostapha Aboulhamid et Dre. Gabriela Nicolescu pour l'aide et l'encadrement dont ils m'ont fourni durant mes études de doctorat. Je voudrais tout particulièrement remercier Mostapha pour sa grande générosité et son appui quasi paternel qui m'ont aidé lors des moments plus difficile de ce parcours.

Je voudrais remercier profondément les personnes proches de moi qui n'ont jamais arrêté de croire en moi; dont leur encouragement et leur support ont été cruciaux à la réussite de ce travail (thx Mom, Dad and MJ).

En terminant, je voudrais remercier sincèrement Steven, Christian et Marie-Josée pour avoir participé à la correction de cette thèse et/ou pour avoir écouté mon « blabla » interminable sur le contenu de celle-ci… :o)

I would like to dedicate this to my mother and father
who have always pushed me to better myself,
and to Zachary, Jacob and Marie-Josée, the three loves of my life,
without whom this would have no meaning

I would also like to dedicate this to my maternal grand-parents for their love

" I began this journey to understand the world around me.
I have learnt very little about it but have learnt a great deal about myself " – Anonymous

Introduction

Les systèmes embarqués (systèmes enfouis) sont fondamentaux à notre style de vie contemporain. Malgré leur omniprésence, la majorité de la population est ignorante du rôle important que joue ces petits systèmes cachés qui font fonctionner comme par magie, les nombreuses choses qui nous entourent. Juste en prenant une petite pause de la rédaction de ce texte, en regardant autour de moi, dans mon bureau, je peux percevoir rapidement une multitude d'items à l'apparence anodine contenant un système embarqué :

- imprimante
- routeur
- téléphone multifonctionnel
- thermostat électronique
- montre électronique
- téléphone cellulaire

Puisque le terme *système embarqué* est omniprésent dans ce texte et pour en faciliter la compréhension, voici une définition informelle permettant de mieux comprendre la portée et le sens du terme :

> *« Un système embarqué peut être défini comme un système électronique et informatique autonome, qui est dédié à une tâche bien précise. Ses ressources disponibles sont généralement limitées. Cette limitation est généralement d'ordre spatial (taille limitée) et énergétique (consommation restreinte). » (Wikipedia.)*

Les systèmes embarqués ont évolué grandement depuis leur apparition initiale. La figure 1 présente un des premiers systèmes embarqués à être produit en série durant les années 60: le système de guidage pour le missile Minuteman 1.

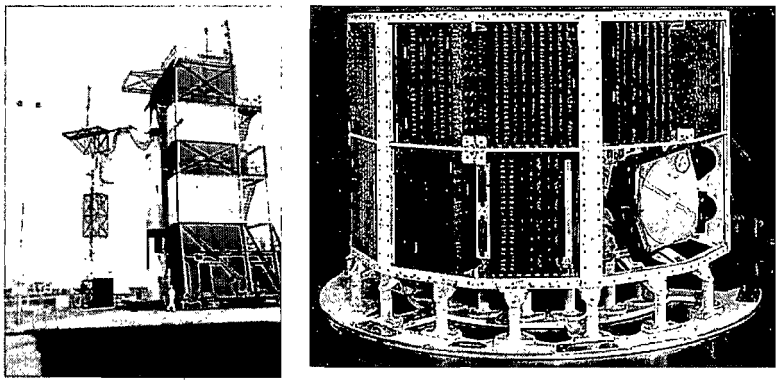


Figure 1 Système de guidage Autonetics D-17 du missile Minuteman I

De nos jours, les systèmes embarqués sont plusieurs ordres de grandeur plus petits et supportent un très grand nombre de fonctionnalités. La figure 2 (à gauche) présente le **©iPod touch** de Apple, un produit de consommation de masse très populaire, qui possède plusieurs fonctionnalités telles que le décodage vidéo, le décodage de musique, la communication réseautique sans-fils, etc. La figure 2 (à droite) illustre bien la taille de certains des plus petits systèmes embarqués pouvant être produits de nos jours : les **Smartdusts motes**. Les **Smartdusts motes** sont des systèmes micro-électro-mécaniques capables de communiquer via un réseau sans-fil. De plus, généralement, ils sont sensibles aux divers aspects de leur environnement : son, lumière, température, etc.

Figure 2 Systèmes embarqués modernes

Malgré l'évolution fulgurante des techniques et des processus de fabrication des systèmes électroniques (dont les systèmes embarqués), l'industrie de la microélectronique fait face à de grands enjeux afin de répondre aux attentes du marché. Notre société de consommation exerce des pressions énormes sur cette

industrie. Ces pressions sont causées en grande partie par des attentes souvent contradictoires ou difficilement conciliables. Le marché demande à la fois des produits avec de plus en plus de fonctionnalités, mais dont le coût d'achat est moindre et le temps de mise en marché plus court.

La loi de Moore [2], une loi fondée sur des extrapolations empiriques faites par Gordon Moore chez Intel, énonce « la quantité de transistors pouvant être intégrés par superficie sur une puce double approximative aux 2 ans ». Cette capacité d'intégration des transistors est due à l'amélioration continue des processus de fabrication. Cette capacité d'intégration croissante est un élément essentiel afin de pouvoir répondre aux besoins du marché. Néanmoins, elle n'est pas un critère suffisant.

Depuis plusieurs décennies maintenant, un fossé énorme se crée entre notre capacité d'intégrer des transistors et notre habileté pour la conception de systèmes les utilisant de façon efficace. La figure 3 illustre graphiquement ce fossé. La ligne noire représente la loi de Moore avec un niveau d'intégration doublant aux 36 mois. La ligne grise du bas représente l'évolution de la productivité de l'industrie en conception de système matériel. On peut percevoir un écart qui croit à un rythme exponentiel entre ces deux lignes. La figure 3 illustre aussi la croissance des besoins additionnels pour du logiciels spécifiques nécessaires au support des systèmes matériels. Cette croissance double aux 10 mois. Donc le vrai gouffre de productivité incluant les besoins de logiciel est énorme (la flèche rouge verticale).

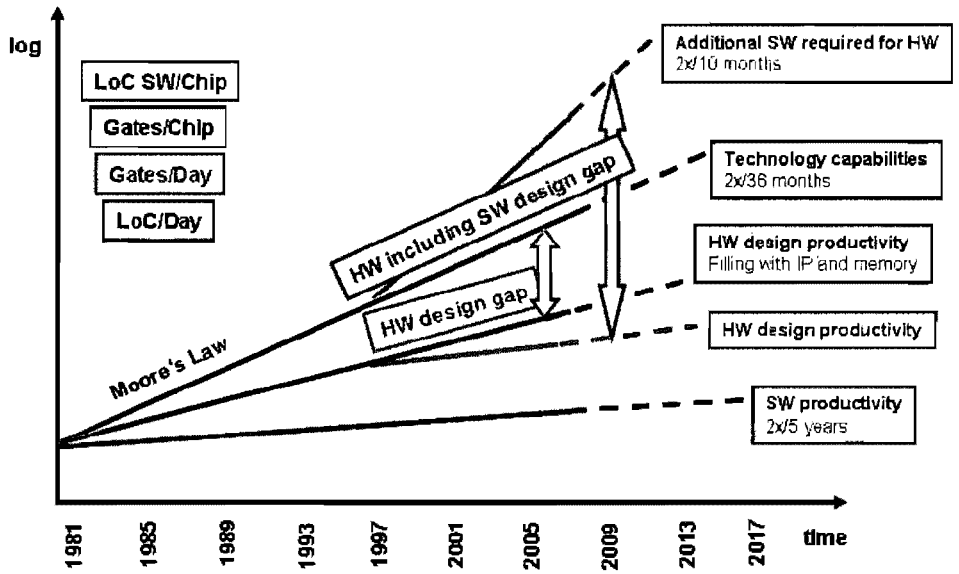


Figure 3 Gouffre de productivité[1]

Les causes freinant la productivité sont grandement attribuables à la complexité du processus de conception des systèmes modernes, et de leur validation. Les grands aspects devant être maîtrisés pour vaincre ce gouffre sont [40] :

- la réutilisation;
- la vérification et le test;
- l'optimisation de la conception de système guidée par les coûts;
- la conception de logiciel embarqué;
- la robustesse des plateformes matérielles d'implémentation;
- la gestion du processus de conception.

La maîtrise du processus de conception des systèmes modernes requiert un grand éventail de techniques, d'outils et de méthodologies qui permettent d'établir de manière prévisible une transformation des exigences en un produit fini.

1.1 La conception assistée par ordinateur

Le domaine de la conception assistée par ordinateur (CAO) a comme cheval de bataille les grands enjeux entourant la conception des systèmes modernes dont les

[1] Source ITRS 2007

systèmes embarqués. La CAO facilite la transformation d'exigences fonctionnelles en produits finis par l'utilisation d'outils informatisés supportant des techniques et des méthodologies de conception. Typiquement, cette assistance augmente de façon significative la productivité des concepteurs en offrant plusieurs fonctionnalités clés :

- la capture et la validation de besoins fonctionnelles;
- le raffinement des spécifications fonctionnelles en implémentation;
- l'exploration et l'analyse des implémentations alternatives;
- la vérification des implémentations.

Dans le passé, le domaine de la CAO a mis beaucoup l'accent sur la dimension outillage et a négligé celui de la méthodologie. Les outils CAO inflexibles au soutien de nouvelles méthodologies rendant ceux-ci mal adaptés et inefficaces sont une conséquence de cette négligence. Il est clair que la tendance des systèmes embarqués est de se complexifier en raison de leur capacité multifonctionnelle de plus en plus ciblée à un domaine précis, ainsi que leur conception incluant un nombre croissant de composantes hétérogènes (micromécanique, micro-optique, etc.). Afin de maintenir cette tendance, il est nécessaire d'avoir des méthodologies de conception éprouvées, efficaces et adaptées pour guider les concepteurs durant le cycle complet de mise en marché – du concept au produit fini testé. Les prochaines générations d'outils CAO se doivent d'être plus **flexibles** afin de répondre aux besoins **méthodologiques**.

1.2 Les technologies modernes du logiciel

Parallèlement et indépendamment de l'industrie des systèmes embarqués, les domaines du logiciel et du génie logiciel ont évolué énormément au cours des 20 dernières années. Cette évolution est due aux mêmes facteurs de pression que subit l'industrie des systèmes embarqués : plus de fonctionnalités livrées à moindre coûts et en moins de temps. Afin de faciliter la création de logiciels de meilleure qualité et d'augmenter la productivité des développeurs, la communauté du logiciel a mis au point plusieurs technologies innovatrices. Quatre de ces grandes technologies sont :

- les langages et les plateformes de programmation modernes tel que C# et .Net [1][28];
- le développement à base de patron de conception [31];
- le développement dirigé par les modèles [45][11];

- les technologies du Web sémantiques [10].

En plus de permettre le développement de logiciels de manière plus efficace, ces technologies permettent de produire de meilleurs logiciels étant souvent plus flexibles et adaptables. Puisque le domaine de la CAO est à la base un domaine axé sur le logiciel, ce domaine peut bénéficier grandement de l'utilisation des technologies mentionnées plus haut afin d'améliorer leur conception et leur degré de flexibilité. Donc l'utilisation du savoir faire du domaine du logiciel peut amener des solutions intéressantes aux problématiques de productivité du domaine des systèmes embarqués.

1.3 Motivations

Les méthodologies de conceptions, les langages de description systèmes ainsi que les environnements de conception implémentant ces deux éléments sont des outils importants pour la conception des systèmes embarqués (ces éléments seront présentés davantage dans le prochain chapitre). Malgré les efforts investis pour l'avancement de ceux-ci, il n'existe pas encore de solutions parfaites. Plusieurs difficultés persistent :

- les environnements de conception ne supportent pas adéquatement les méthodologies; plusieurs discontinuités sont présentes.
- les environnements de conception permettent difficilement l'intégration d'outils tierce-partie afin de supporter des flux de conception personnalisés.
- les langages hôtes utilisés pour les langages de description systèmes indépendants (voir chapitre 2) ajoutent souvent inutilement des complexités rendant leur utilisation et leur personnalisation difficiles.
- les langages de description système orientés bibliothèque (voir chapitre 2) supportent difficilement la conception par blocs de propriété intellectuelle (IP) (voir chapitre 2) en raison d'un manque de séparation entre les aspects de modélisation et de simulation.
- les environnements de conception ne facilitent pas adéquatement la conception par IP; la consommation, le partage et l'analyse des informations décrivant les systèmes et les IP sont souvent difficiles.

Ce travail cible ces enjeux et propose des solutions utilisant les technologies modernes du logiciel. Plus précisément, ce travail s'attarde particulièrement à :

- la conception des langages de description orientés bibliothèque afin de faciliter leur utilisation, leur personnalisation et l'intégration d'outils tierce-partie
- la séparation des aspects de modélisation et simulation chez les langages de description orientés bibliothèque afin de faciliter la conception par IP et l'exploration de partitionnement.
- la gestion des informations décrivant les systèmes et les IP basée sur les technologies sémantiques afin de faciliter la consommation, le partage et l'analyse de celles-ci.

De plus, ce travail porte une attention particulière à l'implémentation du paradigme « Y-Chart » (voir chapitre 2) comme méthodologie pour la conception des systèmes puisque celui-ci propose un flux de conception complet qui ressemble beaucoup à la conception dirigée par les modèles utilisés par la communauté du logiciel.

1.4 Contribution et l'organisation

Les principales contributions de cette thèse sont les suivantes :

- la présentation du lien entre le développement orienté modèle et le paradigme du « Y-Chart ». Une discussion sur ce paradigme est faite, ainsi qu'une comparaison de plusieurs implémentations de celui-ci (chapitre 3).
- la définition d'une nouvelle méthodologie pour la conception des outils CAO basée sur les technologies .Net et une vision orientée modèle (chapitre 4);
- l'élaboration d'une plateforme nommée Esys.Net pour la modélisation et la simulation de systèmes qui utilise cette nouvelle méthodologie (chapitre 4);
- la définition d'une nouvelle architecture cible pour les plateformes de type « Framework » pour la modélisation et la simulation de systèmes matériel/logiciel (chapitre 5);
- l'élaboration d'une plateforme nommée SoCML implémentant cette architecture cible (chapitre 5);
- la définition d'une approche innovatrice utilisant les technologies du « Semantic Web » pour palier au problème de l'utilisation des technologies XML dans le demain de la CAO (chapitre 6);

- la présentation d'une étude de cas basée sur le standard IP-XACT pour l'application des technologies sémantiques (chapitre 6);

Un ensemble précis de technologies logicielles a été crucial pour la réalisation de ces contributions. Ces technologies seront présentées, discutées et appliquées dans le contexte de la CAO et des systèmes embarqués, elles sont :

- le langage C# et la plateforme .Net;
- les patrons de conception avancés tel que le « Proxy » et le « Dependency Injection »;
- le développement dirigé par les modèles;
- les technologies du Web sémantiques.

La forme de cette thèse est « par articles ». Le noyau de son contenu est constitué de quatre articles. Le chapitre 2 présente une revue de la littérature sur les outils de modélisation et de simulation des systèmes embarqués. Ce chapitre présente aussi trois méthodologies de conception de systèmes couramment utilisées. Le chapitre 2 résume aussi des revues déjà présentes dans les articles. Le contenu des chapitres 3 à 6 est essentiellement celui des articles. Le chapitre 7 est une conclusion plus globale que celles se trouvant dans chacun des articles.

Revue de la littérature

La réalité qui nous entoure surpasse de loin notre capacité d'analyse et de synthèse. Pour faire face à cette insuffisance, nous utilisons les outils d'abstraction et de simplification pour créer des modèles de notre réalité qui sont compréhensibles et manipulables. Les systèmes embarqués sont très complexes. Ils comportent une énorme quantité de détails dont ceux-ci se manifestent sur plusieurs échelles de grandeurs – des effets de leurs fonctions perceptibles à notre niveau jusqu'aux phénomènes atomiques de leur implémentation. Donc, le domaine des systèmes embarqués ne peut être abordé sans faire des abstractions et des simplifications.

La modélisation de systèmes matériels complexes discrets ainsi que des langages pour exprimer ces modèles sont en utilisation depuis près de 40 ans [21][13]. Ces langages portent le nom de « hardware description language » (HDL). Avec la standardisation en 1987 par l'IEEE du langage VHDL, l'industrie a rapidement adopté les HDL comme des outils essentiels. Depuis l'apparition initiale des premiers HDL, les systèmes matériels ont évolué vers des formes plus complexes combinant le matériel et le logiciel : les systèmes embarqués. Afin de permettre aux concepteurs de modéliser ces nouveaux systèmes, les HDLs ont évolué aussi vers des langages appelés « system description languages » (SDL).

De concert avec le développement des HDL/SDL, des méthodologies de conception ont été élaborées. Ces méthodologies définissent des paradigmes de conception et/ou des cadres généraux d'activités pour guider le cycle de vie d'élaboration des systèmes complexes. Les trois méthodologies les plus utilisées aujourd'hui sont le raffinement successif, le paradigme « Y-Chart » et la conception à base de IP.

1.5 Les SDL

Il existe deux grandes familles de HDL/SDL soit les indépendants et les orientées bibliothèque. Les HDL/SDL indépendants sont caractérisés par le fait qu'ils ont des syntaxes, des compilateurs et des analyseurs qui leur sont propres. Contrairement à ceux-ci, les HDL/SDL orientés bibliothèques sont définis au moyen de bibliothèques de code implémenté avec un langage de programmation général déjà existant tel que C++, C et Java. Chaque approche a ses avantages et ses inconvénients.

1.5.1 Les HDL/SDL indépendants

Cette première famille de HDL/SDL est composée de langages conçus uniquement pour la description de systèmes matériels et/ou embarqués. La grande majorité de ces langages ont été conçus par l'industrie pour l'industrie. Certains de ceux-ci s'inspirent de proche ou de loin aux langages programmation déjà existants.

1.5.1.1 VHDL [39]

Le développement de VHDL a été initié en 1981 par le « United States Department of Defence » afin d'adresser la crise du cycle de vie des systèmes matériels. VHDL est fortement inspiré du langage de programmation Ada. Les principaux objectifs de VHDL sont :

- définir une notation unifiée pour la description de systèmes électroniques modélisés à divers niveaux d'abstraction;
- être lisible par l'homme et les systèmes informatisés;
- supporter la communication de métadonnées reliées au processus de conception;
- faciliter la maintenance, la modification et l'acquisition de composantes matérielles;
- supporter le développement, la vérification, la synthèse et validation de description matérielle via un langage agnostique des outils.

VHDL est un langage dédié pour la description des systèmes matériels. Plusieurs des concepts nécessaires pour la description adéquate de composantes logiciels sont absentes. Malgré ce fait, VHDL est un langage très bien adapté pour la description de systèmes matériels aux niveaux d'abstraction des portes logiques et des transferts inter registres (RTL). Présentement, plus d'une quinzaine de standards IEEE sont

reliés au VHDL tel que VHDL-AMS [38] pour la description des systèmes analogiques.

1.5.1.2 Verilog [37]

Verilog était le principal compétiteur à VHDL avant l'arrivée de SystemC en 1999. Malgré son apparition initiale en 1985, le langage est devenu un standard IEEE seulement en 1995. Verilog a l'avantage d'avoir une syntaxe beaucoup moins verbeuse que celle de VHDL au détriment d'une expressivité moindre. De plus, la simulation de modèle Verilog est généralement beaucoup plus rapide. Depuis quelques années, VHDL et Verilog convergent de plus en plus offrant ainsi des capacités similaires [5].

1.5.1.3 SystemVerilog [65]

SystemVerilog est une extension de Verilog. Il a été proposé par le consortium Accellera en 2002 et standardisé par IEEE en 2005. SystemVerilog étend Verilog avec des éléments nécessaires pour la modélisation et la vérification de systèmes embarqués. SystemVerilog offre une expressivité très riche en matière de vérification et de validation. Cette expressivité est grandement due à l'intégration du langage OpenVera dans le standard. [6] fait la comparaison entre VHDL, Verilog et SystemVerilog. Par ce travail, on peut percevoir que SystemVerilog est un quasi surensemble de VHDL et Verilog.

1.5.1.4 Handel-C et OCAPI [14]

Quelques articles discutent des avantages à utiliser des HDLs basés sur des langages de programmation existants [16]. OCAPI est un langage basé sur le C++. Il est très efficace pour effectuer de l'exploration à un niveau d'abstraction système. Handel-C est un langage basé sur le C. Il permet la traduction de ceux-ci vers EDIF et VHDL pour l'exécution sur FPGAs. Ces deux technologies ont la particularité qu'elles s'intègrent facilement ensemble pour offrir un environnent capable de supporter un processus de conception de systèmes ciblé pour les FPGA.

1.5.1.5 SpecC [68]

SpecC a été développé en 1997 à l'University of California Irvine. Le langage SpecC est une extension du langage de programmation ANSI-C. SpecC augmente C avec des concepts essentiels pour la conception de systèmes. Il offre une notation formelle destinée aux spécifications et à la conception des systèmes embarqués. En 1999, le SpecC Technology Open Consortium (STOC) a été fondé pour guider l'évolution de SpecC. En décembre 2002, la deuxième génération du langage, SpecC 2.0, a été approuvé par le STOC. SpecC est l'un des rares langages à supporter la spécification explicite d'éléments comportementaux hiérarchiques. De plus, d'autres langages comme SystemC ont été fortement influencés par les concepts de canal de communication, ainsi que les divers niveaux d'abstraction de ceux-ci développés dans SpecC. SpecC a été conçu pour bien supporter par une méthodologie de conception de système basé sur l'assemblage d'IP.

1.5.2 HDL/SDL orientés bibliothèques

La deuxième famille de HDLS/SDLs est celle dont les langages sont définis par l'entremise d'une bibliothèque de code développé avec un langage de programmation déjà existant tel que C++, C et Java. Via une bibliothèque de code, ces langages ajoutent les concepts nécessaires pour la conception de systèmes embarqués. Les concepts typiquement ajoutés sont :

- comportement concurrent;
- notion du temps;
- élément pour la modélisation d'élément de communication.

Pour l'élaboration de ce type de langage, ce sont souvent les langages orienté objet qui sont utilisés comme langage hôte pour les bibliothèques.

1.5.2.1 JHDL [9]

JHDL est un environnement orienté objet qui exploite exclusivement les capacités du langage de programmation Java. JHDL permet la modélisation, la simulation et l'implémentation efficace sur FPGA de systèmes matériels définis au niveau RTL. Le principal objectif de JHDL est de développer un environnement pour la CAO

exploratoire pour l'identification des dispositifs principaux et les fonctionnalités clés que celui-ci doit offrir pour le développement de systèmes ciblé pour les FPGA.

1.5.2.2 SystemC [64]

SystemC, annoncé en 1999 par The Open SystemC Initiative (OSCI), a été reçu avec beaucoup d'enthousiasme de la part de l'industrie et de la communauté académique. SystemC a été le premier SDL orienté bibliothèque basé sur le langage C++. En 2003, l'OSCI a annoncé le SystemC Verification Library (SCV) [27], une bibliothèque ajoutant des fonctionnalités de vérification et d'introspection. En 2005, SystemC a été standardisé par l'IEEE. À l'heure actuelle, c'est un langage très populaire pour la conception des systèmes embarqués. SystemC offre tous les concepts qu'on retrouve typiquement dans les HDLs traditionnels tel que VHDL et Verilog. Il offre aussi un ensemble de concepts supplémentaires permettant la spécification de systèmes à divers niveau d'abstraction. Toutefois, la version courante de SystemC n'offre pas tous les concepts nécessaires pour la modélisation de composantes logicielles. Il est possible d'utiliser les capacités intrinsèques du langage C++, mais le standard ne définit pas formellement comment ses éléments devraient être pris en considération par la bibliothèque. Une étude fait par Doulos [26] illustre que la majorité des compagnies utilisant SystemC font des études de performance, de l'exploration d'architecture, de la modélisation au niveau transactionnel et de la conception logiciel-matériel. L'étude illustre aussi que ces compagnies utilisent généralement des HDL standards comme VHDL pour la modélisation de systèmes matériels et pour la synthèse.

1.6 Les méthodologies

Les méthodologies de conception sont un élément essentiel pour la réalisation de systèmes. Elles définissent un cadre de travail permettant de guider de manière répétitive et prévisible le processus d'élaboration d'un système. Généralement, ces cadres de travail comportent deux composantes : une liste d'artéfacts de travail et un ensemble de flux d'activités. Les artéfacts sont des intrants ou des extrants aux activités du cadre de travail. Le type d'artéfacts utilisés, le processus de réalisation de ceux-ci, ainsi que leur importance dans le processus global d'élaboration sont des éléments différenciant les diverses méthodologies sur le marché. Les trois méthodologies les plus présentes sont :

- l'élaboration par raffinements successifs;
- le paradigm « Y-Chart »;
- la conception à base d'IP.

La plus ancienne mais encore la plus répandue est l'élaboration par raffinements successifs. Le rôle des flux d'activités est d'organiser les différentes catégories d'activités nécessaires pour la conception d'un système afin d'avoir une recette répétable. Les catégories d'activités les plus typiques sont la modélisation, la vérification, la validation, l'analyse de performance et la synthèse. Chaque méthodologie organise différemment les activités.

1.6.1 L'élaboration par raffinements successifs

Au plus simple, l'élaboration par raffinements successifs consiste à créer le modèle d'un système à un haut niveau de détails qui est par la suite détaillé par raffinements successifs afin de produire une description du système pouvant être implantée [71]. Cette approche est orientée « top-down » ou de « haut vers le bas ». Typiquement, le premier modèle du système est très orienté fonctionnel. Celui-ci abstrait tous les aspects d'implémentation afin de cibler uniquement les exigences, les fonctions et l'interaction entre celles-ci. Ce modèle est ensuite vérifié pour valider sa complétude et sa cohérence. Il est finalement raffiné pour obtenir de nouveaux modèles comportant des détails d'implémentation de plus en plus précis. L'objectif de chaque raffinement est d'explorer divers choix de conceptions pour évaluer leurs impacts

(performance, puissance, etc.). Généralement, chaque itération de raffinement comporte un ensemble de tâches réalisées plus ou moins en série :

- raffinement du modèle afin de créer un modèle plus détaillé;
- validation de l'alignement entre le nouveau modèle et les exigences afin de ne pas introduire d'erreur;
- analyse des performances (temps d'exécution, puissant, etc.) du système défini. Cette analyse est soit faite avec des méthodes d'analyse statiques (model checking) ou dynamiques (simulation);
- analyse des résultats;
- définition de la stratégie pour la prochaine itération de raffinement.

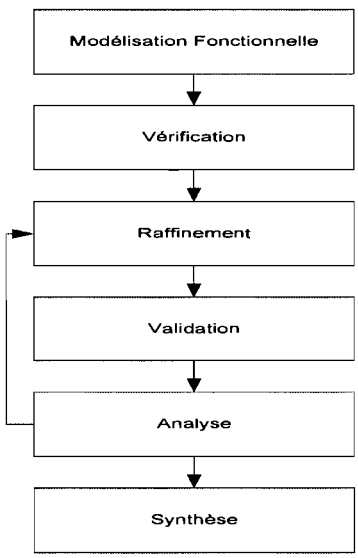


Figure 4 Raffinements Successifs

La figure 4 illustre le flux d'activités de la méthodologie. Les diverses implémentations de cette méthodologie identifient deux axes de raffinement :

- calcul;
- communication.

Les raffinements sur l'axe du calcul consistent à ajouter des détails tels que le temps nécessaire pour faire les calculs, la répartition des fonctions en logiciel ou matériel, la gestion du partage de ressources de calcul, etc. Les raffinements sur l'axe de la communication consistent à ajouter des détails tels que le temps nécessaire pour faire les communications, la gestion du partage des ressources de communication, les protocoles de communication, la médiation entre les divers protocoles de

communication, etc. Dans [24] un raffinement selon des axes est présenté ainsi qu'une taxonomie de modèles ciblant des niveaux de détails précis sur ces axes. [30] présente une autre taxonomie de modèles ainsi que leur position dans un flux de travail.

Les grandes difficultés de cette approche sont d'explorer rapidement des solutions alternatives de raffinement et de garantir que ces raffinements n'introduisent pas d'erreurs. La première problématique est due au fait que généralement les modèles sont biaisés par une certaine conception générale du système. Ce biais est souvent introduit aussitôt que la modélisation initiale. Chaque raffinement introduit de nouveaux biais. Ces biais sont quasiment inévitables, en raison du caractère « top-down » de la méthodologie. Puisque le dernier raffinement fait doit aboutir une spécification d'implémentation avec une architecture bien définie, il est sans contre dit que chaque raffinement amènera le modèle à posséder des caractéristiques de cette architecture finale. Ce biais rend difficile l'exploration de raffinement pouvant aboutir à des architectures très différentes. Chaque raffinement contraint donc les possibles explorations.

La deuxième problématique est due au fait que chaque raffinement nécessite de retoucher au modèle afin d'introduire des nouveaux détails. Les raffinements peuvent consister à ajouter que de simples annotations liées au temps ou la consommation énergétique jusqu'à des modifications majeures nécessitant la réécriture/substitution de certaines portions du modèle. Ce dernier cas est typique lors du passage d'un niveau d'abstraction à un autre tel que le passage d'une modélisation au niveau d'un protocole de communication à un niveau de détails RTL. Donc, chaque raffinement fait que le modèle peut introduire des erreurs qui sont parfois difficiles à détecter tôt. SystemC et SpeC sont des environnements bien adaptés à cette méthodologie.

1.6.2 La conception à base d'IP

À l'opposée de la méthodologie par raffinements successifs, la conception à base d'IP utilise une approche « bottom-up » [55]. L'idée générale est de concevoir un système par l'assemblage d'un ensemble de composantes existantes offrant des fonctionnalités et des interfaces d'intégration standardisées. On peut facilement faire le parallèle avec le jeu de bloc ©Lego. Chaque bloc ©Lego à une forme précise et une interface standardisée qui permettent de le connecter à d'autres blocs ©Lego. Donc, je peux facilement construire une entité aussi complexe fonctionnellement qu'un château avec un ensemble de composantes très simples. Le développement de cette méthodologie a grandement été poussé par le besoin de concevoir rapidement des systèmes complexes pour répondre à des besoins du marché. Puisque typiquement, chaque bloc a été préalablement testé, analysé et synthétisé, il est généralement facile de concevoir un système valide et synthétisable avec les caractéristiques de performance désirées.

Malgré la simplicité théorique de cette méthodologie, l'implémentation de celle-ci n'est pas triviale en raison de plusieurs difficultés pragmatiques. Le support informatisé d'une méthodologie par des outils pour la CAO est important. Il y a plusieurs outils commerciaux sur le marché supportant l'approche, mais il n'y a pas de standard non-propriétaire très répandu pour la description d'IP afin que ceux-ci soit supportés par les outils. Chaque fabriquant doit en pratique prendre la décision de rendre son IP disponible à un sous-ensemble d'outils sur le marché. De plus, il est souvent difficile d'offrir une représentation suffisante d'un IP pour supporter le processus de conception (simulation, test, synthèse), mais dont le détail de la représentation ne dévoile pas les secrets de conception propriétaire du IP. Une autre grande difficulté est qu'il n'existe pas de standard unique d'interface pour les composantes, donc pour des raisons purement pratiques, les fabricants doivent se limiter à supporter un nombre limité d'interface. Donc l'ensemble de ces difficultés fait en sorte qu'il est difficile d'avoir un ensemble de composantes couvrant l'ensemble de nos besoins pouvant être facilement interconnectés et dont le système total peut être facilement testé et synthétisé.

1.6.3 Le paradigme « Y-Chart »

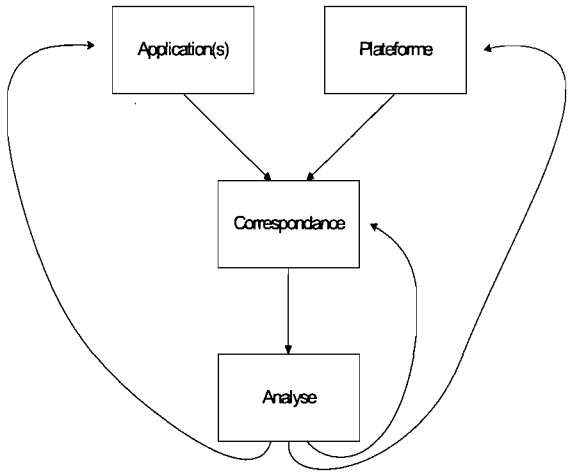


Figure 5 Y-Chart

Le paradigme « Y-Chart » est caractérisé par un flux d'activité débutant avec deux branches indépendantes qui se rejoignent pour se terminer avec une série d'activités [43]. Le tout forme un **Y** comme illustre la figure 5 d'où le nom « Y-Chart ». Cette méthodologie est basée sur l'idée qu'il est important de séparer le problème de conception de systèmes en deux sous problèmes : (i) définir les fonctions d'un système de matière agnostique d'une implémentation (l'application) et (ii) définir une plateforme (matériel et logiciel) offrant un ensemble de ressources de calcul, de communication et de stockage pouvant supporter les besoins fonctionnels. Les exigences de l'application sont typiquement définies au moyen d'éléments de calcul communiquant ensemble via des éléments de communication. Généralement, ce type de définition permet d'expliciter le parallélisme pouvant être exploité dans l'application. Il est difficile d'être plus précis, car chaque implémentation du paradigme utilise son propre formalisme pour la portion application et plateforme, donc nous essayons juste de peindre un portrait général de l'approche.

Une fois ces deux sous problèmes résolus, une correspondance est définie entre les éléments de l'application et les ressources de la plateforme. Cette correspondance établie une configuration entre la ressource de la plateforme utilisée pour implémenter un certain élément applicatif. Une fois la correspondance faite, il est possible d'évaluer les performances (temps d'exécution, puissance, etc.) de

l'implémentation des besoins sur la plateforme selon la configuration établie. Selon les résultats obtenus, le concepteur peut revoir sa correspondance, la conception de la plateforme et la définition des besoins. Il peut boucler dans ce flux aussi longtemps qu'il n'a pas obtenu les résultats voulus.

Les grandes difficultés de cette méthodologie sont la séparation d'un problème en deux modèles agnostiques l'un de l'autre et la mise en correspondance des deux modèles. Ces deux problèmes sont intrinsèquement reliés, car généralement plus l'éloignement des deux modèles est grand plus la mise en correspondance est difficile.

Il n'est pas rare de voir cette méthodologie utilisée avec les deux autres et cela surtout pour la portion plateforme. Originalement, le paradigme « Y-Chart » a été conçu pour adresser la problématique de l'implémentation d'un besoin fonctionnel sur une plateforme existante dont celle-ci pouvait accommoder les besoins de diverses manières. Cette méthodologie permettait donc de faciliter l'exploration de ces diverses implémentations afin de trouver la meilleure. Pour cette raison, les compagnies ciblant une stratégie de réutilisation de plateformes flexibles et puissantes pour implémenter plusieurs produits utilisent souvent cette approche. On appelle cette approche de la conception orienté plateforme ou « plateform-based design» [18]. La définition de cette plateforme peut être le produit d'une conception à base d'IP ou par raffinements successifs. Métropolis [8] est un exemple d'environnement utilisant cette méthodologie.

1.7 Les nouveaux défis

Une autre révolution importante dans le domaine des circuits intégrés est l'intégration d'éléments non-microélectroniques sur silicium tel que les composants micro-optiques et micromécaniques. La conception de systèmes complexes et hétérogènes incluant ces composantes est caractérisée par un ensemble de nouveaux problèmes devant être résolus au niveau de la modélisation et de la vérification.

Le paradigme « Y-Chart » : une discussion sur le développement dirigé par les modèles

Malgré sa grande complexité si on s'attarde aux détails, le processus de développement d'un logiciel ou d'un système embarqué peut être résumé par deux grandes étapes :

- comprendre et décrire le besoin (le quoi);
- implanter la solution (le comment).

La communauté du logiciel s'intéresse à ces deux étapes ainsi que le passage de l'un à l'autre depuis plusieurs années. Dans les années 70, des techniques structurées pour aborder ces sujets ont vu le jour afin de pallier aux nombreux problèmes reliés aux approches précédentes généralement très ad hoc [23]. Au fil des années, influencées par l'émergence de nouveaux paradigmes tel que l'orienté objet, les techniques structurées ont donné place aux techniques d'analyse et de conception orienté objet [12]. Malgré les énormes efforts investis pour avancer la science de l'élaboration des logiciels, le passage du « quoi » au « comment » reste toujours très difficile. Une cause principale de cette difficulté est que le passage du « quoi » au « comment » se traduit souvent par une intervention humaine qui fait l'élaboration d'artéfacts pour implémenter le « comment » basé sur sa compréhension du « quoi ». Ce type d'intervention est souvent long et sujet à l'erreur.

Au début des années 80, une nouvelle approche a été proposée pour rapprocher le « quoi » du « comment », la méthode d'analyse orienté objet et de conception récursive Shaler-Mellor [66]. Cette approche consiste à modéliser le « quoi » de manière assez précise qu'il soit possible d'utiliser une approche par traduction automatisée pour générer le « comment » final. Cette approche préconise aussi, basée sur le principal de l'orthogonalité des aspects, une modélisation séparée des aspects

purement logiciel des aspects de contraintes architecturales, ainsi que la spécification des « ponts » afin de faire le lien entre les deux aspects. Cette approche a été raffinée dans les années 90 et 2000 afin de donner lieux à la conception dirigée par les modèles [11] et l'architecture dirigée par les modèles [45]

Le paradigme « Y-Chart » développé en 1997 peut être clairement classifié comme une approche dirigée par les modèles au même titre que la méthode Shaler-Mellor Quoi que le travail original de [43] ne fait mention que de l'approche orienté objet et non le travail de [66], il est difficile de croire que l'approche n'a pas été influencée par ce dernier ou du moins par les principes véhiculés dans le milieu du logiciel grâce à celui-ci au début de années 90.

L'article qui suit présente le paradigme « Y-Chart », un exemple d'une méthodologie pour la conception de systèmes embarqués dirigé par les modèles. Une grande portion de l'article est consacrée à la présentation/discussion de diverses approches pour l'implémentation de cette méthodologie, un sujets qui n'ont pas beaucoup été traités dans la littérature. Cet article peut-être perçu comme une extension de l'état de l'art de cette thèse.

Les contributions principales de cet article sont:

- une présentation à caractère pédagogique de la méthodologie « Y-Chart », des modèles de calcul et des modèles d'architecture;
- une présentation de trois implémentations de la méthodologie « Y-Chart » afin de faire la comparaison de leurs choix de conceptions.

Les pages suivantes contiennent une copie de l'article [50], dans son format original (sauf la numérotation des pages), soumis à *ACM Transactions on Embedded Computing Systems*.

Y-Chart Based System Design: A Discussion on Approaches

JAMES LAPALME
University of Montréal, Department of Computer Science and Operations Research, Canada
and
BART THEELEN
Eindhoven University of Technology, Department of Electrical Engineering, Netherlands
and
NIKOLAY STOIMENOV
ETH Zürich, Computer Engineering and Networks Laboratory, Switzerland
and
JEROEN VOETEN
Eindhoven University of Technology, Department of Electrical Engineering; Embedded Systems Institute; Netherlands
and
LOTHAR THIELE
ETH Zürich, Computer Engineering and Networks Laboratory, Switzerland
and
EL MOSTAPHA ABOULHAMID
University of Montréal, Department of Computer Science and Operations Research; Canada

Embedded systems are a source of technology that facilitates our modern lifestyle. In order to do so, they tend to increase in complexity as well as integrate in are day-to-day activities. To meet the market's expectations on technological improvement, time-to-market objectives for introducing innovative embedded systems are shorter than ever. Over the last decade, model-based design has been a subject of great interest as a means to accelerate the design of embedded systems. The Y-chart paradigm is a principal approach to model-based embedded system design. Despite the simplicity and conciseness of this paradigm, it has been implemented in several different ways by various methodologies. This variety in implementation designs is due to the particular emphasis a methodology puts on the different steps of the paradigm (application modeling, platform modeling, mapping, analysis and synthesis). This article explores this variety by examining and comparing three Y-chart based design methodologies: Metropolis, the Distributed Operation Layer incorporating Modular Performance Analysis and the Y-chart variant of the Software/Hardware Engineering methodology. These methodologies have been chosen because they: cover a broad domain of applications, have been developed on a relatively long period of time and are representative of typical Y chart approaches. Moreover, these implementations of the paradigm present interesting design approaches which are worth comparing.

This article (i) presents the concepts underlying the Y-chart paradigm as well as models of computation and models of architecture, (ii) discusses the three mentioned implementations, and (iii) compares these implementations to highlight important design differences. The examination and comparison show that the Y-chart paradigm is a very flexible framework that can be implemented in many ways.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: Modeling of Computer Architecture, System Architectures, Systems Specification Methodology; F.1.1 [**Computation by Abstract Devices**]: Models of Computation; B.1.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling Methodologies*

General Terms: Design, Languages, Measurement, Performance, Verification

Additional Key Words and Phrases: Design-Space Exploration, Embedded Systems, Model of Architecture, Model of Computation, Y-Chart

1. INTRODUCTION

The decreasing time-to-market for modern embedded systems encourages the industry to strive for design reuse between multiple products. Moreover, the constantly increasing consumer demands for full featured systems requires the use of high-performance platforms. In order to cope, systems designers often consider platforms such as Multi-Processor Systems-on-Chip (MPSoC) because they allow reuse and offer high-performance. MPSoC include a combination of multiple processors and specialized processing elements to allow the execution of multiple applications in parallel. They also offer the flexibility to optimise aspects such as performance and energy consumption. When elaborating a system, designers face some crucial questions:

—*Which platform is most suitable for realizing the requested functionality?*

—*How to exploit the parallelism provided by the chosen platform efficiently?*

—*Is the application software parallelized in a suitable way?*

Figure 1 illustrates how the Y-chart paradigm introduced in [Kienhuis et al. 1997] provides a framework for answering these questions. This work should must not

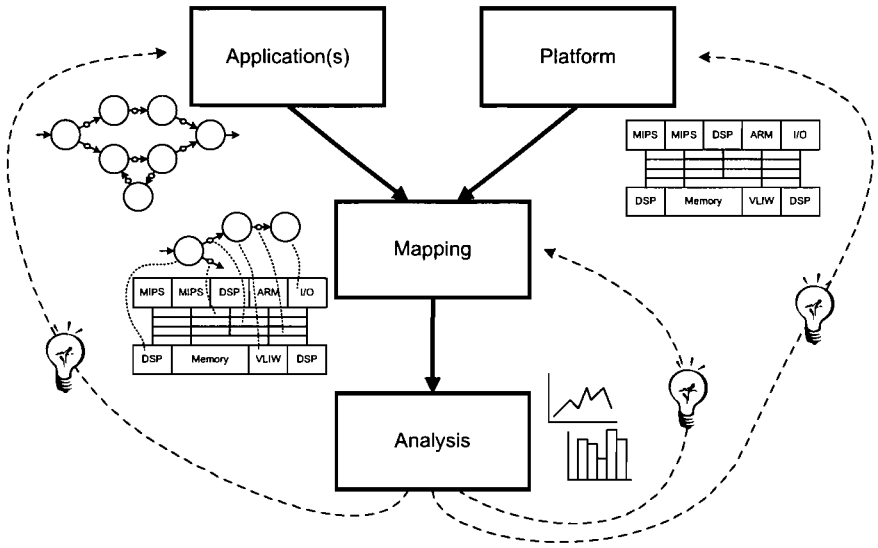


Fig. 1. Design-space exploration with the Y-chart paradigm

be confused with the Gajski and Kuhn Y-chart [Gajski and Kuhn 1983] which is a taxonomy for visualizing design views as well as design hierarchies. Modern embedded systems such as mobile phones support multiple *applications* such as MP3 players and Internet browsers. These applications are targeted to execute on a collection of processors (and specialized processing elements) that make up a *platform* [Martin 1998; H. Chang and Todd 1999; Sangiovanni-Vincentelli and Martin 2001]. By defining a *mapping* to specify which processor (or specialized processing elements) executes what functions of each application (allocation and binding) and at which moments in time (scheduling), one can predict the resulting performance of the overall solution design. The analysis of these results may give hints for improving the application software, the platform design and/or the mapping. The Y-chart paradigm envisions to iteratively apply such improvements until finding a solution that satisfies the end requirements. The final solution is then implemented to produce the desired product. The Y-chart paradigm was initially introduced by Kienhuis et al. [Kienhuis et al. 1997] in the context of dataflow-oriented systems. It was also independently applied in the POLIS project [Balarin 1997] for control dominated systems (e.g. automotive). Recently, it has also been used in others control-oriented domains [Baleani et al. 2000; Sgroi et al. 2001; Zeng et al. 2006]. The work introduced by Kienhuis consisted of Stream-Based Functions for describing applications, a C++ inspired Pamela library for specifying platforms, and a simulator called ORAS to perform the analysis part [Kienhuis 1999].

Although the key concept underlying the Y-chart paradigm of explicitly separating application descriptions from a platform specification is relatively simple, methodologies like POLIS, Metropolis [Balarin et al. 2003], SPADE [Lieverse et al. 2001], and Cadence's Virtual Component Co-Design Environment [Krolikoski et al. 1999] use very different implementations of the paradigm. This article discusses the

Y-chart paradigm for design-space exploration of distributed embedded systems by investigating alternative approaches to implement it. To this end, we focus on alternative formalisms for specifying applications and platforms as well as different views on how a mapping can be described. In addition, we elaborate on the possible analysis and design-space exploration capabilities that various Y-chart based design methodologies support and how they assist a designer in finding a suitable solution for a design problem. We have chosen to write this paper because there are few papers which compare various design approaches for the implementation of the Y-Chart paradigm.

The objective of this paper is not to offer the reader an in-deep comparison of various implementations in order to determine ranking. The objective is rather to present and discuss some interesting design approaches the authors have observed for implementing the different steps of the Y-Chart paradigm. These approaches will be compared in order to highlight capabilities, limitations and trade-offs. The authors would also like to stress that great emphasis will be put on design and not implementation of these approaches, hence issues such as performance and scalability will not be greatly discussed. The three specific implementations of the Y-Chart paradigm used in this paper are Metropolis, the Distributed Operation Layer incorporating Modular Performance Analysis and the Y-chart variant of the Software/Hardware Engineering methodology. The paper also presents the history and key underpinning concepts of the Y-Chart paradigm in order to offer an in-deep presentation of the paradigm. The secondary objectives of this paper are to offer a good introduction to the Y-Chart paradigm as well as contribute to the design debates surrounding its implementation.

This article is organized as follows. After presenting the background of the Y-chart paradigm in more detail, we summarize related work on various design methodologies that implement it. Subsequently, we elaborate on formalisms for describing applications and for specifying hardware platforms. After a detailed overview of three particular methodologies, we give a concise comparison that highlights alternative approaches for implementing the Y-chart paradigm. The examination and comparison of the methodologies show that the implementation of the Y-chart paradigm maybe done in various ways.

2. BACKGROUND

2.1 Y-Chart Based Design-Space Exploration

The Y-chart paradigm is an example of a model-driven engineering framework [Bezivin 2005]. The crux of explicitly separating applications (functionality) from the platform (architecture) as shown in Figure 1 leads to the following five-step approach to minimize overall design time:

(1) Create a model of the functionality performed by each application in a fashion that is independent of any specific platform and which expresses opportunities for parallel execution;

(2) Create a model of the platform that captures key characteristics of the services it can provide to applications using resources like processors, busses, memories and power; as well as the cost of these services in terms of area, performance, energy, etc;

(3) Define a mapping of how the platform is deployed to execute the applications;
(4) Evaluate the quality of the parallelized applications mapped onto the platform in terms of area, performance, energy, etc and decide on improvements for the application software, platform design or mapping in case the results are unsatisfactory. If so, repeat step 1, 2 or 3 respectively according to the new insights until a solution is found that satisfies all design requirements;
(5) Realize the solution in terms of synthesising hardware and compiling software.

Y-chart based design methodologies assist designers by providing : (i) a coherent set of formalisms, (ii) techniques for analyzing design solutions and (iii) techniques for searching the design space. They often offer computer aided design tools. Formalisms refer to languages for writing down models. To efficiently exploit platforms with multiple processing elements, it is necessary to make potential parallelism in applications explicit. *Models of Computation* (MoC) are formalisms that describe systems in terms of computation and communication entities. Computation entities (or tasks) capture functional behaviour, whereas communication entities represent data and control dependencies between concurrently executed tasks. Most MoCs will help designers to make explicit potential parallelism. An application model created in step 1 expresses the services required for executing functional behaviour. These service requests are often characterized by the number of instructions to execute or the amount and size of data to be communicated. The term *Model of Architecture* (MoA) is sometimes used to denote a MoC that is specifically intended for describing platforms. A platform model for step 2 specifies the services that the resources of a platform can provide as well as the cost of using them (cycles, energie, etc.). Notice that although we use MoC and MoA to emphasize the purpose of formalisms, Y-chart based design methodologies may actually use the same formalism for describing both applications and platforms. A mapping in step 3 basically represents a possibly interesting match between the service requesting computation and communication entities in applications and the service providing resources in a platform. When a specific combination of application models, a platform model and a mapping yields satisfactory results with respect to the design requirements, then step 5 realises this particular design solution using appropriate software and hardware synthesis tools.

The Y-chart paradigm is very suitable for design problems where platforms offer multiple resources in order to execute a particular computation. The suitability is a consequence of the ease of defining various mappings between an application and a platform which are defined separately. For design problems where mapping alternatives are fairly absent, such as in telecommunication networks or traditional control systems, applying the Y-chart paradigm can be counterproductive. The reason for this is that such systems actually benefit from a higher degree of coupling between the functionalities they offer and implementation of latter. Methodologies supporting the development of such integrated models include SystemC based approaches [Grotker et al. 2002], System-on-Chip Environment [Gajski et al. 2000] and Software/Hardware Engineering [Theelen et al. 2007]. These methodologies often rely on successive refinement of the integrated model towards a synthesisable design where a distinction between applications and the platform on which they run is less prominent than in the Y-chart paradigm.

2.2 Models of Computation

An interesting view on MoCs is given in [Burch et al. 2001]:

"A model of computation is a distinctive paradigm for computation, communication, etc. For example, the Mealy machine model of computation is a paradigm where data is communicated via signals and all agents operate in lockstep (we use "agent" as a generic term that includes both hardware circuits and software processes). The Kahn Process Network model is a paradigm where data carrying tokens provide communication and agents operate asynchronously with each other (but coordinate their computation by passing and receiving tokens). Different paradigms can give quite different views on the nature of computation and communication. In a large system, different subsystems can often be more naturally designed and understood using different models of computation."

Having different views on the nature of computation and communication implies that modeling an application with various MoCs may result in models that differ significantly in the amount of details that is expressed about computation and computation. In the context of the Y-chart paradigm, MoCs that explicitly express (potential) concurrency between computations are of special interest. We categorize MoCs as control-oriented, dataflow-oriented and process-oriented:

—*Control-oriented* MoCs consider systems as automata, which consist of states and transitions between these states. A state captures a certain status reachable by executing a (collection of) computation(s), while transitions describe possible changes in this status (i.e., the execution steps). Although automata are mostly suited for describing pure sequential systems, some control-oriented MoCs such as Communicating Finite State Machines (CFSM) [Brand and Zafiropulo 1983] and Co-design Finite State Machine (CDFSM) [Balarin 1997] allow describing a system as a collection of concurrent state machines that can communicate with each other.

—*Dataflow-oriented* MoCs describe systems in terms of tasks (actors or processes), channels and tokens. A task is a computation entity that can potentially be executed in parallel with other tasks. Tasks communicate with each other by exchanging tokens through channels. Such a token denotes an indivisible unit of information. The channels often include FIFO buffers to enable the sending and receiving tasks to run at a different rate, while successively communicated tokens are processed in-order.

—*Process-oriented* MoCs use processes and events as major modeling entities. A process represents a computation entity that may be executed in parallel with other processes. Processes can synchronise based on communication events, which can for example be in terms of signals or passing messages.

It is sometimes possible to express a MoC of one type by using a MoC of another, however such alternative representations are not always very intuitive. Such an example is the process-oriented MoC of SystemC [Grotker et al. 2002] that can express various control-oriented, dataflow-oriented and other process-oriented MoCs. Another such example is use of certain types of automata (i.e., control-oriented MoCs) to formally express the semantics of dataflow- and process-oriented MoCs such as Synchronous Dataflow [Lee and Messerschmitt 1987; Ghamarian 2008] and the Par-

	MoC	Reference	Concurrency	Communication	Time	Explicit Non-Determinism	Stochasticity	Analytic Tractability
Control-Oriented	Finite State Machines	[Hopcroft and Ullman 1979]	-	-	Integer-valued*	-	-	++
	Discrete-Time Markov Chains	[Chung 1967]	-	-	Discrete real-valued distributions	-	Discrete distributions on time	++
	Timed Probabilistic Systems	[Alur 1991]	-	-	Discrete real-valued distributions	+	Discrete distributions on behavior and time	+
Dataflow-Oriented	Synchronous Dataflow	[Lee and Messerschmitt 1987]	Asynchronous	Asynchronous FIFO buffered, token-driven	Integer-valued*, discrete integer-valued distributions*	-	Discrete distributions on time*	++
	Cyclo-Static Dataflow	[Bilsen et al 1995]	Asynchronous	Asynchronous FIFO buffered, token-driven	Integer-valued*	-	-	++
	Scenario-Aware Dataflow	[Theelen et al 2006]	Asynchronous	Asynchronous FIFO buffered, token-driven	Discrete real-valued distributions	+*	Discrete distributions on behavior and time	++
	Boolean Dataflow	[Buck 1993]	Asynchronous	Asynchronous FIFO buffered, token-driven	-	-	-	o
	Kahn Process Networks	[Kahn 1974]	Asynchronous	Asynchronous FIFO buffered, token-driven	-	-	-	--
	Dynamic Dataflow	[Buck 1993]	Asynchronous	Asynchronous FIFO buffered, token-driven	-	+	-	--
	Reactive Process Networks	[Geilen and Basten 2004]	Asynchronous	Asynchronous FIFO buffered, token-driven and synchronous control events	-	+	-	--
	Real-Time Calculus	[Thiele et al 2000]	Asynchronous	Asynchronous (FIFO) buffered, data-driven	Continuous	+	+*	++
Process-Oriented	Esterel	[Boussinot and Simone 1991]	Synchronous	Unbuffered asynchronous signals	Discrete integer valued	-	-	+
	Communicating Sequential Processes	[Hoare 1978]	Asynchronous	Synchronous message passing	-	-	-	++
	Communicating Concurrent Systems	[Milner 1989]	Asynchronous	Synchronous message passing	-	-	-	++
	Metropolis Meta Model	[Balarin et al 2003]	Asynchronous	Unbuffered asynchronous events	Discrete real-valued	+	-	o
	Timed Automata	[Alur and Dill 1994]	Asynchronous	Synchronous message passing	Discrete real-valued or continuous	+	Discrete distributions on behavior and discrete or continuous distributions time*	+
	Parallel Object-Oriented Specification Language	[Bokhoven 2002]	Asynchronous (two levels)	Synchronous message passing	Discrete real-valued	+	Discrete distributions on behavior and time	+

Table I. Comparing some example models of computation

allel Object-Oriented Specification Language [Bokhoven 2002]. Some research has been done on unifying various MoCs. Such unification is important when defining subsystems with different MoCs. Examples of unifying MoCs are the Tagged Signal Model Framework (TSM) [Lee and Sangiovanni-Vincentelli 1998] and the Ptolemy Project [Brooks et al. 2005].

Table I gives an overview of a number of widely used MoCs. It summarize their approach to concurrency, communication and time. These aspects are essential for modeling distributed embedded system. A * indicates that the aspect is supported by some variant of the MoC. The table also indicates whether the abstraction mechanisms of explicit non-determinism (i.e., not implied by concurrency but by some other language construct) and stochasticity are supported. The last column gives some impression to what extend the basic form of a MoC is analytically tractable. Comparing MoCs on this last aspect is very difficult and the results in Table I should therefore be taken with precaution. A -- indicates that (nearly) all properties are undecidable at design time. In case some design-time analysis is possible, such as structural consistency checks, a - is used. Conversely, ++

indicates that (nearly) all models expressed in the considered MoC are fully design-time analyzable, both for correctness and performance. A + expresses that not all but many models are fully analyzable. A problem could for example be the implication of an infinite state space by certain models. A o denotes analytical tractability between - and +.

2.3 Models of Architecture

As the previous section suggests, the concept of MoCs is very mature. Conversely, the concept of MoAs has only started to emerge in the last decade. Although there exists no formal definition of the term, two major views on MoAs can be identified:

(1) Describe platform components using existing MoCs;
(2) Specify platforms using dedicated formalisms.

The earliest uses of the first approach are Cadence Alta VCC [Martin 1998] and POLIS. Others examples are SystemC-based solutions, Metropolis [Balarin et al. 2001] and the automated design flow of [Stuijk 2007]. Each of these uses the same modeling constructs to define applications and platform models.

Dedicated formalisms for specifying platforms are also known as Architecture Description Languages (ADL) [Medvidovic and Taylor 2000; Mishra and Dutt 2008]. It is very difficult to categorize ADLs (or MoAs) in a manner similar to Table I since very few ADLs focus on the same platform types or on the same platform details. [Qin and Malik 2002] presents major contributions for ADLs to describe general-purpose processors. [Gries and Keutzer 2005] presents Mescal, an ADL focused on Application-Specific Instruction-set Processors (ASIP). The IP-XACT Standard (www.spiritconsortium.org) [Kruijtzer et al. 2008] is an example of an ADL that focuses on IP-based platforms. Other examples are proprietary languages that configure certain template descriptions in another MoA or MoC. These typically require special compilers to expand (generate) a full specification. Examples of this approach are Colif [Cesario et al. 2001] and the XML specifications for the tool in [Theelen 2007].

In the context of distributed embedded systems, many researchers recognized processing (e.g., general-purpose processors, accelerators and dedicated controllers), communication (e.g., busses, network-on-chip, and i/o interfaces) and storage (e.g., memories and hard disks) resources as elements of a MoA. The need for storage resources originates from using a certain processing or communication resource, which means that storage resources only provide a service to applications via these resources. Another indirectly provided resource is power (or energy), which is consumed by any of the other resource types. Figure 2 illustrates the hierarchical relation between the services that resources provide to each other and to the service requesting computations and communications of applications through mappings. Notice that sharing a processing, communication and storage resource requires scheduling the moments at which each of the involved service requesting entities accesses this resource. Depending on the exact evaluation criteria a designer is interested in (area, performance, energy, etc), a MoA should allow describing the indicated four resource types, thereby taking the service providing relations between resource types and potential contention due to resource sharing into account. Notice that the service providing relations between the different resources are in

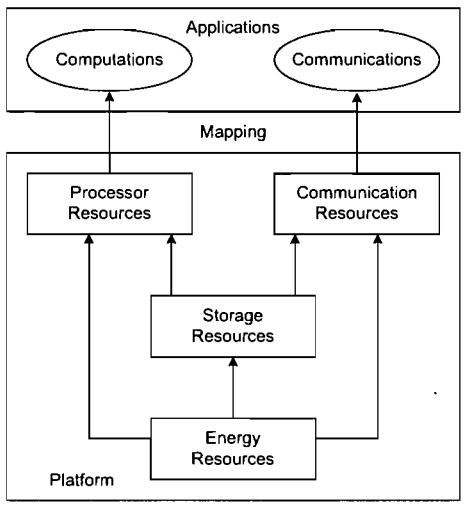


Fig. 2. Hierarchy of providing services between resources in platforms to applications

reality realized by physical connections (in hardware). A MoA may or may not require specifying the physical architecture of how the resources are interconnected, see also Section 4.2. The next section discusses some examples of special languages for describing the services provided by platforms.

3. METHODOLOGIES AND LANGUAGES

The Y-chart paradigm has been integrated into various design methodologies. The earliest well known implementations are Cadence VCC and POLIS. Other well-known examples include SPADE [Lieverse et al. 2001], Daedalus[Thompson et al. 2007] and SymTA/s [Hamann et al. 2004]. This section presents three implementations that will be used for design comparison. This particular selection was made because each implementation proposes some interesting design approaches. Metropolis differentiates itself from the others by its strict adaption of the Y-Chart paradigm. Moreover, it uses an interesting combination of informal specification for models and formal specifications for mappings. The Y-chart variant of the Software/Hardware Engineering methodology differentiates itself by adapting an existing methodology in order to incorporate the Y-Chart paradigm. Moreover, it uses a single formal language for both modeling and mappings. The Distributed Operation Layer incorporating Modular Performance Analysis differentiates itself by using Modular Performance Analysis which permits high-level performance analysis.

3.1 Metropolis

The Metropolis project [Balarin et al. 2003] has been running since 1999 and is a joint initiative of the Gigascale Silicon Research Center, the University of California and the Cadence Berkeley Laboratories. The project focuses on the modeling and the design of systems using a platform-based approach, as well as on the integration

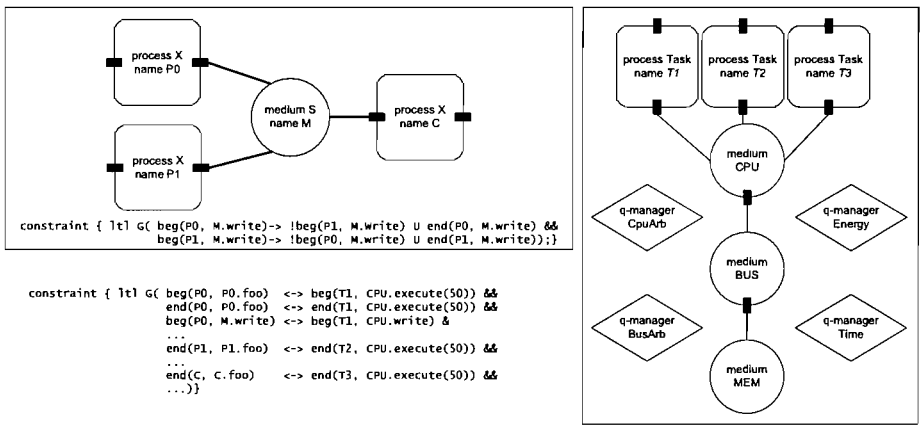


Fig. 3. Metropolis example (application model, platform model and mapping)

of multiple MoC in order to describe heterogeneous systems at various abstraction levels [Sgroi et al. 2001].

The primary objective of Metropolis is to enable the development of design flows for different application domains. To achieve this objective, Metropolis promotes separation of concerns and platform-based design [Pinto 2004]. It strongly relies on the Y-chart paradigm and therefore follows its five step design approach explicitly. Metropolis advocates the concept of reuse by explicitly decoupling the specification of independent aspects over a set of abstraction levels. Other research has focused on a compositional modeling methodology around Metropolis [Goessler and Sangiovanni-Vincentelli 2002]. The Metropolis framework [Davare et al. 2007] consists of three main elements:

—An infrastructure which encompasses the Metropolis Meta-Model formalism, a compiler and an Abstract Syntax Tree specification;

—A Meta-Model Library for MoCs and architecture platforms;

—A tool set for simulation, verification and synthesis.

The Metropolis-Meta-Model (MMM) forms the core of Metropolis. It is a Java inspired language that adds the necessary semantic and syntactical elements for system design (i.e., processes, communication channels, ports, etc.). MMM supports some novel features such as denotational formulas in Linear-Temporal Logic (LTL) and the predicate Logic of Constraints (LOC)[Balarin et al. 2001; Burch et al. 2001]. Moreover, the event model of MMM is based on the Tagged Signal Model Framework (TSM) [Lee and Sangiovanni-Vincentelli 1998]. Metropolis is not a formal method in the sense that it is not based on strict mathematical definitions. However, portions of the MMM language such as the LTL and LOC expressions are formal. These can be verified with model checking technologies [Yang et al. 2006]. Having said this, the MMM language does have precise semantics for modeling and execution. To integrate formal methods into Metropolis, mechanisms are incorporated in MMM that enable tools to process suitable subsets of a design. The work in [Densmore 2004a] is an example of this principle for control graph analysis.

Metropolis has been applied in many academic and industrial case studies ranging over various domains such as automotive, wireless multimedia, analog/mixed signal systems and micro-processor design [Sgroi et al. 2001; Meyerowitz 2004; Zeng et al. 2006]. Similar to SystemC-based methodologies, Metropolis scales to fairly complex systems.

3.1.1 *Application Modeling.* Conforming to the Y-chart paradigm, Metropolis promotes developing application models which are called functional descriptions. As mentioned previously, Metropolis encourages a platform-based design flow, hence an application model is considered as a platform (functional platform) with a high degree of abstraction [Pinto 2004]. In the Metropolis literature, the term *denotational definition* is often used for functional descriptions. The MMM language is based on the TSM denotational framework which has been proven to support a vast amount of MoCs [Lee and Sangiovanni-Vincentelli 1998]. In [Densmore et al. 2006], it is suggested that Metropolis can support any (non-stochastic) MoC.

The basic concepts of MMM [Balarin et al. 2003] are processes, interfaces, ports, events and media. A *process* represents a sequential program and is also called a thread. A process communicates with its environment through one or more *ports*. A port is specified with an *interface*, which refers to a communication contract for exchanging information with the environment. Interfaces are implemented with *media*. Once a network of connected processes and media is defined, like the example shown in Figure 3, the behavior of the network can be specified with the concept of *events*, which represent specific behavioral actions in the application model.

Once an application model is completed, it is possible to add various constraint specifications by using LTL and LOC expressions to it as illustrated in Figure 3. LTL formulas specify coordination constraints between processes. LOC formulas describe performance constraints. The Metropolis framework provides a library of functional platforms elements in YAPI [Kock et al. 2000] and TTL [Pinto 2004]. Metropolis also offers some support for using non-determinism as an abstraction mechanism. Two constructs exist; non-determinate variable assignment and a limited form of non-deterministic code execution.

3.1.2 *Platform Modeling.* As indicated, Metropolis describes application and platform models both as platform models in MMM but at different levels of abstraction. Where application models are concerned with the definition of functional aspects, they do not define any resource utilization aspects. Resource utilization aspects are expressed in platform models. Nevertheless, platform models are defined with the same primitives as application models (processes, interfaces, ports, media and events) capturing the functionality they offer and the cost (efficiency) of that functionality. Platform functionalities are modeled as services (methods) defined in interfaces. The cost of a service is modeled by associating events with the various portions of the implementation of a service and then annotating each event with a value that represents its cost. With the language primitives of MMM, any resource type can in theory be modeled. References [Balarin et al. 2003; Pinto 2004; Meyerowitz 2004] illustrate the modeling of processor, communication and storage resources. The way in which resources are interconnected must be modeled explicitly. To take aspects such as scheduling and power consumption into account,

the concept of quantity managers is introduced. Quantity managers are responsible for the assignment of tags to events and for the ordering of events. Figure 3 shows an example of a platform composed of a single processor, a bus and a memory. The computation resources of the processor are modeled explicitly using tasks.

An important aspect of the platform-based design approach followed by Metropolis is the ability to refine high-level platform models into more detailed ones. The MMM language supports this by providing primitives for declaratively specifying that a particular model is a refinement of an element or a group of elements of another model. The primitives also enable to specify in detail how the original model can be replaced by the more detailed model to achieve refinement [Balarin et al. 2003; Densmore et al. 2004; Densmore 2004b; 2004a]. The latter reference discusses a formal approach to verify that a refinement preserves certain properties of the original model.

3.1.3 *Mapping.* Metropolis offers a novel way of mapping an application model to a platform model based on formal synchronization expressions. LTL is used to specify these expressions. Platform models may contain non-determined values, such as the memory address of a variable or the priority of a task. These non-determined values are fixed by means of value mappings in the synchronization expressions. Hence, it is possible to map a value from an application model to a value in the platform model during synchronization. Figure 3 shows an example of mapping expressions.

3.1.4 *Evaluation (Analysis and Exploration).* The primary vehicle for analysis in Metropolis is simulation. Two simulation tools are available for MMM models, which respectively generate SystemC or pure C++ specifications for simulation purposes. The compiler provided with Metropolis is also capable of generating monitors in order to verify LTL and LOC expressions during simulation [Balarin et al. 2003; Yang et al. 2006]. The work in [Chen et al. 2005] discusses a simulation-based deadlock analysis technique. Any quantities that have been explicitly added to a model can be traced and analyzed after a simulation. Instead of using simulation, parts of an MMM model can be formally verified using model checkers. Metropolis offers a tool to produce PROMELA code from an LTL expression, which can then be verified using SPIN [Yang et al. 2006]. The framework also provides a LOC expression checker [Balarin et al. 2003].

The design-space exploration step of the Y-chart paradigm is not automated in Metropolis [Zeng et al. 2006]. Designers must manually create alternative platform models and mappings for an application model and use the supported analysis techniques for evaluation. The work in [Balarin et al. 2003] presents a *quasi-static* scheduling technique in order to schedule a concurrent specification on shared resources. One may consider that the technique allows automatic design-space exploration for shared resource utilisation.

3.2 Distributed Operation Layer and Modular Performance Analysis

The design framework DOL (Distributed Operation Layer) as described in [Thiele et al. 2007] closely follows the Y-Chart paradigm of Figure 1. It is targeted towards the mapping of applications to multi-processors on a chip (MPSoC). The approach is based on a very early implementation of the Y-chart paradigm pre-

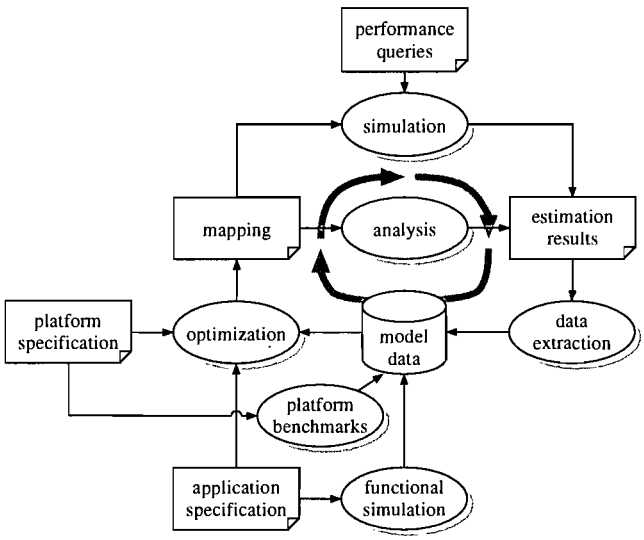


Fig. 4. Y-chart based optimization cycle in the DOL environment

sented in [Teich et al. 1998]. This latter approach uses graph structures to formally specify the application and target platform. The mapping is also modeled as a graph, which captures not only the binding information, but also scheduling. Only the feedback-edges regarding the improvement of the platform and mapping shown in Figure 1 are supported by [Teich et al. 1998], where the necessary optimizations are performed automatically based on multi-objective evolutionary algorithms. The DOL environment, see Figure 4, applies the same principle but with a much more refined application/platform modeling and analysis approach. In particular, the application is specified with a set of communicating tasks, where the communication structure is provided as an XML specification. The individual tasks are sequential programs that are given in a programming language for which an appropriate compiler is available; the API of DOL only provides the necessary semantical interfaces to the communication channels. In a similar way, the underlying hardware platform and its capabilities are described by an annotated graph structure, including processing and memory resources and their interconnection through communication resources like buses and networks. The mapping, again specified by means of an XML specification, relates tasks to computing resources and memories. It also links tasks to paths on the communication platform and specifies the applied resource sharing mechanism.

The analysis is performed using a hierarchy of various methods. The fastest evaluation of a system configuration is done using the Modular Performance Analysis (MPA) framework [Chakraborty et al. 2003; Wandeler et al. 2006], a compositional performance analysis method for heterogeneous distributed embedded systems. In addition, the necessary parameters are determined using a simulation-based profiling of the application and the underlying hardware platform. The multi-objective optimization is based on evolutionary algorithms and uses the PISA environment

[Bleuler et al. 2003] that is publicly available. The mapping information is used to generate highly efficient code for the target platform using a dedicated generation of the hardware-dependent software including calls to the underlying operating system. Finally, a trace-based simulation can be used to determine the extra-functional properties of mapped applications with high accuracy.

We now give a short overview of the theoretical core underlying the design-space exploration framework of MPA. It is a very efficient method due to the high-level of abstraction of the models that it uses. MPA is an analytical approach based on the Real-Time Calculus [Thiele et al. 2000], which has its foundations in methods for worst-case analysis of communication networks (Network Calculus) [Cruz 1991]. MPA is an example of exact performance analysis approaches that can determine guaranteed performance limits. While these techniques can compute hard performance bounds, they abstract from the complex interactions and state dependent behavior in the system. MPA uses a unifying model for the representation of different event patterns in the form of *arrival curves* known from the communication domain [Cruz 1991]. In addition, it uses a similar concept called *service curves* to represent the resources and their computational or communication capabilities, which allows MPA to model complex hierarchical scheduling schemes in distributed embedded systems. The detailed modeling of the capabilities of the shared resources and the event streams can lead to highly accurate performance results, see for example [Chakraborty et al. 2003].

An MPA model is a performance network of components, where application tasks are mapped to computation and communication resources. One can differentiate between three main entities: *event streams* represented as arrival curves, *resource streams* represented as service curves, and *application tasks* represented as processing components. Application tasks are activated by event streams which they process by considering the interaction of the event streams with the resource streams. On a higher level, the model is a network of components that communicate with each other through *event interfaces*. Performance metrics for the whole application are computed by combining the behavior of the individual components. This modularity aspect achieves short analysis times even for large systems. Typical performance metrics computed with MPA are upper and lower bounds on buffer levels, end-to-end delays experienced by events, and the number of events that can be processed in a time unit (throughput). MPA supports refinement of the elements of a performance network with the extension Real-Time Interfaces [Thiele et al. 2006]. It promotes interface-based design of embedded systems with the concept of adaptive interfaces.

The method has been implemented as a Matlab toolbox [Wandeler and Thiele 2006b] where a system is described as a Matlab script file. The toolbox implements the Min-Plus and Max-Plus algebraic operators and provides the facilities to represent the arrival and service curves, and the processing components. It also contains a library of predefined components that assist the designer in building a system performance model. Different case studies have been performed, covering for example car infotainment systems, MPSoC platforms for multimedia applications, digital signal processing systems and network processors.

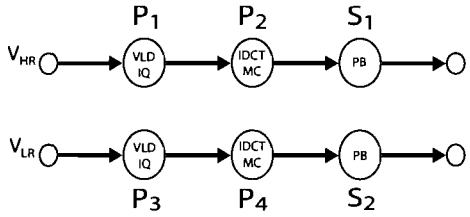


Fig. 5. Task graph description of an application processing two MPEG2 video streams in parallel

3.2.1 *Application Modeling.* Modeling an application in DOL-MPA involves capturing the application tasks and the event flows between them. The application task model provides information about the processing semantics of the tasks. An example of an application task graph for parallel decoding of two MPEG2 video streams is shown in Figure 5.

The goal of MPA is to analyse the timing behavior of an application considering a large class of possible event flow characteristics. Traditionally, the timing behavior of event flows is modeled as being periodic or periodic with jitter. However, such abstract representations cannot adequately capture the complex timing behavior of event flows in a highly parallel or distributed system. Hence, a more powerful abstraction is needed. Variability Characterization Curves (VaCC) substantially generalise the traditional representations and can express any possible timing behavior of an event stream. Event streams in MPA are captured by a special kind of VaCCs, denoted as arrival curves. They provide upper and lower bounds on the number of events in any time interval. For an event stream α, there are at most $\alpha^u(\Delta)$ and at least $\alpha^l(\Delta)$ events within any time interval $[t, t+\Delta)$ for all moments t. Figure 6 illustrates how arrival curves bound the behavior of a periodic event stream. Information about the arrival curves representing the interactions with the environment can come from several sources. Firstly, they can be computed analytically if an interaction has some pattern such as periodic, periodic with jitter, sporadic, etc. In case that they are unknown, they can be computed as an envelope of a set of recorded traces. Finally, they can be derived from specifications like UML sequence diagrams that describe the behavior of the event-generating devices.

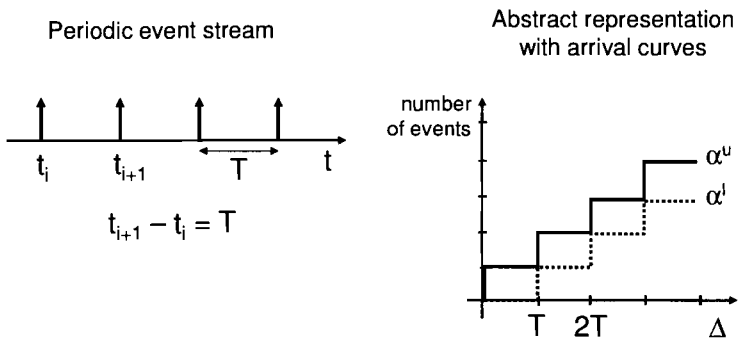


Fig. 6. Modeling periodic event streams in MPA

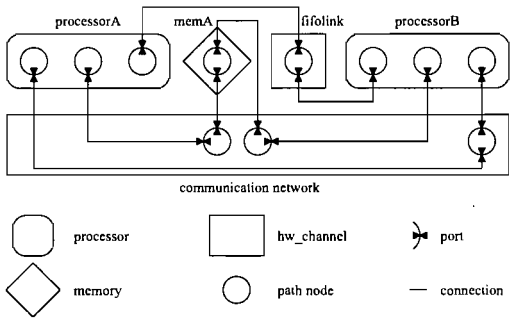


Fig. 7. Platform model in the DOL environment

For an accurate performance analysis, it is vital that the resource demands associated to individual events are modeled precisely. To this end, MPA supports the conversion between event-based arrival curves and resource-based arrival curves. Besides simple models based on best-case and worst-case behavior, automata can be used to model arrival patterns of different event types and the corresponding resource requirements [Wandeler and Thiele 2007]. A similar approach is used in MPA to model state-dependent workload demands as introduced by cache memories [Chakraborty et al. 2007].

3.2.2 *Platform Modeling.* The DOL environment extracts non-functional properties of the platform and builds abstract models of the resource services offered by the platform based on the MPA model. More specifically, MPA models the resource capabilities of all computation and communication resources and it can provide information on how these capabilities are affected by the workload of tasks and communications. Resources in MPA are modeled explicitly and therefore considered 'first class citizens' of the approach. Figure 7 shows a simple platform specification in the DOL environment.

Resource capabilities, like event streams, can be described with VCCs. The service curves $\beta^u(\Delta)$ and $\beta^l(\Delta)$ provide upper and lower bounds on a service β within any time interval $[t, t+\Delta)$ for all moments t. The unit of service depends on the kind of resource, for example instructions or cycles for computation, and bytes for communication. The service curves of a resource can be determined using data sheets, analytically derived properties, or by measurements. Figure 8 illustrates the service curves that bound the service offered to a single task by a single slot in a Time Division Multiple Access (TDMA) resource. Using service curves, MPA can model any arbitrarily complex resource capabilities and is able to model arbitrary scheduling hierarchies.

3.2.3 *Mapping.* In a real-time system, an incoming event stream is usually processed by a set of Hardware/Software components. After the mapping of tasks to computing resources and streams to communication paths in the DOL environment, a performance model of the system is determined. Based on the MPA method, this performance model is a network of performance components where each of them has as inputs abstract event streams and abstract resource streams. More specifically, a

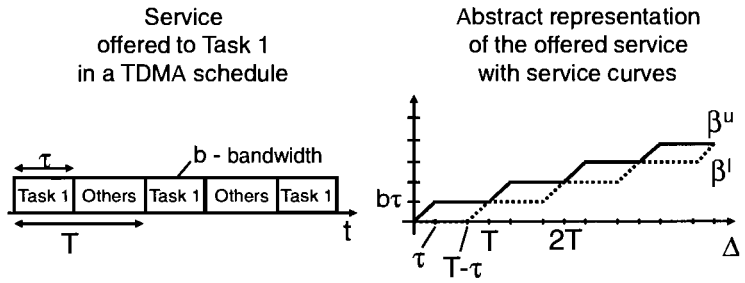


Fig. 8. Modeling a TDMA service in MPA

performance component defines equations for functional transformations of arrival and service curves where the actual equations depend on the processing semantics of the modeled task. In other words, the abstract resource streams interact with the abstract event streams in a performance component. This performance component produces as outputs a transformed abstract event stream and a remaining abstract resource stream that is available to other tasks mapped onto the same resource. The mapping and the respective MPA performance network of the task graph from Figure 5 are shown in Figure 9.

Given a specific mapping, MPA also needs information about the workloads induced by the mapped application tasks running on the specific resource. This information is needed by the performance components for the transformations of arrival and service curves. Usually, it represents upper and lower bounds on the service needed by the component to process one, two, and more consecutive events from the incoming event stream. Such information can come from cycle-accurate simulations of the application tasks or from static analysis of the program code and the chosen hardware architecture [Wilhelm et al. 2008], see also Figure 4.

If several tasks are mapped to the same resource, a resource sharing policy needs to be determined. Scheduling in MPA is modeled by the way performance components are interconnected. Supported scheduling policies are preemptive and non-preemptive fixed priority, TDMA, earliest deadline first (EDF), generalised processor sharing (GPS), first-in first-out (FIFO), hierarchical scheduling, and different server strategies [Wandeler and Thiele 2006a]. An example for modeling of preemptive fixed priority scheduling are tasks $P2$ and $P4$ in Figure 9 sharing $CPU2$.

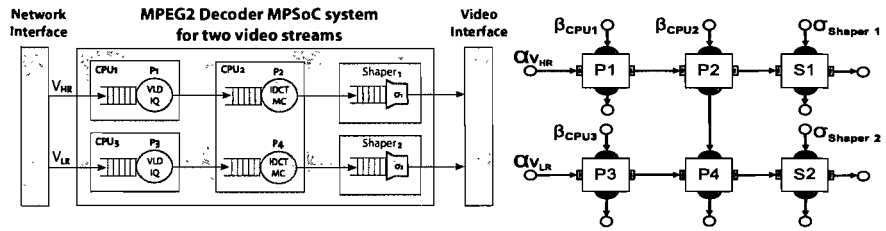


Fig. 9. Specifying a mapping configuration as an MPA performance network

3.2.4 *Evaluation (Analysis and Exploration).* The DOL design environment closely follows the Y-chart approach. The exploration is done using a multi-objective evolutionary optimization approach using the PISA platform [Bleuler et al. 2003]. In terms of performance estimation, DOL is not bound to a particular method. It can use analytical methods such as MPA or simulation-based ones such as functional and trace-based, see Figure 4.

MPA can determine the characterisations of all event and resource streams in the network of performance components using the abstract characterisations of the workloads and the input event and resource streams. From the computed arrival and service curves, MPA deducts information about the utilisations of the resources, the end-to-end delays between any two components, the necessary buffer spaces for event and packet queues, and the throughput. The modularity, efficiency and scalability of the MPA models makes the method highly suitable for quickly analysing a large number of different mappings and resource sharing policies during design space exploration. Extension of applications by adding tasks is modeled by simply adding components in the performance network. A single performance model can include different resource sharing strategies without affecting the accuracy of the performance analysis results.

MPA provides hard upper and lower bounds on the computed performance results. However, it is a worst-case approach that covers all possible corner cases regardless of their probability of occurring. Even if the results can be very close to simulation results [Chakraborty et al. 2003], in some cases a designer is interested in the average case behavior of a distributed embedded system. In this sense, MPA is a complementary method to other simulation-based or stochastic analysis techniques. It is able to analyse the timing non-determinism of complex distributed embedded systems while providing hard guarantees on the resulting end-to-end behavior.

3.3 Software/Hardware Engineering

Originally introduced in [Putten and Voeten 1997], Software/Hardware Engineering (SHE) evolved into an industrial-strength system-level design methodology accompanying many methods, techniques and tools for the design, analysis and synthesis of distributed real-time hardware/software systems [Theelen et al. 2007]. SHE considers the Y-chart paradigm as a way to specialise its generic model-driven engineering framework to facilitate a flexible approach towards multi-processor design for streaming multi-media applications [Wijk et al. 2003; Florescu et al. 2007; Theelen 2008]. Because the provided methods, techniques and tools are not specifically targeted to the Y-chart paradigm, SHE is more generally applicable than methodologies that enforce a separation between application and platform models.

SHE is built around the Parallel Object-Oriented Specification Language (POOSL), which is a very expressive general-purpose modeling language with a process-algebraic formal semantics [Bokhoven 2002]. It includes powerful primitives for intuitively describing (hierarchical) structure, concurrency, communication, data, time and stochasticity. POOSL distinguishes three types of object classes originating from the idea of modeling active and passive system components separately. *Data objects* represent passive information that is generated, communicated, processed, etc. by active components, which are modeled with processes and clusters. *Processes* represent basic active components that may initiate both se-

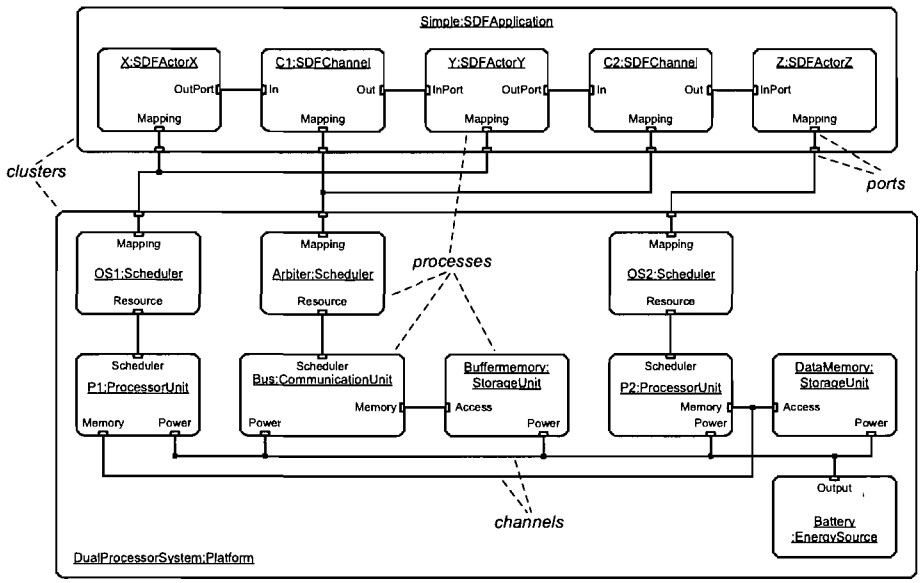


Fig. 10. Instances of process and cluster class modeling patterns capturing a streaming system

quential and concurrent behavior. *Clusters* allow describing hierarchical structures between active components (processes and clusters). Processes and clusters can communicate with each other by passing messages over channels through ports. Such messages may include data objects to specify the exchange of information. Figure 10 illustrates a structure of active components in POOSL, where a simple application with 3 tasks (actors) is mapped onto a battery-powered dual-processor platform.

The mathematically defined semantics of POOSL is the crux in supporting

—Interactive simulation of models [Geilen 2002; Bokhoven 2002] to facilitate validation of whether a model adequately represents the system under design;

—Model checking and simulation-based analysis of functional correctness [Geilen 2002] and performance [Theelen 2004] like absence of deadlock and throughput;

—Generation of real-time control software [Huang 2005], which relies on a step-wise refinement approach that guarantees preservation of functionality and timing.

Combining the Y-chart paradigm with SHE currently exploits only the first two aspects and hence does not cover the step of synthesising the final design solution (see also Section 2.1). The Y-chart variant of SHE is primarily based on modeling patterns. *Modeling patterns* are parameterised model components for capturing typical aspects of systems in a certain modeling language [Theelen 2004]. In this case, it refers to a collection of template data, process and cluster classes targeted to the application domain of streaming multi-processor systems. Several modeling patterns have been developed to ease constructing performance models for Y-chart based design-space exploration. Hence, these modeling patterns cover both application modeling and platform modeling. A specialised tool called PREMADONA

automates constructing POOSL models by properly instantiating the modeling patterns from an MPSoC specification given in XML [Theelen 2008].

SHE has been applied in many academic and industrial case studies [Theelen et al. 2007] ranging from communication protocols, internet routers, television systems, car-infotainment systems, network-on-chip based multi-processors, printer systems up to wafer scanners. Although model checking has proven to be feasible for relatively small systems, most case studies relied on simulation-based analysis. The simulation tools of SHE have shown to be competitive with tools like OPNET and SystemC in terms of simulation speed. Analysing models with over 10^6 parallel processes demonstrated scalability to systems of industrial complexity.

3.3.1 *Application Modeling.* Modeling applications following the Y-chart variant of SHE is based on capturing service requesting behaviors, such as those shown in Figure 2, in POOSL. Because of its expressive power, any MoC can in essence be represented, even when including all functional details. However, considering the Y-chart's focus on performance analysis, expressing applications in POOSL urges to abstract from functional details, which conforms to SHE's strategy. Relevant aspects like the structure of how computations interact with each other and resource requirements like execution times and memory usage must be taken into account. To ensure an adequate representation of any dynamism in applications, one may even use probabilistic and non-deterministic approaches as abstraction mechanisms. Moreover, POOSL allows data or control events to originate from files, which facilitates evaluating how a system reacts for example to observed user interactions.

To ease using the Y-chart paradigm with SHE, several POOSL modeling patterns have been developed for MoCs that abstract from functional details but still allow annotations with key resource requirement characteristics similar as to what SHE advocates for performance modeling. These include the dataflow-oriented MoCs of Synchronous Dataflow (SDF), Cyclo-Static Data-flow (CSDF) and Scenario-Aware Dataflow (SADF) [Theelen 2008]. The top cluster in Figure 10 shows actually a combination of the modeling patterns for an SDF application. For time-driven and event-driven task specifications similar to those common in traditional scheduling theory, POOSL modeling patterns have been developed that probabilistically mimic the behavior of such tasks for uncertainties regarding activation latency, release jitter and output jitter [Florescu 2007]. Figure 11 illustrates how the modeling patterns for SDF capture the typical interaction between computation and communication entities in such applications. It also shows how the scheduler of a platform acknowledges a request for executing a computation.

Although modeling patterns form the crux to Y-chart based design with SHE, specifying POOSL may not be the most convenient way of constructing application models. Therefore, the more intuitive XML specification formats for specifying SDF, CSDF and SADF models defined in [Stuijk 2007; Theelen 2007] have been adopted as input language for the model generation tool PREMADONA. Similarly, task graphs consisting of time-driven and event-driven tasks can be described in the XML format defined in [Florescu 2007]. Given such an XML specification, PREMADONA automatically instantiates the appropriate combination of the patterns.

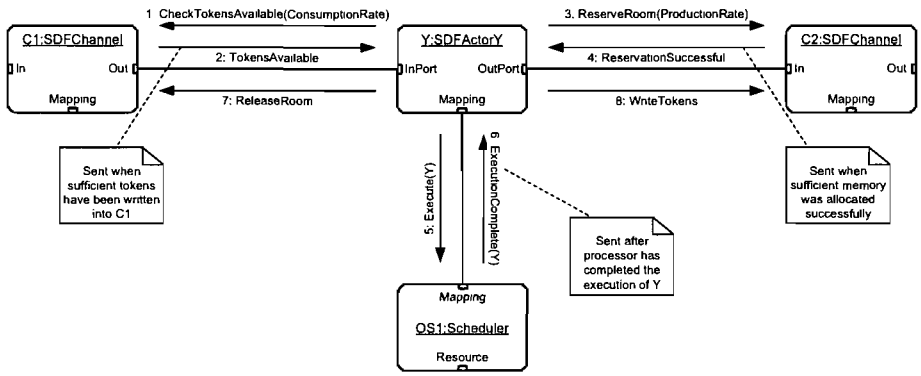
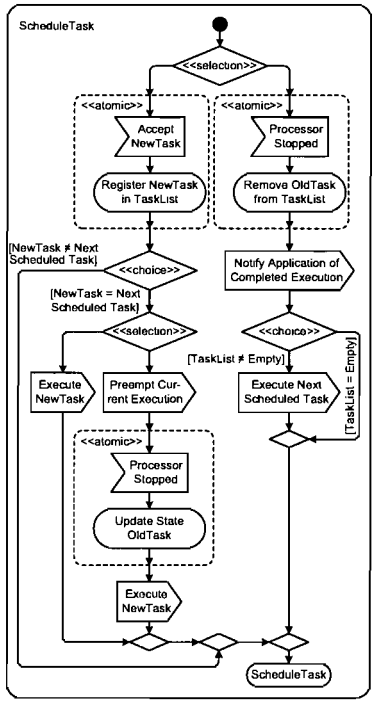


Fig. 11. Example of an interaction between an SDF application and a scheduler of a platform

3.3.2 *Platform Modeling.* Modeling platforms with SHE is based on capturing service providing behavior in POOSL. In essence, any resource type can be expressed, even at a fully synthesizable cycle-accurate register transfer level. However, since the Y-chart paradigm aims at performance evaluation of design alternatives, SHE advocates abstraction from implementation details; hence, only capturing crucial aspects that affect performance. These include the number and type of resources, the way in which they provide services to each other, and any aspect related to contention that arises from sharing resources, including any scheduling or arbitration mechanisms. To ensure an adequate representation of contention when abstracting implementation details, one may use probabilistic and non-deterministic approaches. These allow capturing technology-dependent uncertainties like unreliable communication media or deep submicron issues.

Various POOSL modeling patterns have been developed to represent all four resource types (processor, communication, storage and energy resources) of MPSoCs together with various types of non-preemptive and preemptive schedulers that can be used when sharing processor and communication resources [Theelen 2008; Florescu 2007]. In addition, a basic resource manager is available [Kumar et al. 2006]. Figures 12 and 13 show some behavioral details of two modeling patterns using activity diagrams and the corresponding POOSL code. The PREMADONA tool instantiates the modeling patterns when generating platform models from an XML specification that contains key parameters like clock frequencies, voltage/frequency scaling factors, and power consumption characteristics. The current collection of modeling patterns together with the XML specification format give a MoA for describing relatively simple network-on-chip based multi-processor platforms, which differs in various ways from other approaches such as those in [Balarin et al. 2003; Gries 2004; Thompson et al. 2007]. An example of such a difference is that no actual data processing is done, nor are instruction sets emulated (i.e., execution of the model does not provide a functional result). It is also not required to specify for example the size of memories or the number of concurrent connections that a network-on-chip can realize. These aspects are considered to be a result of the evaluation step. The abstraction goes even further in not requiring to specify how



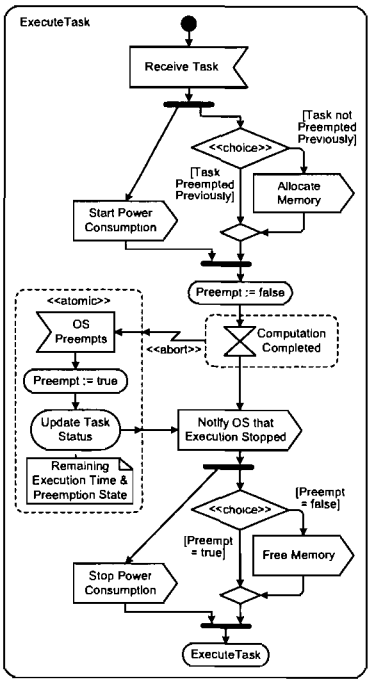

```
ScheduleTask()() |NewTask, OldTask: Task|

sel
   Tasks?Execute(NewTask)
      {TaskList register(NewTask)};
   if TaskList nextTask == NewTask then
      sel
         Processor!Execute(NewTask)
      or
         Processor!Preempt;
         Processor?Stopped(OldTask)
            {TaskList updateState(OldTask)};
         Processor!Execute(NewTask)
      les
   fi
or
   Processor?Stopped(OldTask)
      {TaskList remove(OldTask)};
   Tasks!ExecutionCompleted(OldTask);
   if TaskList notEmpty then
      Processor!Execute(TaskList nextTask)
   fi
les;
ScheduleTask()().
```

Fig. 12. POOSL Modeling pattern for preemptable scheduling of computations by a `Scheduler`

resources are interconnected, not even the topology of routers for a network-on-chip must be specified. These are seen as a result of the dependencies between application tasks and the chosen mapping.

3.3.3 *Mapping.* Figure 10 clearly shows that the Y-chart variant of SHE requires explicit specification of which processor resources execute what computations and which communication resources realize what communications [Wijk et al. 2003]. Explicitly mapping communications enables to abstract from the physical structure of how resources in the platform are interconnected, which conforms to the focuses on the service relations in Figure 2. By assuming a single interconnect between processor resources (i.e., a network-on-chip), the PREMADONA tool can automatically derive the mapping of communications from a mapping specified for computations. In that case, specifying the mapping of communications is obsolete [Theelen 2008].

The mapping of computations to processor resources and communications to communication resources is accomplished by means of exchanging service request - service acknowledgment messages between the involved modeling patterns, see also Figure 11. The fact that an application model gets feedback on what is going on in the platform is essential to model QoS and resource management as well as the reaction of the system to unpredictable events like user interactions [Goldschmidt and Hennessy 1993]. This approach differs from traditional trace-based mapping




```
ExecuteTask()() |StartTime: Real,
Preempt: Boolean, Task: Task|

OS?Execute(Task){StartTime := currentTime};
par
   Power!StartConsumption(ProcessorPower)
and
   if Task neverPreempted then
      Memory!Allocate(Task getRequiredMemory)
   fi
rap;
Preempt := false;
abort
   delay Task getExecutionTime
with
   OS?Preempt{Preempt := true;
      Task reduceTime(currentTime - StartTime)};
OS!Stopped(Task);
par
   Power!StopConsumption(ProcessorPower)
and
   if Preempt not then
      Memory!Free(Task getRequiredMemory)
   fi
rap;
ExecuteTask()().
```

Fig. 13. POOSL modeling pattern for preemptive execution of computations by a Processor

strategies as used in for example [Lieverse et al. 2001; Kienhuis et al. 2000].

3.3.4 *Evaluation (Analysis and Exploration).* SHE offers a broad spectrum of analysis opportunities based on the formal semantics of POOSL [Theelen et al. 2007]. The theory supports model checking by specifying functional correctness properties as well as best/worst-case, average-case and (expected/probabilistic) reachability performance properties in real-time temporal logics such as MTL [Koymans 1990], MITL [Alur 1991] and Temporal Rewards [Voeten 2002] using tools like SPIN [Holzmann 1991], UPPAAL [Larsen 1997] and PRISM [Kwiatkowska et al. 2002]. In case model checking is expected to suffer too much from state-space explosion, SHE offers simulation-based analysis of the mentioned property types as an alternative based on requiring explicitly extending a model with monitors expressed in POOSL. Predefined monitors for evaluating common types of long-run average metrics include accuracy analysis based on confidence intervals [Theelen 2004], where the estimation results are proven to converge to exactly the same results obtainable with model checking. SHE focuses on the evaluation of individual design alternatives. However, feeding the performance results obtained from models into approaches like those exploited in [Noonan and Flanagan 2006; Gries 2004; Thompson et al. 2007] would facilitate fully automated design-space exploration.

The PREMADONA tool utilises the simulation-based analysis techniques of SHE to enable evaluating various performance metrics as specified by the user in XML, see [Theelen 2008]. PREMADONA can add monitors to an application model for evaluat-

ing throughput, latency (minimum, maximum, average, variance), response delay, buffer occupancy (maximum, average, variance) and deadline miss probabilities. For platforms, it can add monitors to evaluate for example processor utilisation, memory occupancy (maximum, average, variance), communication load (maximum, average, variance) and power consumption (peak, average). PREMADONA could easily be extended to add monitors for evaluating any other metric of interest.

4. COMPARISON

This section compares the various design decisions and approaches used for implementing the methodologies discussed in the previous Sections. Table II[1][2][3] lists the features of Metropolis, DOL-MPA and the Y-chart variant of SHE.

4.1 Abstraction and Refinement

An important aspect of developing models during design is the necessity to abstract from implementation details [Theelen 2004]. Although literature mostly emphasises the advantage of increasing analysis efficiency, the key advantage of abstraction is actually the ability to focus on answering specific design questions. In case of the Y-chart paradigm, the focus is on performance-related questions and hence, one should abstract implementation details that do not affect performance to a great extend. Another reason for abstraction is the fact that the system is actually being designed and hence the implementation details are still to be decided.

The inherent difficulty of model-driven engineering is to make good abstractions such that a model properly represents the system, while still being able to answer the design questions of interest. On one hand, abstraction urges discarding many implementation details, while on the other hand, obtaining credible analysis results requires including those aspects that impact the performance (in our case). Models that properly capture all relevant aspects affecting the performance are sometimes called *adequate* [Theelen 2004]. Notice that adequacy is a property of a model, while *accuracy* is a property of an analysis result. The accuracy of a result depends highly on the type of analysis technique that is used for deriving the result; approximation, estimation or heuristic approaches will give less accurate results than exact techniques. Notice that adequacy of a model and accuracy of results are two orthogonal concepts. A 100% adequate model can give very inaccurate results for example due to combining it with inappropriate analysis techniques or by relying on simulations that ran way too short for the modeled behaviour to stabilise in the operation mode of interest. Conversely, a model can be very inadequate while analysing it gives 100% accurate results by using exact analysis techniques. Any model-driven engineering exercise includes a point in time where a constructed model must be considered as being adequate such that the analysis phase can start.

[1]Although Metropolis supports any behavior, resource, scheduling to be modeled in MMM, the table lists only those aspects for which elements are available in the Meta-Model library.
[2]Although DOL supports any behaviour to be specified, the table lists only features relevant to modeling and analysing systems with MPA
[3]Although SHE supports any behavior to be modeled in POOSL, the table lists only those aspects for which modeling patterns have been developed. The table also lists only those performance metrics for which the PREMADONA tool allows automatic addition of monitors to a model.

		Metropolis	*DOL-MPA*	*Y-chart Variant of SHE*
Application	*Supported MoCs*	YAPI, TTL and multi-rate synchronous dataflow	Task Graphs for Real-Time Systems	SDF, CSDF, SADF and Task Graphs for Real-Time Systems
Platform	*Supported Resources*	n/a	Processor, Communication	Processor, Communication, Storage, Energy
	Supported Schedulers	n/a	Preemptive and non-preemptive Fixed Priority and Rate Monotonic, preemptive Earliest Deadline First, First-Come First-Serve, Generalised Processor Sharing, Time Division Multiple Access and any hierarchical combinations of these	Preemptive and non-preemptive policies, including Round Robin (with or without skipping), Earliest Deadline First, First-Come First-Serve, Rate Monotonic, Generalised Processor Sharing and Time Division Multiple Access
	Refinement	Model element substitution	Model refinement with Real-Time Interfaces which ensure preservation of non-functional properties such as timing	SHE includes a model refinement approach ensuring preservation of functionality and timing, which is not used by the Y-chart variant
	Interconnection Type	Explicit platform interconnection structure	Implicit platform interconnection structure	Implicit platform interconnection structure
Mapping	*Approach*	LTL expressions extended with value mapping between application and platform events	Binding specification which defines mappings between processes and processor, software channels and communication paths. Resource streams are inputs for performance components	Service request - acknowledgement interactions between application and platform models
	Mapped Elements	Computations only	Computations and communications	Computations and communications
Analysis	*Application*	Throughput, end-to-end delays, buffer occupancy	Throughput, end-to-end delays, buffer occupancy	Throughput, response delay, inter-firing latency, buffer occupancy, deadline miss probabilities
	Platform	Processor load, utilization of communication resources, memory occupancy	Processor load, utilization of communication resources, memory occupancy	Processor load, utilization of communication resources, memory occupancy, power consumption
	Type	Simulation: Worst, best and average case. LTL and LOC monitors. Model checking : LTL via SPIN LOC via Checker	Worst and best case	Simulation-based analysis of functional correctness and worst/best-case, average case and probabilistic/expected reachability performance metrics. SHE has model-checking techniques for these metrics but there are no automated tools
	Accuracy	All results from model checking are exact. All results from simulation are approximations.	Hard upper and lower bounds	Observation results for functional correctness and worst/best case and probabilistic/expected reachability performance metrics. Confidence intervals with accuracy result for average-case performance metrics
Exploration	*Type*	Manual exploration via alternative platforms and alternative mappings	Use of DOL; Manual exploration or automatic with an optimization framework such as [Bleuler et al. 2003] based on multi-objective evolutionary algorithms	Manual exploration or the approach of [Noonan and Flanagan 2006] based on evolutionary algorithms

Table II. A comparison of Y-chart based design methodologies

All three methodologies in Table II focus on the evaluation of performance properties and therefore prescribe or require abstraction from functional details and the actual content of data that is being processed. Both Metropolis and SHE would in principle allow refining these aspects to complete implementation details, but

they encourage a designer to capture only those aspects that (potentially) affect the performance. By relying on the Real-Time Calculus, MPA goes even a step further in disabling the possibility of specifying implementation details completely. As a consequence, developing adequate models in MPA may be more difficult than when using Metropolis or SHE but the advantage is a better analysis efficiency and scalability. Conversely, all three methodologies support model refinement towards a more detailed specifications. Both SHE and MPA include formal techniques for model refinement by means of decomposing model components into a collection of more detailed components. Metropolis also supports such model refinement but only certain approaches can guarantee that properties don't change by refinement. SHE furthermore includes formal techniques for synthesising real-time control software on single-processor platforms that guarantee preservation of functionality and performance as specified and analyzed in a model. This work needs however further extension to allow application in the MPSoC setting of the Y-chart paradigm.

Popular abstraction mechanisms are the use of probabilistic distributions and non-determinism. Both these mechanisms allow abstract specification of choices. As opposed to non-determinism, using stochasticity requires knowledge about the relative occurrence of each possible alternative. The ability to use these mechanisms strongly depends on the formalisms underlying a methodology (see also Table I). Both Metropolis and SHE support the use of non-deterministic choices between alternative behaviours or alternative data items. Metropolis offers the use of non-determinate variables to express the latter. However, such variables become concrete data items when interpreting a mapping specification for a system. POOSL allows non-determinism between alternative data values by using different assignments (possibly of different types of data) in a non-deterministic behavioural choice. Non-deterministic behaviour is supported by a specific language primitive in POOSL, which concerns a selection between alternative actions. MMM includes a similar language primitive. There is however also a difference between the non-deterministic choice in MMM and POOSL. In case no alternatives are enabled for the non-deterministic choice in MMM, the overall behaviour will block forever. In POOSL, a similar situation may occur but other behaviour running in parallel with the blocked non-deterministic behaviour may unblock one or more of the alternatives. For MPA, non-determinism is inherently present in the arrival/service curves due to abstracting from the exact moments in time that data is communicated.

The use of probabilities is only supported in the SHE methodology. The SADF application models accepted by the PREMADONA tool are an example of where probabilistic choices between alternative behaviour and timing can be specified.

4.2 Explicit versus Implicit Resource Interconnects

Table II states that Metropolis, DOL-MPA and the Y-chart variant of SHE use different ways to specify the topology of platform resources as shown in Figure 14.

Platform models in Metropolis have an explicit resource topology, see top half of Figure 14. Hence, the models explicitly capture how for example processor resources are interconnected via communication resources. Because of this explicit topology, the mapping specification only defines the correspondences between computations in the application model and processors in the platform model. The path that messages take between computations is implicitly defined by the chain

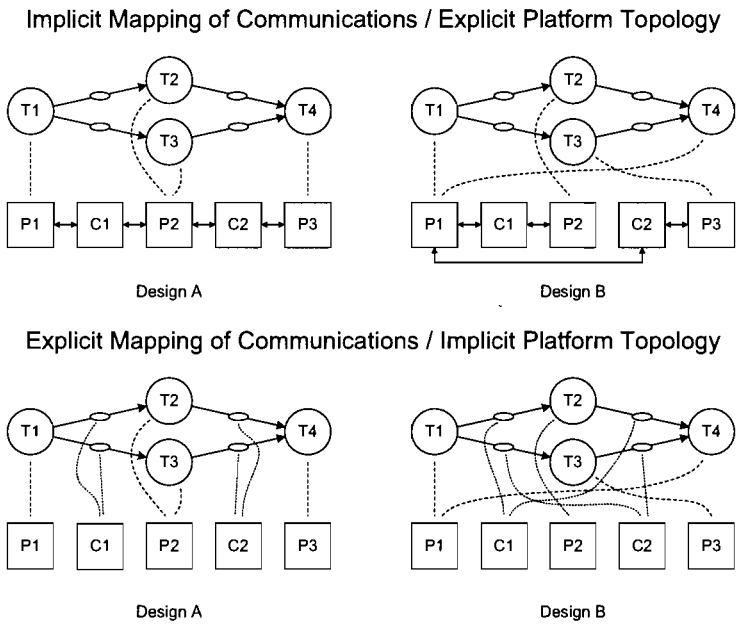


Fig. 14. Approaches to capture mapping and the topology of resources in platforms

of communication resources between the processors that execute the computations and the mapping. This approach has the benefit that it is easy to define complex (hierarchal) combinations of communication or storage resources, where busses or memories serve other busses and memories respectively, while having a small mapping specification. The disadvantage is that changing the mapping may require to change the platform model as well in order to ensure that resource interconnections are consistent with dependencies between computations and the chosen mapping.

On the other hand, the Y-chart variant of SHE uses platform models that do not contain any topological information. This approach is illustrated in the lower half of Figure 14. The mapping specification contains a binding for both computations and communications of an application. Hence, the topology of how the processor and communication resources are interconnected emerges from the mapping specification. This gives much flexibility when evaluating alternative mappings since there is no need to change the platform model to ensure that resource interconnections are consistent with dependencies between computations and the chosen mapping. However, in case complex (hierarchical) combinations of communication or storage resources are to be considered, one would need to introduce artificial elements in the application model that are not part of the real application. These would represent the tasks of bridging protocols between communication resources or the transfer of data between two storage resources respectively. The reason for the need to introduce these artifacts in the application model is that the mapping rules do not allow distributing for example a communication in the application over multiple communication resources such as a hierarchy of busses.

Platform models in the context of DOL-MPA are closer to the approach taken by Metropolis. Platform models define explicitly the resources which are available such as processors, memories and their interconnections. Moreover, end-to-end communication paths with nodes on the affected resources are used in order to model communication. This allows networks-on-chip to be modeled adequately. As discussed earlier, mapping must be done in the spatial domain as well as in the temporal domain. The spatial domain mapping specification explicitly defines the binding between processes and software channels to their corresponding processors and communication paths. This is very similar to Metropolis but with the extra task of mapping the communication paths. The approach has the advantage of explicitly stating which communication path should be used between two resources when multiple paths are present. This facilitates end-to-end communication performance analysis. The temporal mapping specification explicitly defines the scheduling policy on each resource and the corresponding parameters. DOL-MPA differs from Metropolis and the Y-chart variant of SHE in that scheduling is part of the mapping and not part of the platform model.

4.3 Exact Analysis versus Simulation-Based Analysis

According to Table II, Metropolis, DOL-MPA and the Y-chart variant of SHE accomplish the fourth step of the Y-chart paradigm (See Section 2.1) on evaluating a proposed design solution in different ways. The considered approaches can be categorised in exact analysis and simulation-based analysis. Metropolis and SHE offer state-space exploration based analysis techniques for correctness and performance properties, while DOL-MPA supports exact analysis based on the Real-Time Calculus [Thiele et al. 2000; Chakraborty et al. 2003]. Metropolis utilises traditional model checking techniques for determining whether LTL and LOC properties are satisfied. For exact analysis of functional correctness, SHE follows the approach of linear-time temporal logic verification, where typically the logical negation of a required property is converted into an automaton by means of a so-called tableau construction [Geilen 2001]. Subsequently, automata theoretic techniques are used to test whether the property is satisfied. Although the properties that can be model checked include certain types of timing properties, the traditional approach is not suitable for amongst others long-run average performance metrics like throughput. To compute exact results for such metrics, SHE supports a more liberal form of model checking that relies on Markov theory [Theelen et al. 2007; Theelen 2004]. Nevertheless, SHE does not include automatic tools to actually perform these calculations as opposed to Metropolis. An important disadvantage of state-space based analysis is that it does not scale to large systems. MPA circumvents this problem by using Real-Time Calculus as an alternative exact analysis approach. The Real-Time Calculus is derived from the Max-Plus algebra [Baccelli et al. 1992], which allows computing the latest moment in time at which events (e.g., the arrival of data) can occur. Using this as a basis, analysis with MPA is limited to determining performance bounds.

Since Metropolis and SHE suffer from state-space explosion issues when using their exact analysis techniques, they offer simulation-based analysis as a scalable alternative. However, simulations are never exhaustive (in general) and the obtained results are only valid for that part of the state-space that is actually covered

during a simulation. Hence, simulation-based analysis gives estimations of the actual performance of a system. An essential problem is therefore how long a simulation should run before the results are considered accurate. While Metropolis does not provide support to evaluate the accuracy of results, SHE includes techniques to evaluate the accuracy of results for any type of long-run average performance metric based on confidence intervals [Theelen 2004]. Another crucial feature of SHE is that the simulation-based estimation results converge to the same results that would be obtainable with its exact techniques. This originates from founding both approaches on the same mathematical model defined by the formal semantics of POOSL. Informal simulation-based approaches (including those for Metropolis) cannot guarantee such correspondence between exact and estimation results.

5. CONCLUSION AND FUTURE TRENDS

Embedded systems become more complex with each generation. A good portion of this complexity originates from the growing level of heterogeneity. Modern systems include a variety of components types that are not electronic such as micro-mechanical and micro-optical. In addition, there is a certain momentum in the industry toward IP based design. In order to adequately support these tendencies, implementations of the Y-chart paradigm will have to support MoCs and MoAs with sufficient expressivity. New MoCs or adaptations of existing ones will have to be developed. A long-standing challenge in this area are MoCs supporting the integration of various abstraction levels. On the other hand, MoAs are currently fairly immature - most are adaptations of existing MoCs. Hence, developing sufficiently expressive MoAs is an interesting challenge for future research.

Application of the Y-chart paradigm to domains other than streaming multi-media systems is a subject of current research. The domain of high-tech mechatronic systems is a good example that could benefit for the Y-chart paradigm. Problems that originally arose in streaming multi-media systems are now also observed in these types of systems, where feedback and feedforward control strategies have to be implemented with tight performance requirements.

Most current Y-chart based design methodologies focus on the tasks of modeling and analysis. However, in order to meet the objective of shortening time-to-market, a new generation of methodologies are required, which incorporate extensive synthesis capabilities. The main challenge to address is the issue of heterogeneity. Analysis models are of a different nature than synthesis models. They are optimised for efficient determination of properties of interest and therefore only contain the essential information. The focus of synthesis is efficient implementation. Synthesis models are therefore more detailed, but they also need to be complete. In general analysis models cannot simply be refined into synthesis models, making it difficult to establish proper relations between them. Understanding these relations is a prerequisite to integrate them into seamless design flows.

This article explores the richness of how design methodologies have implemented the Y-chart paradigm . The Y-chart paradigm was developed more then 10 years ago in order to address the challenges surrounding the exploration of the design-space of streaming systems. It is defined by a simple Y-shaped sequence of tasks (functional application modeling, platform architecture modeling, mapping, evalu-

ation and synthesis). The philosophy of separating functional design and platform design offers a very effective solution to design problems, where alternative deployments of platforms are an essential aspect. Despite the simplicity and straightforwardness of the Y-chart paradigm, many variations exist in implementing it in design methodologies. Key areas of differences are (i) the supported MoCs and MoAs, (ii) the approaches used for mapping, and (iii) the type of metrics that can be analyzed as well as the level of accuracy of results.

The comparison has highlighted that each design approach has its advantages and inconveniences. Metropolis has the advantage of offering a simple yet very powerful mapping approach by using LTL expressions. Moreover, by using LTL expressions, application and platform models are truly kept separate. A relative weakness of Metropolis vs the other implementations is the informal nature of models which limits evaluation to mostly simulation. The Y-Chart variant of SHE/POOSL offers a great deal of expressivity due to the POOSL language and simulation analysis because of SHE. Moreover, integration with other design paradigms it possible because the implementation of the Y-Chart approach is achieved in the context of a broader methodology framework (SHE). A relative weakness of the implementation is that the application models must make explicit calls to the platforms models which bind them. Moreover, the implicit platform topologies of the platform because harder to use for complex cases. DOL-MPA has the advantage of offering exact analysis through the means of Real-Time Calculus on models with no implementation details. However, the use of Real-time Calculus has the disadvantage of being harder to use.

REFERENCES

ALUR, R. 1991. Techniques for Automatic Verification of Real-Time Systems. Ph.D. thesis, Stanford University, California, USA.

BACCELLI, F., COHEN, G., OLSDER, G., AND J.P.QUADRAT. 1992. *Synchronization and Linearity.* John Wiley & Sons.

BALARIN, F., BURCH, J., LAVAGNO, L., PASSERONE, R., SANGIOVANNI-VINCENTELL, A., AND WATANABE, Y. 2001. Constraints Specification at Higher Levels of Abstraction. In *Proceedings of HLDVT.* IEEE, 129–133.

BALARIN, F., WATANABE, Y., HSIEH, H., LAVAGNO, L., PASSERONE, C., AND SANGIOVANNI-VINCENTELLI, A. 2003. Metropolis: An Integrated Electronic Design Environment. *IEEE Computer 36,* 4, 45–52.

BALARIN, F., G. P. J. A. P. C. S. E. T. B. C. M. H. H. L. L. S.-V. A. S. K. 1997. *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach.* Springer.

BALEANI, M., FERRARI, A., SANGIOVANNI-VINCENTELLI, A., AND TURCHETTI, C. 2000. HW/SW Codesign of an Engine Management System. In *Proceedings of DATE.* IEEE, 263–267.

BEZIVIN, J. 2005. On the Unification Power of Models. *Software and System Modeling 4,* 2, 171–188.

BLEULER, S., LAUMANNS, M., THIELE, L., AND ZITZLER, E. 2003. PISA - A Platform and Programming Language Independent Interface for Search Algorithms. In *Evolutionary Multi-Criterion Optimization (EMO 2003).* Springer Verlag, Faro, Portugal, 494 – 508.

BOKHOVEN, L. V. 2002. Constructive Tool Design for Formal Languages: From Semantics to Executing Models. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

BRAND, D. AND ZAFIROPULO, P. 1983. On Communicating Finite-State Machines. *Journal of the ACM 20,* 2, 323–342.

BROOKS, C., LEE, E., LIU, X., NEUENDORFFER, S., ZHAO, Y., AND ZHENG, H. 2005. Heterogeneous

Concurrent Modeling and Design in Java (Volume 1: Introduction to Ptolemy II). Tech. Rep. UCB/ERL M05/21, University of California, Berkeley, USA.

BURCH, J., PASSERONE, R., AND SANGIOVANNI-VINCENTELLI, A. 2001. Overcoming Heterophobia: Modeling Concurrency in Heterogeneous Systems. In *Proceedings of ACSD*. IEEE, 13–32.

CESARIO, W., NICOLESCU, G., GAUTHIER, L., LYONNARD, D., AND JERRAYA, A. 2001. Colif: A Design Representation for Application-Specific Multiprocessor SoCs. *Design & Test of Computers 18*, 5, 8–20.

CHAKRABORTY, S., KÜNZLI, S., AND THIELE, L. 2003. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs. In *Design Automation and Test in Europe (DATE)*. IEEE Press, Munich, Germany, 190–195.

CHAKRABORTY, S., KÜNZLI, S., THIELE, L., HERKERSDORF, A., AND SAGMEISTER, P. 2003. Performance Evaluation of Network Processor Architectures: Combining Simulation with Analytical Estimation. *Computer Networks 41*, 5 (April), 641–665.

CHAKRABORTY, S., MITRA, T., ROYCHOUDHURY, A., THIELE, L., BORDOLOI, U., AND DERDIYOK, C. 2007. Cache-Aware Timing Analysis of Streaming Applications. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems*. IEEE Computer Society Washington, DC, USA, 159–168.

CHEN, X., DAVARE, A., HSIEH, H., SANGIOVANNI-VINCENTELLI, A., AND WATANABE, Y. 2005. Simulation based deadlock analysis for system level designs. In *Proceedings of DAC*. IEEE, 260–265.

CRUZ, R. 1991. A Calculus for Network Delay, Part I: Network Elements in Isolation. *IEEE Transactions on Information Theory 37*, 1, 114–131.

DAVARE, A., DENSMORE, D., MEYEROWITZ, T., PINTO, A., SANGIOVANNI-VINCENTELLI, A., YANG, G., ZENG, H., AND ZHU, Q. 2007. A Next-Generation Design Framework for Platform-Based Design. In *Proceedings of DVCon*. IEEE.

DENSMORE, D. 2004a. Formal Refinement Verification in Metropolis. Tech. rep., University of California, Berkeley, USA. UCB/ERL M04/10.

DENSMORE, D. 2004b. Metropolis Architecture Refinement Styles and Methodology. Tech. rep., University of California, Berkeley, USA. UCB/ERL M04/36.

DENSMORE, D., PASSERONE, R., AND SANGIOVANNI-VINCENTELLI, A. 2006. A Platform-Based Taxonomy for ESL Design. *IEEE Design & Test of Computers 23*, 5, 359–374.

DENSMORE, D., REKHI, S., AND SANGIOVANNI-VINCENTELLI, A. 2004. Microarchitecture Development via Metropolis Successive Platform Refinement. In *Proceedings of DATE*. IEEE, 10346–10357.

FLORESCU, O. 2007. Predictable Design for Real-Time Systems. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

FLORESCU, O., VOETEN, J., VERHOEF, M., AND CORPORAAL, H. 2007. Reusing Real-Time Systems Design Experience through Modeling Patterns. In *Advances in Design and Specification Languages for Embedded Systems*, S. Huss, Ed. Springer, Chapter 20, 329–348.

GAJSKI, D. AND KUHN, R. H. 1983. New VLSI Tools. *Computer*, 11–14.

GAJSKI, D., SHU, J., RAIDER, D., GERSTLAUER, A., AND ZHAO, S. 2000. *SpecC: Specification Language and Methodology*. Springer.

GEILEN, M. 2001. On the Construction of Monitors for Temporal Logic Properties. In *Proceedings of the 1st Workshop on Runtime Verification (RV'01)*.

GEILEN, M. 2002. Formal Techniques for Verification of Complex Real-Time Systems. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

GHAMARIAN, A. 2008. Timing Analysis of Synchronous Data Flow Graphs. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

GOESSLER, G. AND SANGIOVANNI-VINCENTELLI, A. 2002. Compositional Modeling in Metropolis. In *Proceedings of EMSOFT*. Springer, 93–107.

GOLDSCHMIDT, S. AND HENNESSY, J. 1993. The Accuracy of Trace-Driven Simulations of Multiprocessors. *Performance Evaluation Review 21*, 1, 146–157.

GRIES, M. 2004. Methods for Evaluating and Covering the Design Space during Early Design Development. *Integration, the VLSI Journal 38,* 2, 131–183.

GRIES, M. AND KEUTZER, K. 2005. *Building ASIPs: The Mescal Methodology.* Springer.

GROTKER, T., LIAO, S., MARTIN, G., AND SWAN, S. 2002. *System Design with SystemC.* Kluwer Academic Publications.

H. CHANG, L. COOKE, H. H. G. M. A. M. AND TODD, L. 1999. *Surviving the SOC Revolution : A Guide to Platfrom-Based Design.* Klumer Academic.

HAMANN, A., JERSAK, M., RICHTER, K., AND ERNST, R. 2004. Design Space Exploration and System Optimization with SymTA/S - Symbolic Timing Analysis for Systems. In *Proceedings of RTSS.* IEEE, 469–478.

HOLZMANN, G. 1991. *Design and Validation of Computer Protocols.* Prentice-Hall.

HUANG, J. 2005. Predictability in Real-Time System Design. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

KIENHUIS, B. 1999. Design Space Exploration of Stream-based Dataflow Architectures: Methods and Tools. Ph.D. thesis, Delft University of Technology, Delft, Netherlands.

KIENHUIS, B., DEPRETTERE, E., VISSERS, K., AND WOLF, P. V. 1997. An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures. In *Proceedinsg of ASAP.* IEEE, 338–349.

KIENHUIS, B., DEPRETTEREL, E., VISSERS, K., AND VAN DER WOLF, P. 1997. An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures. *ASAP*, 338–349.

KIENHUIS, B., RIJPKEMA, E., AND DEPRETTERE, E. 2000. Compaan: Deriving Process Networks from Matlab for Embedded Signal Processing Architectures. In *Proceedings of CODES.* IEEE, 13–17.

KOCK, E. D., ESSINK, G., SMITS, W., WOLF, P. V., BRUNEL, J., KRUIJTZER, W., LIEVERSE, P., AND VISSERS, K. 2000. YAPI: Application Modeling for Signal Processing Systems. In *Proceedings of DAC.* ACM, 402–405.

KOYMANS, K. 1990. Specifying Real-Time Properties with the Metric Temporal Language. *Real-Time Systems 2,* 4, 255–299.

KROLIKOSKI, S. J., SCHIRRMEISTER, F., SALEFSKI, B., ROWSON, J., AND MARTIN, G. 1999. Methodology and Technology for Virtual Component Driven Hardware/Software Co-Design on the System-Level. *ISCAS*, 456–459.

KRUIJTZER, W., WOLF, P., KOCK, E., STUYT, J., ECKER, W., MAYE, A., HUSTIN, S., AMERIJCKX, C., PAOLIAND, S., AND VAUMORIN, E. 2008. Industrial IP Integration Flows based on IP-XACT Standards. In *Proceedings of DATE.* IEEE, 26–31.

KUMAR, A., MESMAN, B., THEELEN, B., AND CORPORAAL, H. 2006. Resource Manager for Non-Preemptive Heterogeneous Multiprocessor System-on-Chip. In *Proceedings of ESTIMedia.* IEEE, 33–38.

KWIATKOWSKA, M., NORMAN, G., SEGALA, R., AND SPRONTSTON, J. 2002. Automatic Verification of Real-Time Systems with Discrete Probability Distributions. *Theoretical Computer Science 282,* 1, 101–150.

LARSEN, K. 1997. UPPALL in a Nutshell. *Journal on Software Tools for Technology Transfer 1,* 1-2, 134–152.

LEE, E. AND MESSERSCHMITT, D. 1987. Synchronous Data Flow. *IEEE Proceedings 75,* 9, 1235–1245.

LEE, E. AND SANGIOVANNI-VINCENTELLI, A. 1998. A framework for Comparing Models of Computation. *Transactions on Computer-Aided Design of Integrated Circuits and Systems 17,* 12, 1217–1229.

LIEVERSE, P., WOLF, P. V. D., VISSERS, K., AND DEPRETTERE, E. 2001. A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology 29,* 3, 197–207.

MARTIN, G. 1998. Design Methodologies for System Level IP. In *Proceedings of DATE.* IEEE, 286–289.

MEDVIDOVIC, N. AND TAYLOR, R. 2000. A Classification and Comparison Framework for Software Architecture Descripton Languages. *Transactions on Software Engineering 26,* 1, 70–93.

MEYEROWITZ, T. 2004. Metropolis ARM CPU Examples. Tech. Rep. UCB/ERL M04/39, University of California, Berkeley, USA.

MISHRA, P. AND DUTT, N. 2008. *Processor Description Languages: Applications and Methodologies.* Morgan Kaufmann.

NOONAN, L. AND FLANAGAN, C. 2006. Utilising Evolutionary Approaches and Object-Oriented Techniques for Design-Space Exploration. In *Proceedings of DSD.* IEEE, 346–352.

PINTO, A. 2004. Metropolis Design Guidelines. Tech. Rep. UCB/ERL M04/40, University of California, Berkeley, USA.

PUTTEN, P. V. D. AND VOETEN, J. 1997. Specification of Reactive Hardware/Software Systems. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

QIN, W. AND MALIK, S. 2002. Architecture Description Languages for Retargetable Compilation. In *The Compiler Design Handbook: Optimizations & Machine Code Generation.* CRC Press, 535–562.

SANGIOVANNI-VINCENTELLI, A. AND MARTIN, G. 2001. Platform-Based Design and Software Design Methodology for Embedded Systems. *Design and Test of Computers 18,* 6, 21–33.

SGROI, M., SHEETS, M., MIHAL, A., KEUTSER, K., MALIK, S., RABAEY, J., AND SANGIOVANNI-VINCENTELLI, A. 2001. Adressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design. In *Proceedings of DAC.* IEEE, 667–672.

STUIJK, S. 2007. Predictable Mapping of Streaming Applications on Multiprocessors. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

TEICH, J., BLICKLE, T., AND THIELE, L. 1998. System-Level Synthesis Using Evolutionary Algorithms. *J. Design Automation for Embedded Systems 3,* 23–58.

THEELEN, B. 2004. Performance Modelling for System-Level Design. Ph.D. thesis, Eindhoven University of Technology, Eindhoven, Netherlands.

THEELEN, B. 2007. A Performance Analysis Tool for Scenario-Aware Streaming Applications. In *Proceedings of QEST.* IEEE, 269–270.

THEELEN, B. 2008. Performance Model Generation for MPSoC Design-Space Exploration. In *Proceedings of QEST.* IEEE.

THEELEN, B., FLORESCU, O., GEILEN, M., HUANG, J., PUTTEN, P. V., AND VOETEN, J. 2007. Software/Hardware Engineering with the Parallel Object-Oriented Specification Language. In *Proceedings of MEMOCODE.* IEEE, 139–148.

THIELE, L., BACIVAROV, I., HAID, W., AND HUANG, K. 2007. Mapping Applications to Tiled Multiprocessor Embedded Systems. In *Proc. 7th Intl Conference on Application of Concurrency to System Design (ACSD 2007).* IEEE Computer Society, Bratislava, Slovak Republic, 29–40.

THIELE, L., CHAKRABORTY, S., AND NAEDELE, M. 2000. Real-time Calculus for Scheduling Hard Real-time Systems. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS).* Geneva, Switzerland, 101–104.

THIELE, L., WANDELER, E., AND STOIMENOV, N. 2006. Real-Time Interfaces for Composing Real-Time Systems. In *International Conference On Embedded Software (EMSOFT 06).* Seoul, Korea, 34–43.

THOMPSON, M., NIKOLOV, H., STEFANOV, T., PIMENTEL, A., ERBAS, C., POLSTRA, S., AND DEPRETTERE, E. 2007. A Framework for Rapid System-Level Exploration, Synthesis, and Programming of Multimedia MP-SoCs. In *Proceedings of CODES/ISSS.* IEEE, 139–148.

VOETEN, J. 2002. Performance Evaluation with Temporal Rewards. *Performance Evaluation 50,* 2/3, 189–218.

WANDELER, E. AND THIELE, L. 2006a. Interface-Based Design of Real-Time Systems with Hierarchical Scheduling. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).* San Jose, USA, 243–252.

WANDELER, E. AND THIELE, L. 2006b. Real-Time Calculus (RTC) Toolbox.

WANDELER, E. AND THIELE, L. 2007. Workload Correlations in Multi-processor Hard Real-time Systems. *Journal of Computer and System Sciences 73,* 2 (March), 207–224.

WANDELER, E., THIELE, L., VERHOEF, M., AND LIEVERSE, P. 2006. System Architecture Evaluation using Modular Performance Analysis: A Case Study. *International Journal on Software Tools for Technology Transfer 19*, 649–667.

WIJK, F. V., VOETEN, J., AND BERG, A. T. 2003. An Abstract Modeling Approach Towards System-Level Design-Space Exploration. In *System Specification and Design Languages*, E. Villar and J. Mermet, Eds. Kluwer Academic Publishers, Chapter 22, 267–282.

WILHELM, R., ENGBLOM, J., ERMEDAHL, A., HOLSTI, N., THESING, S., WHALLEY, D., BERNAT, G., FERDINAND, C., HECKMANN, R., MUELLER, F., PUAUT, I., PUSCHNER, P., STASCHULAT, J., AND STENSTRM, P. 2008. The Determination of Worst-Case Execution Times—Overview of the Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems (TECS) 7*, 3 (April).

YANG, G., HSIEH, H., CHEN, X., BALARIN, F., AND SANGIOVANNI-VINCENTELLI, A. 2006. Constraints Assisted Modeling and Validation in Metropolis Framework. In *Proceedings of Asilomar Conference on Signal, Systems and Computers*. IEEE, 1469–1474.

ZENG, H., DAVARE, A., SANGIOVANNI-VINCENTELLI, A., SONALKAR, S., KANAJAN, S., AND PINELLO, C. 2006. Design SPace Exploration of Automotive Platforms in Metropolis. In *Scociety of Automotive Engineers Congress*.

Received Month Year; Revised Month Year; accepted Month Year

Une nouvelle méthodologie pour la conception d'outils CAO

Malgré tous les efforts investis par l'industrie et le milieu académique, les concepteurs de systèmes embarqués ont besoin de nouvelles solutions pour la modélisation et la simulation. Ce besoin est une conséquence du fait qu'il n'existe toujours pas de solution bien intégrée répondant à un ensemble de critères clés nécessaires pour l'obtention d'une solution de modélisation et de simulation efficace et flexible. Ces critères sont :

- permettre la spécification de composantes logicielles de manière simple ainsi que l'intégration de celles-ci dans la spécification globale d'un système embarqué;
- offrir des syntaxes de modélisation et de programmation simples permettant la définition de modèle moins sujets à l'erreur, facilitant la spécification de systèmes complexes ainsi que leur réutilisation [26];
- offrir des fonctionnalités d'introspection pour faciliter l'analyse et de débogage de spécifications complexes;
- permettre l'annotation de modèles pour divers besoins tels que pour diriger les outils de synthèse et guider l'intégration d'outils de vérification [62];
- supporter la transformation de modèles et de spécifications en format intermédiaire standardisé afin de permettre la conception d'outils CAO de manière agnostique des langages de description [51]. Ces formats doivent être assez riches pour supporter les outils durant toutes les activités de conception (simulation, vérification, raffinement, etc.);
- être multi-plateformes et multi-langages afin de décrire des systèmes à base de composantes hétérogènes [41];
- offrir la gestion de mémoire afin d'accélérer le processus de spécification et pour éliminer une source importante d'erreurs [65]
- intégrer avec des environnements de conception distribués basé sur le Web afin de permettre de la conception coopération à distance et du traitement distribué [22].

Malgré leur popularité et leurs fonctionnalités, les environnements de conception tels que SystemC et SystemVerilog ont plusieurs lacunes fondamentales. Ils ne supportent

qu'un seul langage et ont des capacités très limitées à rendre leurs modèles et leurs spécifications disponibles à des outils externes. La programmation sujette aux erreurs, et le manque d'un système de typage strict et de l'introspection dans C++ constituent un obstacle dans le développement avec SystemC. SystemVerilog n'est pas un environnement dont les sources sont ouvertes donc il est difficile de construire des outils CAO autour de celui-ci pour supporter des méthodologies de conceptions personnalisés.

En 2000, un an après le déploiement de SystemC, le .Net Framework a été annoncé. Cette technologie possède des caractéristiques et des fonctionnalités qui auraient sûrement influencé le choix d'utiliser de C++ pour la construction de SystemC. La plateforme .Net a introduit le langage C#. Une excellente comparaison entre C#, C++ et Java est faite dans [3]. Cette comparaison démontre que C# est conçu avec les forces de Java et ceux de C++ afin de d'offrir un langage puissant et élégant. Plusieurs des fonctionnalités prévues pour SystemC et la deuxième version de SystemVerilog sont déjà présentes dans C#. Un avantage clé de la plateforme .Net est sa capacité de supporter plusieurs langages [59][67], une caractéristique important pour la conception et la modélisation des systèmes embarqués.

Les contributions principales de cet article sont:

- l'introduction des technologies .Net à la communauté des systèmes embarqués;
- la définition d'une nouvelle taxonomie pour les représentations de modèles;
- la présentation de l'importance d'aborder les flux de conception de système selon une perspective de représentation de modèle, ainsi que l'influence de cette perspective sur la conception d'outils CAO;
- la définition d'une nouvelle méthodologie de conception d'outils CAO pour les systèmes embarqués. Cette méthodologie met beaucoup l'accent sur la représentation de modèles, leur consommation ainsi que leur transformation. De plus, elle repose sur la technologie .Net. Cette méthodologie offre plusieurs bénéfices tels que son efficacité, mais aussi, par son emploi, elle procure des caractéristiques importantes aux outils conçus tels que le support multi-langage, l'utilisation d'un format standardisé ouvert riche en métadonnées et l'introspection;
- la construction d'un outil CAO pour la modélisation et la simulation de systèmes embarqués dont la conception de celui-ci suit cette nouvelle

méthodologie. Cet outil, au nom de « Embedded Systems with .Net » (eSys.Net), offre la majorité des fonctionnalités de SystemC avec une performance similaire. De plus, elle comporte plusieurs avantages importants sur SystemC.

Les pages suivantes contiennent une copie de l'article [49], dans son format original (sauf la numérotation des pages), publié dans *ACM Transactions on Embedded Computing Systems Special Issue on Concurrent Hardware-Software Design Methods for MPSoC, vol 5, Num 2, 2006.*

A New Efficient EDA Tool Design Methodology

JAMES LAPALME
Université de Montréal
EL MOSTAPHA ABOULHAMID
Université de Montréal
and
GABRIELA NICOLESCU
Ecole Polytechnique de Montréal

New sophisticated EDA tools and methodologies will be needed to make products viable in the future marketplace by simplifying the various design stages. These tools will permit system design at a high abstraction level and enable automatic refinement through several abstraction levels to obtain a final prototype. They will have to be based on representations that are clean, complete, and easy to manipulate. In order to develop these new EDA tools, key features such as standardization, metadata programming, reflectivity, and introspection are needed. This work proposes a .Net Framework-based methodology, which possesses all these required key features. This methodology simplifies specification, synthesis, and validation of systems and enables the efficient creation/customization of EDA tools at low cost and development time. We show the effectiveness of this methodology by presenting its application for the design of a new EDA tool called ESys .Net (Embedded System design with .Net). We emphasize the specification and simulation aspects of this tool.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Hardware description languages, Simulation, Verification, VHDL, Verilog*; D.3.2 [**Programming Languages**]: Language Classifications—*Design languages, C#, C++, Concurrent, distributed, and parallel languages*; I.6.2 [**Simulation and Modeling**]: Simulation Languages; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Environments*

General Terms: Algorithms, Documentation, Performance, Design, Reliability, Experimentation, Standardization, Languages, Verification.

Additional Key Words and Phrases: ESys.Net, .Net Framework, C#, SystemC, VHDL, attribute programming, SoC, CoDesign, embedded systems, modeling and simulation.

Authors' addresses: James Lapalme and El Mostapha Aboulhamid, Université de Montréal, CP 6128, Succ. Centre-Ville, Montréal, Québec, Canada, H3C 3J7; email: e[redacted] Gabriela Nicolescu, Ecole Polytechnique de Montréal, Montréal, Canada.

1. INTRODUCTION

By 2010, transistor integration in the order of billions will be achievable; consequently, it will be possible to build an entire multiprocessor system on a single chip [ITRS 2005]. These systems will have to be fast, cheap, reliable, and ready for the market in minimum time. In order to be competitive under these constraints, new EDA tools will be required. These tools will permit system-on-chip (SoC) design at a very high level of abstraction and will support automatic abstraction refinement through several levels.

In the domain of system-on-chip modeling and simulation, many efforts have been invested and several contributions have been proposed. Designers currently have at their disposal efficient standard solutions for hardware modeling and simulation (e.g., VHDL, Verilog), but none are close to perfect.

SystemC [The Open SystemC Initiative (OSCI) 2005], announced in September 1999 is very popular for system-on-chip design. It is based on a library/framework approach implemented with C++. At its core is an event-driven simulation kernel. SystemC provides all the basic concepts used by HDLs (e.g., modules, ports, signals, and time); it also provides additional concepts of higher abstraction, such as interfaces, communication channels, and events. SystemC is a very good solution, but it currently lacks most of the features needed for software modeling, such as dynamic process creation, process control (suspend, resume, kill, etc.), preemption, and software-specific communication primitives, such as monitors.

There exist several other approaches that provide higher level modeling and verification solutions as extensions to existing HDLs. One solution that is representative of this approach is SystemVerilog [Rich 2003; ...Bailey 2003], which is an extension of Verilog. SystemVerilog adds to its predecessor a significant set of features, such as high-level concepts for abstract system modeling and simulation, test-bench automation, and a better C interoperability API.

Despite all the efforts of the community, SoC designers still need new modeling and simulation solutions. This is mainly because of a set of requirements that is mandatory for an efficient modeling and simulation solution, but is still not provided by a single existing environment:

- Easier software component specification and their integration into an overall HW/SW system specification;
- Clean programming features to achieve less error-prone models, easier specification for complex systems and reuse of such specification for further designs [Doulos 2003];
- Introspection features for easier debugging and analysis of complex specifications [Keating and Bricaud 1999; Doucet et al. 2003];
- Possibility of annotating models for different purposes, (e.g., directing synthesis or hooking to verification tools, and creating user friendly HDL syntax) [Newkirk and Vorontsov 2002];
- Translation to a *standard* intermediate format to permit the design of EDA tools independently of description languages [Lee and Neuendorffer 2000],

but also be able to offer accurate information for the different stages of the design (simulation, verification, and refinement, etc.),

- Integration in a distributed web-based design environment to allow remote processing and easy system documentation to facilitate cooperation between different groups of designers [Dalpasso et al. 2002];
- Multiplatform and multilanguage features for describing and designing embedded systems composed of heterogeneous components [Jerraya and Ernst 1999];
- Easier memory management to accelerate the specification process and to eliminate an important source of errors [Rich 2003].

The contribution of our paper is to propose a new, .Net-based methodology enabling fast and efficient creation of EDA tools for complex systems design. This methodology enabled the design of a new tool called ESys.Net (Embedded system design with .Net). This tool: (1) provides most of the concepts of the presented high-level modeling and simulation solutions, (2) respects all the requirements enumerated above, and (3) preserves comparative performances with existing environments.

The reminder of this paper is organized as follows. Section 2 constitutes the background of this contribution: it presents the .Net characteristics arguing our choice of this framework and gives a classification of the model representations used in the current tools. Section 3 presents the core of the methodology that we propose: it presents its originality from the model representations point of view and shows how we can take advantage of .Net Framework's attribute programming and reflectivity capabilities. Section 4 presents ESys.Net, a powerful tool that we designed with our methodology. Section 5 is devoted to the implementation of ESys.Net simulator and illustrates the use of reflectivity. Section 6 discusses the performance and the different features of the environment, such as the multilingual aspects and hooking to verification tools. Section 7 summarizes the current status and presents some perspectives. Section 8 concludes the paper.

2. BACKGROUND

2.1 The .Net Framework—General Presentation

The .Net Framework is a new platform that simplifies application development for the highly distributed Internet environment [Microsoft 2005]. The .Net core represented by the CLI (Common Language Infrastructure) was standardized in December 2001 by ECMA and in April 2003 by ISO [ECMA/ISO 2003]. CLI unifies the design, development, deployment, and execution of distributed components or applications. Its architecture is mainly composed of:

- A Common Intermediate Language (CIL) that supports high-level notions (e.g., classes) and is independent of platforms and programming languages.
- Metadata that enables the addition of information about the context, quality, condition, and characteristics of data. Metadata is used by the .Net Framework to describe programs and their elements. It is possible to add custom

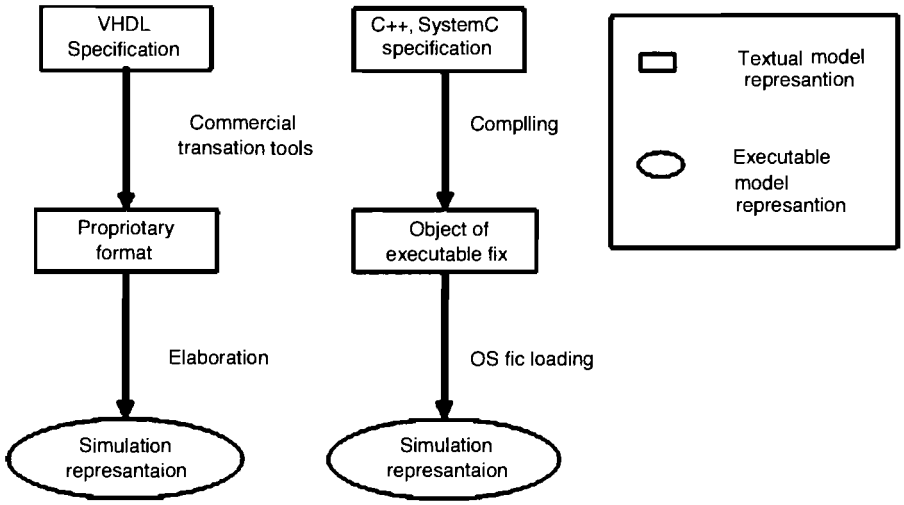


Fig. 1. VHDL/SystemC textual and executable models used in the simulation flow.

metadata to a program by using attributes (attribute programming). A program's metadata may be accessed by using a mechanism called reflection. In the context of embedded system design, metadata may be used to parameterize or describe embedded system components (e.g., abstraction level, refinement, and consign). Advantages of .Net's custom attributes are discussed in Newkirk and Vorontsov [2002].

- A virtual machine that represents the model of an execution environment for applications. The virtual machine simplifies programming by providing a rich run-time infrastructure for applications, such as automatic memory management (garbage collection), remoting, metadata management, and type checking, just to name a few. It should be noted that memory management is planned for the implementation of SystemVerilog, which shows its importance.
- A set of classes providing important functionalities, such as thread management and reflection. It also provides XML [The World Wide Web Consortium (W3C) 2005] data manipulation, text management, collection functionality, and web connectivity, etc.

Alongside the CLI core, .Net Framework presents a set of classes that add supplementary features, such as web services, native and web forms, transaction, scalability, and remote services. The .Net architecture may help system-on-chip tools designers in their efforts to manage complexity of systems, but also with the complexity of design flows. We will give more information in section 3, by explaining how we exploited this methodology.

2.2 Model Representations and Tools

In a traditional SoC development process, many different but complementary model representations are used. We can divide model representations into two categories: textual and executable (Figure 1).

As their name implies, *textual model representations* are system descriptions that are kept in text files. Textual models are not meant to be interacted with during runtime. We analyze textual models by the means of parsing and semantic analysis. Textual models may be *language-oriented* or *metadata-oriented*. An example of a language-oriented textual representation would be a VHDL specification. Language-oriented representations permit designers to describe circuits and hardware/software systems. Since these descriptions are stored using a standard format, it is possible to manipulate them with tools without having any knowledge of previous manipulations. Very often, these descriptions lack information that would be necessary to better inform tools about them, such as flagging a part of a description, indicating that it represents a CPU or indicating what parts of a description should be synthesized; this optional information is called *metadata*. Comments can be used to add metadata; but the format of the information within the comments is not standard resulting in compatibility issues between tools.

Thus, from the language-oriented representation, a metadata-oriented textual representation can be derived. In the case of VHDL, this second representation, also called intermediate format is generally a proprietary nondisclosed format. It will contain all the hierarchy expressed by the designer, such as entities, architectures, packages, reconfigurations, and processes. In the case of SystemC, this level is an object file which may be very poor in metadata, especially if the model has been compiled to optimize execution speed.

At the opposite of textual model representations, *executable model representations* only exist at run-time and can be dynamically interacted with. Executable model representations are usually created from a textual model representation. In the case of VHDL, this representation is obtained from the intermediate format by the process of elaboration where all the details about the configurations, entities, and architectures are removed to obtain a representation of the system as "a sea of processes connected by signals." It is this representation that is used by the VHDL simulator. It is through these model representations that EDA tools manipulate system models.

Figure 1 illustrates the different representations used by VHDL and SystemC. We note that the executable models used by VHDL and SystemC are very well adapted for simulation: the information that they generate is useful for this stage of the design, but it may be incomplete for other stages (e.g., verification or refinement). Since these representation models are adapted to a certain design phase (e.g., simulation) and will be exploited by a specific tool (e.g., a simulator), we call them *tool-specific executable representations*.

Taking into account these definitions, three categories of tools may be defined:

1. Tools using textual representations—this category of tools is generally used for synthesis and static model verification.
2. Tools using executable representation as their input—this category is generally used for custom verification tools binding to an executable model representation that is being managed by a simulator or another verification tool, e.g., assertion-based verification by simulation.

Table I. Model Representations and Tools Manipulating Them

Model Representation	Tools
Textual representation	Static analysis
	Metric tools
	(Non) Formal static representation
	Format translation tools
	Static synthesis tools
Executable model representation	Dynamic analysis and metric tools
	Assertion and observer tools
	Simulation tools
	Dynamic synthesis tools
Executable and textual	Profiling

3. Tools using both mentioned inputs—this category is generally used for profiling certain parts of a model execution that have not changed in the executable model representation compared to the earlier textual format, or if the EDA framework maintains enough information about the mapping from one representation to the next one.

Table I gives some tools examples for each presented category. Current commercial development methodologies offer tools that manipulate textual and executable models. As mentioned earlier, the problem with these solutions is their closeness.

Custom textual model analysis tools are very costly to create because of the complexity of parsing and analysis of complex model description languages. Developers are almost condemned to using commercial software for textual model representation analysis. Also, current commercial tools do not give access to simple clean executable model representations, executable models are usually hidden behind complex APIs, such as Verilog's PLI [.Sutherland 2002], or are cluttered with simulation elements, which is the case in SystemC after macro expansion. This really makes the development of custom tools very difficult.

3. OUR METHODOLOGY FOR EDA TOOLS DESIGN

Our methodology exploits the advance capabilities of the .Net framework in order to achieve considerable advantages for SoC EDA tool design. Using .Net for system model representation has several benefits [Lapalme et al. 2005]. Attribute programming, introspection, and reflectivity are key elements that we have intensively exploited to facilitate EDA tool design and implementation. This will be presented in more detail in sections 3.2 and 3.3.

3.1 Model Representation

The backbone of our methodology is the use of the .Net framework and the C# programming language in order to create different model representations.

A language-oriented representation is considered to be the first textual model representation of our methodology. It can be written with one or more programming languages supported by .Net (Figure 2). Since these languages were not intended for SoC design, a software framework, which implements specific SoC concepts, is required (modules, communication channels, ports, and system

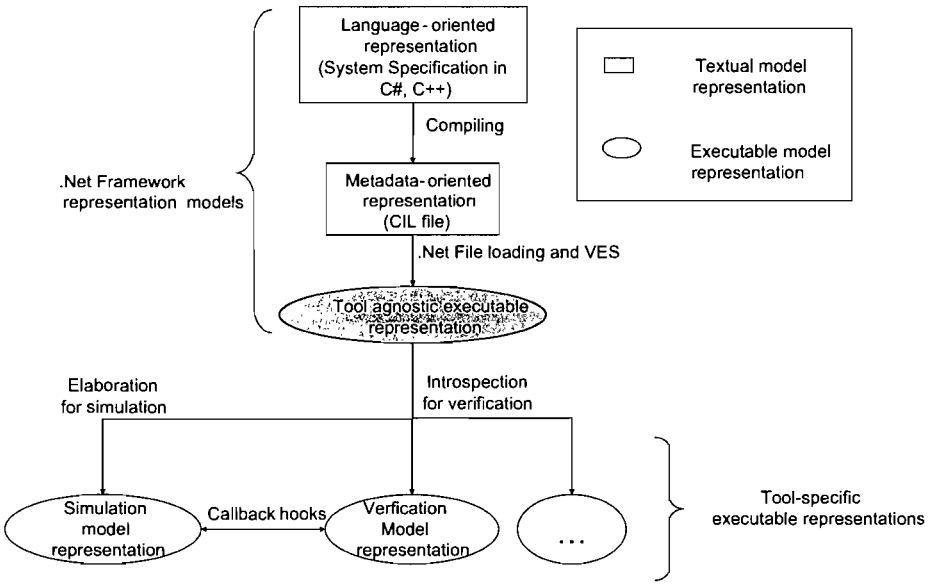


Fig. 2. Proposed methodology.

clocks). We will present such a framework in section 4. At this level of system specification, .Net brings several advantages, the most important being:

- the garbage collector, which alleviates the designers memory management task;
- the thread management functionality, which can be exploited for software component descriptions;
- the metadata, which may be used to annotate models;
- web services, which enable web-based designs.

Our methodology permits the system specification to be compiled into a metadata-oriented textual representation, a CIL file. Through the help of the reflective capabilities of .Net, a CIL textual model representation may be analyzed. Because of the inherited symbolic nature of the CIL representation, all the "plumbing code" of a design tool is hidden so we obtain a "clean" model representation. A CIL textual model representation may easily be transformed by the .Net Virtual Execution System into a rich executable model representation. This executable representation is seen as a rich one, because it contains all the information contained in the CIL textual representation, for example, class hierarchy is still present. It is not specific to a particular simulator and can be manipulated by or interacted with different tools. We call this kind of representation a *tool agnostic executable representation*. Any model created with the methodology will produce an executable model representation (Figure 2).

In summary, using our methodology, a language-oriented representation is transformed into a metadata-oriented one (CIL file), then a tool agnostic

executable is generated, and finally a tool-specific executable view is obtained. Thus, if we compare our methodology with those presented in Figure 1, the main difference is that before we generate a tool specific model representation, we first generate a tool-agnostic executable representation. The latter brings a lot of flexibility and scalability to the development process of design tools: it generally enables the easy addition of new tools into an existing platform. This is mainly because of the fact that it brings information that can be available only at the run-time, such as class polymorphism or variables values at a given time instance. Sections 3.2 and 3.3 will discuss how this information may be retrieved and exploited from the agnostic tool representation.

3.2 Metadata and Attribute Programming

We believe that because of the proliferation of tools other than simulators that use descriptions as their source, a standard mechanism for adding metadata to the description is vital in order to create better EDA tools.

Both the C# and the CLI standards defined a way for adding declarative information (metadata). Since the .Net has at its core the CLI, it also has metadata support. The mechanism through which metadata may be added to a program is called attribute programming. Attributes can be defined and added to basically all the elements of a program [...ECMA/ISO 2003].

The mechanism for retrieving these attributes (metadata) has also been standardized, permitting software components developed by different teams or even companies to interact and discover each other through metadata. Metadata may even be used to control how the program interacts with different run-time entities; it is this capability that we will exploit.

3.3 Introspection and Reflectivity

Generally the capabilities of EDA tools depend heavily on the accuracy of the collected information concerning the system to design and/or the design requirements. A common source of information used by EDA tools is a specification. Tools may extract information by the means of static analysis of a model specification and may even extract more information by inferring data from this static analysis. However, static analysis may be very tedious and confines the model to be predetermined before an EDA tool can be of use—leaving no room for dynamic model construction and analysis—all dynamic elements of a model may not be determined (like signal values, resolution of polymorphism, etc.). Reflection and automated introspection fill the gap left by static analysis and are regarded as necessary for the development and use of EDA tools Nicolescu et al. [2002].

In the system-level modeling context, various kinds of important information may be reflected. The three main information categories are (i) design information (structural and behavioral), (ii) run-time infrastructure information, and (iii) modeling information provided by attribute programming or other means. This information when reflected can allow one to design EDA tools to navigate, manipulate, compose, and connect components, verify the interface compatibilities, and synthesize appropriate interfaces.

```
1. public class MetadataInspecter{
2.    public static void Main(){
3.    Type type = typeof(master);
4.    MethodInfo[] methods = type.GetMethods(Instance|NonPublic));
6.     foreach(MethodInfo method in methods){
7.          Objects[] objs = type.GetCustomAttributes(typeof(EventListAttribute),false);
8.          if(objs. Length == 1){
9.             EventListAttribute eventList = objs[0] as EventListAttribute;
10.            Console.WriteLine("Event: " + eventList.ev + " Port: " + eventList[0]);
11.         }
12.   }
13.  }
14. }
```

Fig. 3. Static analysis of a user model.

These concepts are illustrated in the reflection capabilities of C#, where it is possible to query the CLI to know the structure of an object. To such a query, the CLI returns an object that is an instance of a metaclass named *Type* that fully describes the type. Figure 3 represents the code to introspect the class **master** defining a module of a system. The class has been hard-coded in line 3 for simplification. Lines 4–10 look for all the methods that have an **EventList** attribute and print the associated event and port. This shows how easy it is to introspect a class and, therefore, a model. An example of C# class defining the master module will be explained later, in Section 4, Figure 5b. For that example, the output would be: *Event: posedge Port: clk.*

Note that all the functions that we have used in order to retrieve metadata (see the highlighted named in Figure 3) are already provided by .Net.

As we have seen, attribute programming allows the creation of metadata rich models that can be analyzed easily by EDA tools with the help of introspection and without the need to parse source code. These EDA tools, if constructed while respecting the .Net interoperability rules can be customized by the users to their proper need, other tools can also be created and integrated seamlessly in the design flow.

Using our methodology, we created the ESys.Net (Embedded Systems with .Net) tool based on .Net, Framework and the C# language. It was announced in December 2003 [Goering 2003]. We chose the C# language because it is an ISO standard; it is also the de facto language of the .Net Framework and ensures an efficient use of the framework's capabilities. C# is a strongly typed object-oriented language designed to give the optimum blend of simplicity, expressiveness, and performance. C# and .Net are quite symbiotic [Albahari 2000]. ESys.Net will be presented in the next section.

4. MODELING AND SIMULATING SYSTEMS IN THE ESYS.NET ENVIRONMENT

As we stated in Section 2, the first prerequisite to model and simulate embedded systems in the C# language is the implementation of the basic concepts specific to these systems. Thus, for a better explanation, we start by presenting briefly

the basic concepts specific to SoC and that have to be provided by the ESys.Net environment.

4.1 Concepts for Modeling and Simulation

In terms of modeling, we represent systems as a set of interconnected *modules* communicating through *communication channels* by the intermediary of their *interfaces*. Modules may be hierarchical—composed of several modules—or they may be a leaf in the hierarchy and only consist of an elementary behavior, which may be described with one or several *processes*. Processes may be methods (that cannot be explicitly suspended) or threads (that may be suspended and reactivated) or light threads, called fibers, that cannot be preempted [Shankar 2003].

The same concepts (module, communication channel, and interface) may be represented at different abstraction levels. ESys.Net aims the abstraction levels defined in Nicolescu et al. [2002]. At the highest abstraction level, the *functional level*, a system is described as a set of collaborating functions. The next abstraction level is the *architecture level*, where a system is described as a set of components communicating through abstract channels. Each component represents a component in the final architecture, but the hw/sw partitioning is not yet decided.

After the HW/SW partitioning, we consider different abstraction levels for communication, hardware, and software modules. For the hardware modules, we consider the classical abstraction levels (the transaction level and the Register Transfer level). For the communication, we consider the physical signals and abstract channels (encapsulating complex behaviors). Our work also takes into account abstractions for software models. For the design of software components, such as applications and operating systems (OS), higher abstraction levels than the ISA (Instruction Set Architecture) level are required because of their complexity:

- at the *OS architecture level*, the OS is abstracted and only system calls corresponding to the OS services are visible;
- at the *driver level*, the implementation of the OS's services is fixed but device drivers are still abstracted. Thus, the hardware on which software is executed (e.g., CPU, memory) can be variable. The application code is extended with OS layers implementing OS services (e.g., task scheduling management, and interruption management).

These levels (especially the driver level) are still difficult to model using the existing specification solutions [Yoo et al. 2003]. This underlines the advantage of using the .Net Framework: the thread management functionality (e.g., threads creation and threads control) that is already provided by the framework; it is the main features required to describe software at the driver level.

In addition, our environment presents some concepts related to verification. Thus, it provides the possibility to insert preconditional and postconditional

statements for the different processes of the model (this will be illustrated in Section 4.1.). In terms of simulation semantics, our environment is based on an event-driven simulation kernel.

In the following, we will present the modeling and simulation capabilities of ESys.Net for embedded systems with .Net by the means of the C# language. In order to specify a system, designers must manipulate the presented basic concepts that are provided by the environment core.

4.1.1 *The Core of ESys.Net Environment.* The core of ESys.Net is based on a set of classes, which encapsulate the concepts of modules, communication channels, signals, interfaces, and events.

User-defined modules are obtained by derivation from the abstract **BaseModule** class. ESys.Net is able to detect all instances, whose class derives from the **BaseModule** class, present within the model and registers them automatically. All modules, channels, events, or interfaces instantiated within the hierarchy of a user module are also automatically registered in the simulator's database. This is possible because of the reflection mechanism provided by .Net. In addition, all classes derived from the **BasedModule** class can access information stored in the simulator about the system and execution status (e.g., the current simulation time and the current module name).

User-defined communication channels are derived from the abstract **BaseChannel** class. Since the **BaseChannel** class derives from the **BaseModule** class, all user channels inherit the presented features of a user module (automatic registration of included elements and access to simulator information). The **BaseChannel** class offers the functionality of updating channels at the end of a simulation cycle.

User-defined signals derive from the abstract **BaseSignal** class. This class provides the functionality of storing information that will be readable only at the next simulation cycle. It also provides a transaction event (that indicates that a new data is stored in the signal and its value is equal to the precedent stored value) and a sensitive event (that indicates that a new data is stored and its value is different from the precedent stored value).

Interfaces are directly provided by the .Net Framework. An interface is composed of a set of declarations of methods, but provides no implementation for these methods. Our environment unifies the concept of high-level interfaces and ports. In fact, ports are implemented as predefined interfaces provided by ESys.Net (e.g., inBool, outBool, inoutBool and inInt).

One of the important characteristics of ESys.Net is that it offers the designer the possibility to easily specify execution directives by tagging the different concepts in the specification. These directives concern the association of a thread or parallel method (MethodProcess) semantics to a class method, the addition of a sensitivity list for a method or a thread, the call of methods before or after the execution of a certain process, and the execution of a class method at a specific moment during the execution. This was implemented by exploiting attribute programming provided by .Net and the C# language.

Table II summarizes the available attributes, their semantics, and the concepts to which they are applied.

Table II. Attributes and Their Role in ESys.Net

Attribute	Description		Applied to concept
Process	Associate a thread to a class method		Class method
MethodProcess	Associate a method process to a class method		Class method
FiberProcess	Associate a fiber process to a class method		Class method
EventList (list of events)	Add sensitive list for a process		Process
ManualRegistration	Manually registration of the element		Field
PreCall (Name of Method)	Directives indicating methods execution in explicit points of the execution flow	Method to be called before the process	Process
PostCall (Name of Method)		Method to be called after the process	Process
SimInit (Name of Method)		Simulation init	Class method
SimEnd (Name of Method)		Simulation end	Class method
CycleInit (Name of Method)		Cycle initialization	Class method
CycleEnd (Name of Method)		Cycle end	Class method
DeltaInit		Delta cycle initialization	Class method
DeltaEnd		Delta end	Class method
FinalDelta		Last delta	Class method
Reset		Simulator reset	Class method

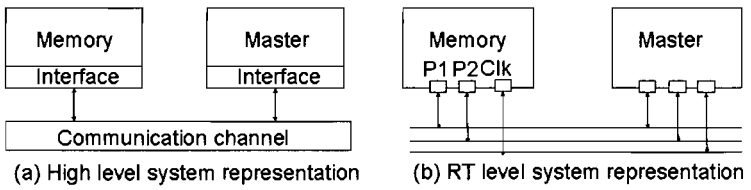


Fig. 4. Simple system example.

In summary, using the implemented concepts described previously, the principal stages of system specification process are:

1. Modules definition;
2. Communication channels definition;
3. Instantiation and interconnection of all the defined modules and communication channels in the overall system specification;
4. Specification of different execution directives.

We will now use an example to better illustrate the use of our system. The example consists of a simple system composed of two communicating modules: a memory module and a master module writing or reading data from the memory. The system is represented at the transaction and RT level (Figure 4). Note that, for clarity, we highlighted the key words specific to our environment.

Figure 5a gives the definition of the *Master* module from the system illustrated in Figure 4(a). This module has an interface to the communication

```
1. public class master : BaseModule
2. {
3.   //module interface
4.   public bus_interface
   interface;
5.   (...)
6.   [Process]
7.   public void main_action()
8.   { while (true) (...)}
9. }
```

(a) Transaction level case

```
1. public class master : BaseModule
2. {
3.   //module interface
4.   public outBool P1;
5.   public InBool P2;
6.   public Clock Clk;
7.   (...)
8.   //process sensitive to the
   positive edge of the clock
9.   [Process]
10.  [EventList("posedge","Clk")]
11.  public void main_action()
12.  { while (true) (...)}
13. }
```

(b) RT level case

Fig. 5. Example of module definition.

```
1.  public class simple_bus :BaseChannel, Simple_bus_direct_if {
2.
3.  // interface of the bus
4.        public SlaveIntf slave_intf;
5.        public Event write_done;
6.
7.        // function that is executed    at the simulation initialization
8.        [SimInit]
9.        public void end_of_elaboration(){...}
10.
11.  // bus interface to the master module
12.       public bool read(ref int data, uint address){...}
13.       public bool write(ref int data, uint address){
14.            (...)
15.            write_done = new Event();
16.            (...)
17.       }
18. }
```

Fig. 6. Communication channel definition example.

channel (line 4). Its behavior is described in a function named *main_action* (see line 7). This function is encapsulated in a process, using an attribute (line 6). In the case where the same module is specified at the RT level, its interface becomes a set of explicit ports (Figure 5b). We consider that alongside the clock connection port (line 6), the module presents an output port and an input port (lines 4 and 5), both of them transferring Boolean data types. The method describing the behavior of the module is encapsulated in a process sensitive to the positive edge of the clock signal. As we explained above, this is specified using attributes (see lines 9 and 10), note that keywords between brackets designate defined attributes.

Figure 6 gives the definition of the high-level communication channel illustrated in Figure 5a. This channel presents an interface for the connection to the memory module (line 4) and an interface for the master module consisting in

```
1.  public class simple_bus_test :  SystemModel {
2.
3.   // channels declarations
4.         public simple_bus bus;
5.   // module declarations
6.         public master master_instance;
7.          public memory memory_instance;
8.
9.   // attaching system to the simulator
10.        Pulic simple_bus_test(ISystemManage manager):base(manager) {
11.
12.   // channel and module instantiation
13.         master_instance = master("master_instance");
14.         memory_instance =  memory("memory_instance");
15.         bus = new simple_bus("bus");
16.
17.   // connect the different  modules to the channels
18.         master_instance_interface = bus;
19.         bus.slave_intf = memory_instance;
20.  }
21. }
```

Fig. 7. Overall system specification example.

the communication primitives for reading and writing data (lines 12 and 13). It encapsulates a function which is executed only once, during the initialization of the simulation (line 9). This is specified using the **SimInit** attribute (line 8). This module also illustrates the use of events for synchronization. The lines 5 and 15 show the declaration and, respectively, the instantiation of an event. Processes may wait for this event (using the syntax *wait (event_name)*) or notify this event (using the syntax *event_name.notify()*). Events can be also part of a sensitivity list specified by an attribute.

Finally, Figure 7 gives the specification of the overall system illustrated in Figure 4a. This specification requires the declarations of the necessary top level modules and channels (lines 4, 6, 7, respectively). The declared modules and channels are then instantiated (lines 13–15) and interconnected (lines 18 and 19). To enable its simulation in ESys.Net, each system has to be attached to the simulator, called, in our case, manager (line 10).

ESys.Net is intended to be an evolution of SystemC by offering a user-friendly environment free of eclectic implementation details like macros, pointers, function, and prototyping. It is also intended to be a superset of SystemC's core functionalities extending it with features like automatic memory management, system level primitives, strong typing, native interfaces, safe pointers, reflective capabilities, remoting, and dynamic thread creating/control, etc. We have also added hooking points within the kernel which are easily used by third-party tools, written in specialized languages [Kilgore 2002], permitting the analysis, synthesis, verification, and viewing of models. It would have been possible to port SystemC to the .Net Framework, but at the price of using nonstandard extensions to C++. It is, however, possible to execute SystemC models in cooperation and in parallel with our models in the same binary file.

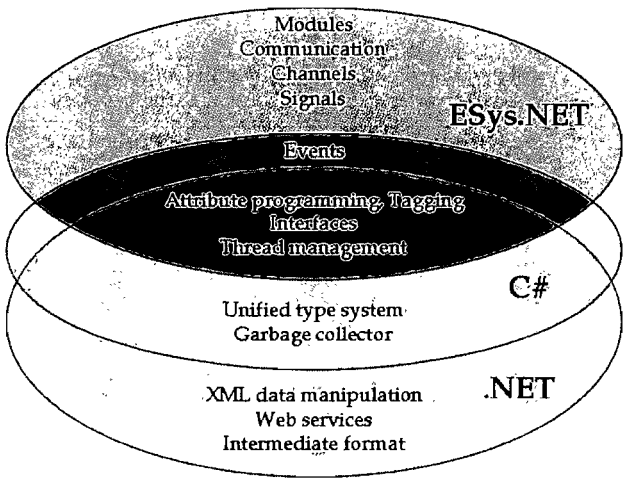


Fig. 8. Layered architecture of ESys.Net.

4.1.2 *Implementation Issues Provided by .Net Framework and C# Language.* The .Net Framework and the C# language provided us with several implementation advantages.

First, to implement the event-based simulation kernel, we exploited the thread management functionality provided by the .Net Framework. Thus, in the ESys.Net environment, a process that describes an element of a module's behavior may be mapped to a .Net thread. ESys.Net processes are very useful for hardware/software modeling and are also very efficient compared to the full-fledged processes. Using the threads provided by .Net eliminated the need of implementing thread management mechanisms. ESys.Net also permits the modeling of processes without the use of threads. These processes are called *MethodProcess*. MethodProcess are similar to **SC_METHOD**s in SystemC. MethodProcess and **SC_METHOD**s do not rely on an underlying thread library or complex synchronization mechanisms. They only have one entry and exit point. There is also no preservation of context from one execution to the next.

The reflection and attribute programming capabilities of the .Net Framework were also useful for the implementation of ESys.Net. The C# language allowed us to exploit efficiently these features by offering the possibility of adding metadata to various programming entities (e.g., objects and methods). As illustrated in the previous section, the various entities in the system (e.g., modules) are automatically registered in the simulator; this is accomplished with the help of reflection. Actually, the implementation of the overall event-based simulation kernel is based on reflection and metadata.

As a summary, Figure 8 illustrates the layered architecture of ESys.Net. As the figure shows, the core presents a reduced number of elements that we have to implement on top of .Net or C#. This highlights the advantage of building the environment on the top of a framework, comparing to building on a language.

The framework gives us several features and parts of them were exploited efficiently because of the use of the C#.

The power of expressiveness provided by the C# language enables us to make a clean and simple implementation of the environment. Thus, all the presented concepts for modeling and simulation were implemented in approximately 1550 lines of code.

5. METHODOLOGY APPLICATION FOR ESYS.NET SIMULATION ENGINE IMPLEMENTATION

This section illustrates the application of the proposed methodology in creating custom tools using attribute programming and introspection. Taking into account future possible interaction between these tools is also discussed. We will use as an example the building of ESys.Net simulator. We will focus on elaboration, initialization, and the event-driven simulator semantics and we will highlight the advantages given by our methodology.

5.1 Elaboration

It is during this phase that ESys.Net model element instances are created and connected together. However, unlike most environments, the elaboration phase is done dynamically at run-time. ESys.Net models do not take for granted a specific simulator. At run-time, a model is bound to a simulator, that, in turn, through .Net's introspection capabilities, analyzes the model (structure and directives) and creates a simulation representation of the model. This permits models to be compiled separately from a specific simulator and to bind the models at a later time to a specific simulator. This also allows having different simulators or tools work on different models or parts of models in a unique binary execution. Verilog, VHDL, and SystemVerilog take for granted that there is only one simulator, so it is implicit and it is during compilation that a model is bound to it. SystemC is bound to an implicit simulator, but part of the elaboration is done at compile time by macro expansion and the rest at runtime.

Figure 9 is a simplified and partial pseudocode of the algorithm we use to discover the elements of a model. As we can see we are interacting with the rich dynamic representation to generate the executable representation pertinent to simulation (Figure 2), by calling the different *Register* methods.

5.2 Initialization

Initialization is the first step in the ESys.Net scheduler. Processes are not executed by default and only processes that have been tagged with a **SimInit** attribute or processes that do not have a sensitivity list, are executed during this phase.

5.3 ESys.Net Scheduler

The semantics of the ESys.Net simulation scheduler are defined by the following steps. A simulation cycle (delta cycle) consists of steps 3 through 11.

```
SubModelEltRegistration(ModelObject element)
  type := GetType(element)
  fields:= GetAllFields(type)
  foreach field in fields
    fieldinstance:=GetFieldInst(field,element)
    select(field instance)
      case Clock:
        RegisterClock(field instance)
      case Channel:
        RegisterChannel(fieldinstance)
        SubModelEltRegister (fieldinstance)
      case Module:
        RegisterModule(field instance)
        SubModelEltRegister (field instance)
      case Signal:
        RegisterSignal(field instance)
      case Event:
        RegisterEvent(field instance)
```

Fig. 9. Registration of aspects related to simulation.

1) Initialization Phase.
2) **Execute cycle initialization callbacks**
3) **Execute delta initialization callbacks**
4) Evaluate Phase. Select a process from the set of those that are ready to run. The order in which processes are selected for execution from the set of processes that are ready to run is unspecified.
5) **Execute pre-method callbacks for the current process**
6) Resume current process execution
7) **Execute post-method callbacks for the current process**
The execution of a process may cause immediate event notifications to occur, possibly resulting in additional processes becoming ready to run in the same evaluate phase.
8) Repeat Step 4 for any other ready process
9) **Execute pre-update callbacks**
10) Update Phase. Update delta-cycle dependant elements that requested updates (signals and primitive channels)
11) If there are pending delta-delay notifications, determine which process is ready to run and go to step 3.
12) **Execute last delta callbacks**
13) If there are no more timed event notifications, go to step 18.
14) **Else, execute cycle end callbacks**
15) Advance the current simulation time to the time of the earliest (next) pending timed event notification.
16) Determine which processes become ready to run due to the events that have pending notifications at the current time. Go to step 2.
17) **Execute simulation-end callbacks**
18) Simulation end

As illustrated by the steps of a simulator scheduler, we have added many hooking points (steps in bold) within the simulation kernel. Since these hooking points are implemented with delegates and events, it is possible to hook many callbacks to a same point and callbacks may be class instance methods or static class methods, which is more general than SystemVerilog's callbacks, which

```
1.  public class SimpleBusApp{
2.    static public void Main(){
3.    My VeriTool tool = new VeriTool();
4.    MyModel  model = new MyModel(sim);
5.    Simulator  sim = new Simulator(sbt);
6.    sim.cycleInit += new HookPoint(tool.verify);
7.    sim.Run(100000);}}
```

Fig. 10. Hooking two collaborative tools by callbacks.

are only global C functions. Since we are also using delegates, it is possible to bind a method to a hook point at run-time because it is not necessary to know at compile time the names of the methods we want to bind to a delegate.

Figure 10 is a simple code that illustrates the instantiation of a model, a simulator and a verification tool [Gorse et al. 2004], and their binding. Notice that with a simple line of code, we can bind a method to the **CycleInit** event of a simulator (line 6). The **CycleInit** event is triggered at the beginning of every simulation cycle.

6. ANALYSIS AND EVALUATION

We believe that for system-level modeling and tool building/customization, this environment possesses many advantages over the current ones, because of the following characteristics.

6.1 Safe Modeling and Tool Building

.Net provides a large number of safety features that can be applied to modeling as well as tool creation or customization, such as type checking and automatic memory management.

6.2 Separation of Concerns between Models and Simulation

The intermediate format of .Net (CIL) is used as a neutral and public format between tools. The intermediate format is clean and complete—complete in the sense that all of the metadata present within the model description and code structure and organization are kept. We say that the format is clean because very little nonmodel-dependent information is present. In order to illustrate the two points, we will take SystemC as an example.

Being based on a C++ library, SystemC has inherited all the wonders of the C++ language: speed, power, and flexibility, but has also inherited all its evils: error-prone, complex, and, most importantly, lacking run-time features like: memory management, type verification, and introspective capabilities. When C++ IP blocks are compiled, a lot of information is lost: structures are flattened, abstract data structures are minimized, and it is not possible to obtain precise information on elements found in the model. Thus, even though C++ IP block are fast to simulate, this reduced observability makes them hard to manipulate.

In regard "cleaness" in SystemC's, its source code has a fairly clean layout that can be used for static analysis, but when it is compiled all the macros are

expanded adding a lot of code that has nothing to do with the model, but is necessary for SystemC's elaboration phase and for binding the model to the simulator kernel. Tools that must analyze SystemC IP blocks must deal with a polluted model description.

Thus, we can say that our methodology offers a fairly clear separation of concerns between description models and simulation kernel. We believe that the strong separation is better than the environment for component-based approaches.

6.3 Dynamic Models

An important downfall of most current system level design environments is that they only manipulate static models, meaning that a model cannot be assembled or modified without changing the models description and recompiling it. This enormously hinders the automatic exploration of different architectures and model organizations that could help with partitioning issues. Since current environments have an implicit and unique simulator, it is also not possible to simulate either multiple models with different simulators or split a model between different simulators (software/hardware cosimulation) within the same binary simulation.

Consequently, since dynamic model composition with IPs cannot be done in existing environments, the creation of Rapid Application Development (RAD) is very difficult. As explained in Section 3, this drawback has been alleviated by this methodology.

6.4 Easier Model Analysis

In the case of ESys.Net, since it is based on the .Net framework, it inherits a clean intermediate format and an agnostic tool-executable representation, which leads to powerful reflective run-time capabilities. These two important points permit the easy creation of custom EDA tools that can use high-level languages to explore and analyze models. By using logic languages coupled with introspection, it would be possible to rapidly develop verification engines for ESys.Net. By using rich pattern-matching language, it could be possible to quickly analyze the intermediate format for synthesis. Visual Basic and Java open the door for the creation of elegant visualization tools. This is the first time we have a rich intermediate format that can be manipulated by multiple languages within the same binary execution.

This multilanguage interoperability allowed us to use fibers (lightweight processes managed by the simulator application) instead of full-fledged threads to implement processes equivalent to those of SystemC. This resulted in increased performance compared to results published in Lapalme et al. [2004]. Since fibers are accessible through C++, but not yet in C#, the use of these processes is done using C++. Figure 11 shows that the fibers may be one order of magnitude more efficient than the threads provided in .Net C#. The base of comparison is sc_methods and sc_threads of SystemC. The heavier threads of .Net can still be used when preemption, priority, or other system features are needed.

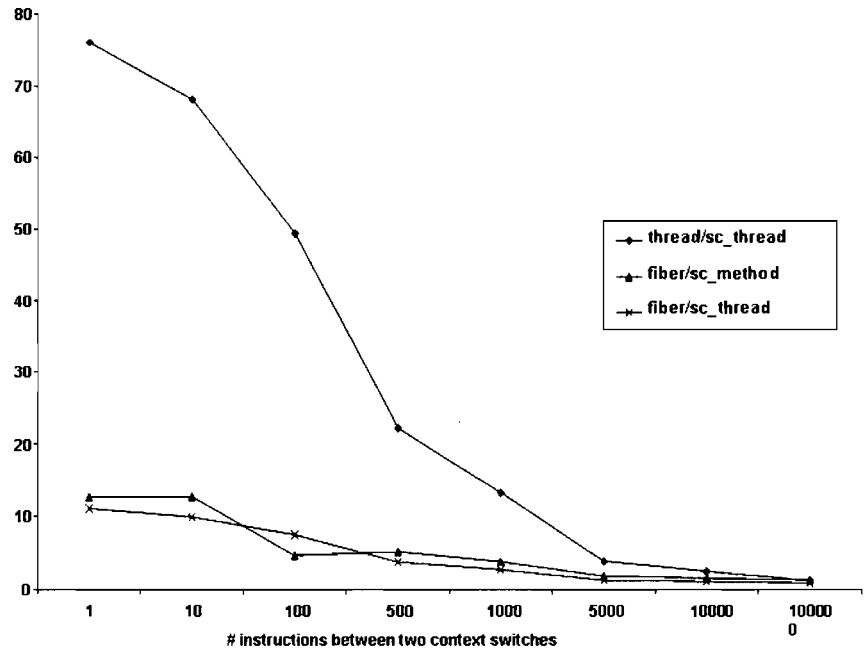


Fig. 11. Performances of fibers and threads vs. sc_threads.

6.5 Easier EDA Tools Hooking

Our methodology fulfills the need for third-party tool binding with its multiple hooking points that are easily accessible. The need for complex API's such as PLI [.Sutherland 2002] or the need for kernel modification that can lead to tool incompatibility are removed.

7. CURRENT STATUS AND FUTURE WORK

Our methodology conferred to ESys.Net several interesting characteristics: the connection to a metadata-oriented standard intermediate format, the multilanguage features of the .Net Framework, and the expressiveness, the reflectivity, and the attribute programming model provided by the C# language. These advantages allow us to consider very interesting perspectives for the future development of ESys.Net.

We are currently working on the implementation of the software abstractions presented in Section 4. We successfully modeled in our environment an existing OS at the various abstraction levels.

Our future work will have as prime objective the exploration of refinement under its many facets. First, the standard intermediate format will permit the creation of new refinement tools that can be guided in better ways by the use of metadata, which annotates and extends manipulated models. Moreover,

in order to bridge the gap between our environment and other existing tools (and environments) for hardware synthesis and model analysis (ex. CoCentric [Synopsys 2001]), we will focus on the automatic translation of C# specification to SystemC and/or VHDL models. We have recently started working on a new approach that will extend the ESys.Net environment with HW/SW partitioning features. The implementation is facilitated by the use of attributes (for partitioning directives) and the reflective capabilities of .Net (for in-line checking of the respect of constraints and even system hw/sw

Concerning the issues of simulation, we intend to prove the multilanguage and distributed capabilities of .Net by exploiting them in the study of software application and operating systems executed at different abstraction levels (see Section 3.1).

In the mid-term, our team will address the problems surrounding the verification of specifications and models [Gorse et al. 2004]. When we designed the simulation core of ESys.Net, we had in mind the future needs required by verification and analysis tools; for instance, many hooking points within the simulation kernel have been provided enabling the integration of external tools. Several APIs also enable the introspection of the system's status at any given moment during a simulation.

8. CONCLUSION

Today, in order to respect the time to market and strict cost constraints, embedded system designers need new modeling and simulation solutions. These solutions must enable easier memory management. They must also permit software component modeling, component integration in a distributed web-based environment, easier debugging of complex specifications, multilanguage features, and mitigated connection with other existing or new CAD tools.

In this paper, we presented a methodology for tool design. The main advantage of this methodology is the ability to obtain a tool-agnostic executable representation of the system under design. This representation is the common reference for all the designers of new tools. This increases dramatically the possibility to create a complete, consistent, and unique solution that provides tools for different key stages in SoC design (simulation, verification, partitioning, etc.). Since the methodology is based on .Net, this common reference representation is already standardized and all the APIs needed for its manipulation are currently provided.

This methodology enabled us to design a new environment for SoC modeling and simulating. This environment fulfills the set of requirements that we enumerated in the paper and with no important performance cost. The key point of our approach is the use of the advanced features present in the .Net Framework and the C# language.

This work also confirms that EDA tools efficiency is influenced by the methodology used for their design and that software expertise might bring substantial contribution for the design of SoC; a design flow that will require more and more the "coming together" of several domains of expertise.

REFERENCES

ALBAHARI, B. 2000. *A Comparative Overview of C#.* Available from: http://genamics.com/developer/csharp_comparative.htm.

BAILEY, S. 2003. "Comparison of VHDL, Verilog and SystemVerilog," Model Technology, Digital Simulation White Paper.

DALPASSO, M., BENINI, L. AND BOGLIOLO, A. 2002. Virtual simulation of distributed IP-based designs. *Design and Test of Computers, IEEE,* 19, 92–104.

DOUCET, F., SHUKLA, S. AND GUPTA, R., 10382. 2003. Introspection in system-level language frameworks: meta-level vs. integrated. *In Proceedings of the Conference on Design, Automation and Test in Europe,* March 03 - 07, IEEE Computer Society, Washington, DC.

DOULOS 2003. *SystemC in Europe—current usage and future requirements.* Available from: http://www.doulos.com/systemc_report/

ECMA/ISO 2003. *ECMA and ISO/IEC C# and Common Language Infrastructure Standards.* Available from: http://msdn.microsoft.com/netframework/ecma/.

GOERING, R. 2003. *Researchers tap .Net, C# for system-level design.* Available from: http://www.eetimes.com/.

GORSE, N., METZGER, M., LAPALME, J., ABOULHAMID, E. M., SAVARIA, Y. AND NICOLESCU, G. 2004. Enhancing ESys.Net with a semi-formal verification layer. *In Proceedings of the 16th IEEE Intl Conference on Microelectronics (ICM'04),* Tunis, Tunisia, Dec. 2004. 388–391.

ITRS 2005. *International Technology Roadmap for Semiconductors, Edition 2003.* Available from: http://public.itrs.net/.

JERRAYA, A. AND ERNST, R. 1999. Multi-language system design. *In Proceedings of the Conference on Design, Automation and Test in Europe DATE '99,* Munich, Germany, ACM Press, New York, NY, 134.

KEATING, M. AND BRICAUD, P. 1999. *Reuse Methodology Manual,* Kluwer Academic Publi., Boston, MA.

KILGORE, R. A. 2002. Open source initiatives for simulation software: multi-language, open-source modeling using the microsoft .NET architecture. *In Proceedings of the 34th Conference on Winter Simulation: Exploring New Frontiers,* San Diego, CA, December 08–11, 2002. 629–633.

LAPALME, J., ABOULHAMID, E. M., NICOLESCU, G., CHAREST, L., DAVID, J.-P., BOYER, F. AND BOIS, G. 2004. ESys.NET: A new solution for embedded systems modeling and simulation. *In Proceedings of the ACM SIGPLAN/SIGBED 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04),* Washington, DC, June 11–13, 2004. 107–114.

LAPALME, J., ABOULHAMID, E. M. AND NICOLESCU, G. 2005. Leveraging model representations for system level design tools. *In Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping (RSP'05),* Montreal, Canada, June 2005. 33–39.

LEE, E. A. AND NEUENDORFFER, S. 2000. "MoML—A Modeling Markup Language in XML, Version 0.4," U. of California, Berkeley, Technical Memorandum UCB/ERL M00/12.

MICROSOFT 2005. *Microsoft .NET Framework.* Available from: http://msdn.microsoft.com/netframework/.

NEWKIRK, J. AND VORONTSOV, A. A. 2002. How .NET's custom attributes affect design. *IEEE Software,* 19, 18–20.

NICOLESCU, G., YOO, S., BOUCHHIMA, A. AND JERRAYA, A. A. 2002. Validation in a component-based design flow for multicore SoCs. *In Proceedings of the 15th international Symposium on System Synthesis ISSS '02,* Kyoto, Japan, October 02–04, ACM Press, New York, NY. 162–167.

RICH, D. I. 2003. The Evolution of SystemVerilog. *IEEE Design and Test of Computers,* 20. 82–84.

SHANKAR, A. 2003. *Implementing Coroutines for .Net by Wrapping the Unmanaged Fiber API.* Available from: http://msdn.microsoft.com/msdnmag/issues/03/09/CoroutinesinNET/default.aspx.

SUTHERLAND, S. 2002. *The Verilog PLI Handbook,* Kluwer, Acad. Publ., Boston, MA.

SYNOPSYS 2001. *CoCentric® System Studio.*

THE OPEN SYSTEMC INITIATIVE (OSCI). 2005. *SystemC 2.1 Language Reference Manual.* Available from: http://www.systemc.org/.

THE WORLD WIDE WEB CONSORTIUM (W3C). 2005. *Extensible Markup Language (XML).* Available from: http://www.w3.org/XML.

YOO, S., BACIVAROV, I., BOUCHHIMA, A., PAVIOT, Y. AND JERRAYA, A. A. 2003. Building fast and accurate SW simulation models based on hardware abstraction layer and simulation environment abstraction layer. *In Proceedings of the Conference on Design, Automation and Test in Europe*, March 03–07, 2003, IEEE Computer Society, Washington, DC.

Received February 2005; accepted June 2005

La séparation des aspects de modélisation et de simulation dans les langages orientés-« framework » de modélisation matériel/logiciel

À l'heure actuelle, les concepteurs de systèmes embarqués ont plusieurs solutions relativement efficaces pour la modélisation et la simulation (e.g. SystemC, SystemVerilog, etc.). Les langages utilisés par ces solutions pour la spécification et la modélisation de systèmes peuvent être catégorisés comme des langages dédiés. Ces langages sont dédiés à des domaines applicatifs précis, donc ils se distinguent des langages généralistes, comme Java et C ++, qui n'ont pas de biais pour un domaine en particulier. Depuis quelques années, dans le monde du logiciel, les langages dédiés sont devenus un sujet de recherche important quant à leur conception et leur utilisation. Plusieurs avantages sont souvent attribués à l'utilisation de ces langages [60] :

- ils permettent d'exprimer des solutions avec les tournures diomatiques au niveau d'abstraction du domaine traité. En conséquence, les experts du domaine eux-mêmes peuvent comprendre, valider, modifier, et souvent même développer des programmes en langage dédié.
- ils facilitent la documentation du code.
- ils améliorent la qualité, la productivité, la fiabilité, la maintenabilité, la portabilité et les possibilités de réutilisation.
- ils permettent la validation au niveau du domaine. Aussi longtemps que les éléments du langage sont sûrs, toute phrase écrite avec ces éléments peut être considérée comme sûre.

Il existe deux catégories de langages dédiés, soit les indépendants et les enfouis. Les langages dédiés indépendants sont caractérisés par le fait qu'ils ont des syntaxes, des compilateurs et des analyseurs qui leur sont propres. Les langages dédiés enfouis sont caractérisés par le fait qu'il utilise un langage hôte, typiquement un langage

généraliste, pour leurs syntaxes, les compilateurs, etc. Ce type de langage est généralement défini via une bibliothèque de code qui implémente la sémantique nécessaire pour traiter du domaine visé par celui-ci. Puisque les langages généralistes orienté objet sont bien adaptés pour la conception de bibliothèques bien conçues, ils sont souvent utilisées comme langage hôte. On peut faire le parallèle entre les deux familles de SDL présentées dans la revue et les deux catégories de langage dédié. Les SDL indépendants sont des langages dédiés indépendants et les SDL orienté bibliothèque sont des langages dédiés enfouis.

Beaucoup d'effort ont été investis dans la définition de langages enfouis pour le domaine des systèmes tel que SystemC. Malgré ces efforts, très peu de travail a été fait sur l'investigation des techniques de conception et technologies logiciels qui seraient les plus appropriées pour la définition de ces langages dédiés. Les bibliothèques de codes, donc par conséquence les langages dédiés enfouis, sont très difficiles à concevoir. Leur conception influence grandement :

- leur efficacité;
- leur facilité d'utilisation;
- leur capacité à promouvoir des solutions biens conçues;
- leur capacité à être étendu.

Depuis plusieurs décennies, la communauté du logiciel a vu émerger plusieurs technologies de grande valeur telles que la conception par patron [31], le principe de conception par séparation d'aspects, et les plateformes de développement évoluées comme .Net. Par l'assemblage de ces technologies, il est possible de concevoir une nouvelle génération d'environnements et de langages pour la conception de systèmes embarqués. Cette nouvelle génération d'outils offre une séparation non ambiguë entre les diverses facettes de la conception des systèmes telles que la modélisation, la simulation, la vérification, etc. Cette séparation offre plusieurs bénéfices.

Les contributions principales de cet article sont:

- l'introduction du concept des langages dédiés à la communauté des systèmes embarqués;
- l'application du principe de la séparation des aspects aux langages dédiés enfouis pour les systèmes embarqués;

- la présentation de diverses technologies modernes du génie logiciel qui peuvent être appliquées à la conception des langages dédiés enfouis pour les systèmes embarqués;
- la définition d'une architecture cible d'outils CAO pour les systèmes embarqués. Cette architecture met en pratique la séparation des aspects et les technologies modernes du génie logiciel. Cette architecture confère aux outils plusieurs bénéfices par rapport aux autres solutions disponibles;
- la construction de deux outils, un pour la modélisation et l'autre pour la simulation de systèmes embarqués utilisant cette nouvelle technologie. Le langage de modélisation porte le nom de « SoCML ». Ces outils offrent la majorité des fonctionnalités de SystemC. De plus, ils comportent plusieurs avantages importants sur SystemC;

Les pages suivantes contiennent une copie de l'article [47], dans son format original (sauf la numérotation des pages), publié dans *The Arabian Journal for Science and Engineering, Vol 32, Num 2C, 2007.*

SEPARATING MODELING AND SIMULATION ASPECTS IN HARDWARE/SOFTWARE FRAMEWORK-BASED MODELING LANGUAGES

James Lapalme†, El Mostapha Aboulhamid

Laboratoire LASSO, IR,OUniversité de Montréal
C.P. 6128 , Succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7

Gabriela Nicolescu

École Polytechnique de Montreal, Canada
C.P. 6079, succ. Centre-Ville, Montréal Québec, Canada H3C 3A7

and Frédéric Rousseau

TIMA, 46 av. Felix Viallet, 38031 Grenoble CEDEX, France

الخلاصـة:

من المعروف أنه كلما زاد عدد الترانزستورات في الشريحة الواحدة زادت الفجوة الإنتاجية مع صناعة السليكون اتساعاً. لذلك يسعى العديد من الباحثين لحل المشكلة من زوايا مختلفة. فبينما يرى البعض أن الحل يكمن في لغات النمذجة المختصة بحقل معين يرى آخرون أن الحل يكمن في استخدام لغات النمذجة التي تعتمد على طريقة المكتبة البنية. ويرى فريق آخر أن استخدام الأدوات المعقدة والمسجلة هو الحل. والحقيقة أن أياً من هذه الحلول ليس حلاً كاملاً. ومع هذا فإن الحلول القائمة على الهياكل المبنية علي التوجه إلى الهدف مثل نظـام (C) تكتسب زخماً كبيراً وقبولاً من الصناعة. وبالرغم من كل الجهود المبذولة من أجل تطوير هذه الحلول فإن جهوداً قليلة هي التي بُذلت من أجل دراسة تقنيات تصميم البرمجيات اللازمة لتطوير حلول قائمة على الهياكل المبنية. ولذلك فإن الغرض الرئيسي من هذه الورقة هو عرض كيفية استخدام تقنيات هندسة البرمجيات للحصول على حلول أفضل للنمذجة القائمة علي الهياكل. وهذه الحلول تتميز بالفصل الواضح بين الاعتبارات الخاصة بالنمذجة وتلك الخاصة بالمحاكاة. وتقدم الورقة طريقة جديدة للنمذجة الهيكلية تُسمّى (SoCML) و تتمتع بالخاصية السالفة الذكر. وتتمتع (SoCML) أيضاً بمزايا أخرى عديدة مثل التحقق باستخدام الاعتراض، وكذلك الدعم البديل للمحاكاة.

† To whom correspondence should be addressed.
E-mail: [redacted]

Paper Received: 16 April 2007; Revised: 15 November 2007; Accepted: 28 November 2007

James Lapalme, El Mostapha Aboulhamid, Gabriela Nicolescu, and Frédéric Rousseau

ABSTRACT

As transistor integration reaches the order of billions, the already significant productivity gap which plagues the silicon industry will only widen further. Many are working on the problem from different angles. Some regard domain-specific modeling languages as a solution. Others believe in modeling languages which are based on a library/framework approach. Yet others believe in sophisticated proprietary tools. None of the current paths seem to be silver bullets. However, object-oriented framework-based solutions, such as SystemC, are gaining a great deal of momentum and acceptance from the industry. Despite all the efforts which have been spent on the development of these types of solutions, very few efforts have been spent on the cornerstone task of investigating which software design techniques and technologies should be used to develop effective framework-based solutions. The main objective of this article is to present how modern software engineering technologies may be used to create better framework-based modeling solutions. These solutions are characterized by a clear separation of concerns between modeling and simulation aspects. A novel modeling framework called SoCML is presented which possesses the above characteristic. SoCML has many benefits such as verification by interception and alternative simulation support.

Key words: logic design hardware description languages, simulation, verification, VHDL, programming languages: design languages, C#, C++, concurrent, simulation and modeling: simulation languages, modeling methodologies, environments

James Lapalme, El Mostapha Aboulhamid, Gabriela Nicolescu, and Frédéric Rousseau

SEPARATING MODELING AND SIMULATION ASPECTS IN HARDWARE/SOFTWARE FRAMEWORK-BASED MODELING LANGUAGES

1. INTRODUCTION

As transistor integration reaches the order of billions [1], the already significant productivity gap which plagues the Electronic Design Automation (EDA) industry will only widen further. Many are working on the problem from different angles:

- Some are working on design flows based on dedicated modeling languages in order to aid designers model complex systems effectively and at higher levels of abstractions then was previously possible [2].
- Others have taken the path of library/framework-based solutions which rely on existing mainstream programming languages. These solutions capitalize on existing tools and technologies and allow the integration of new ones in order to achieve novel design flows [3].
- Others are looking towards sophisticated tools—electronic design automation (EDA)—based on proprietary technology in order to offer "out of the box" design-flow solutions [4].

None of the mainstream approaches seem to be silver bullets; however, object-oriented framework-based solutions such as SystemC [3] are gaining a great deal of momentum and acceptance by the industry. Given this fact, we started, in 2003, working on a new .Net based methodology which enabled the fast and efficient creation of EDA tools for complex systems design. This methodology made the design of a new tool called ESys.Net [5] (Embedded System Design with .Net) possible. Esys.Net: (1) provides most of the concepts of high-level modeling and simulation solutions, (2) inherits features from .Net which allow less error prone modeling, (3) facilitates tool interoperability, (4) permits custom annotation definition, (5) enables model enrichment by annotation (*e.g.* directing synthesis or hooking to verification tools, creating user friendly HDL syntax, *etc.*); and (6) preserves comparative performances with existing modeling and simulation solutions [6,7].

Despite all the efforts which have been made to develop framework-based solutions, as well as the numerous satellite tools, very few efforts have been made on the cornerstone task of investigating which software design techniques and technologies should be used to develop effective solutions.

Software frameworks are quite difficult to build; their design has tremendous impact on:

- Their effectiveness,
- Their ease of use,
- Their ability to promote good designs,
- Their capability to be extended easily.

The software community, over the past decade, has invested a great deal of effort in the domain of software design; pattern-oriented designs are the fruits of these efforts [8]. Moreover, the conflicting needs of the software industry for "rapid time-to-market" (quick design and implementation) solutions which have a low "cost of ownership" (flexible and may easily evolve) has caused the emergence of novel software engineering technologies. The "container-based" implementation approach and 3GL programming languages which support rapid development exemplify these new technologies. Software design principles such as "separation of concerns" have also been maturing, becoming more present in technologies such as software containers, aspect-oriented programming, and strategic programming.

By combining the advanced capabilities of a modern object-oriented programming language such as C#/.Net and the flexibility and elegance of modern software design patterns such as *Inversion of Control* and *Proxy*, it is possible to create a novel framework-based solution for hardware/software system modeling and simulation. We will present a solution which permits a clear and unambiguous separation between the modeling, the verification and the simulation aspects, hence achieving perfect separation of concerns. The level of separation of concerns offered by the solution permits the elaboration and refinement of various simulation engines such as software, distributed and emulation without any modification to those system models which were previously created.

The main objective of this article is to present how new software engineering technologies may be used to create better framework-based modeling solutions. We will (1) present interesting software engineering technologies; (2) show

how current solutions lack "separation of concerns" in their design and discuss the impact; (3) present a novel modeling framework based on the technologies presented earlier; (4) present a simulation framework for the modeling framework; and (5) discuss the benefits of the solution with regards to simulation, synthesis, and verification of the modeled hardware/software systems.

2. BACKGROUND

2.1. Separation of Concerns (SoC)

Probably coined by Edger W. Dijkstra in his paper on the role of scientific thought [9], the concept of separation of concerns is an important principle which lies at the heart of modern software engineering. The basis of SoC is the decomposition of a problem into sub-problems which are orthogonal to each other. SoC takes a classical divide-and-conquer approach to problem solving but relies on aspect decomposition instead of traditional functional decomposition.

Within the context of system modeling and simulation, it can be said that the problem of modeling a system is orthogonal to the problem of simulating a system. Even though both problems are related to one another by common modeling semantics, one is concerned about the "what to simulate" and the other about "the how to simulate". For example, if we have a system such as a cruise control unit which must be modeled and simulated, it should be possible to model the system with certain modeling semantics, such as a hierarchical sea of process, without taking in consideration whether the system will be simulated or emulated.

2.2. C#/.Net 2.0 and Generics

At the end of 2005, Microsoft released the next official versions of .Net and the C# programming language, both versioned 2.0 [10], [11]. Of the many enhancements made to .Net and C#, the implementation of generics types is especially important.

Generic programming, popularized by C++ (templates), is a programming paradigm used by statically type languages in which a piece of software is specified in a way abstracting type information. When a piece of generic software must be used, a programmer must specify a type binding which specializes the software for a given type. Most often, the compiler will duplicate the original generic code but with the type information added in order to enforce static typing. Both Java 1.5 and C++ used this kind of compile time resolution in order to implement the generic programming paradigm.

The designers of .Net took a different approach than the above when implementing generics. The .Net technology is built from the ground up on metadata. In .Net, when a piece of software is compiled, it is transformed into a language agnostic intermediate format called CIL. The CIL instruction-set is based on an abstract stack machine. The intermediate format contains a lot a metadata about the structure of the software that was compiled. The concept of generics was implemented as an extension of the metadata and instruction-set. Because of its implementation strategy, .Net generics are resolved at run-time and not compile time. This makes a big difference at runtime. Through the use of reflection, it is possible to determine if an object is an instance of a generic type as well as the bound types of a generic instance. It is also possible to dynamically bind a generic type and create instance of that binding.

Here is a simple example of dynamic binding and instantiation:

```
Type aType = anObject.GetType();
if (aType.IsGenericType) {
 if (aType.GetGenericTypeDefinition() == typeof(signal<>)) {
   Type signalType = aType.GetGenericArguments()[0];
   Type newType = typeof(GenericSignal<>).MakeGenericType(signalType);
   Object newObj = Activator.CreateInstance(newType));
  }
 }
```

This implementation of generics allows the runtime analysis of generic bindings, the creation of new bindings and the instantiation of those bindings which are very powerful features that we shall explore later in the article. These capabilities, to our knowledge, are unique for a statically type programming environment.

Moreover, the implementation of generics proposed by .Net permits the definition of constraints in order to restrict the types that may be bound to a generic definition.

```
public class Dictionary<K, V> where K : IComparable {}
```

In the above example, a generic dictionary class must be bound to a type for its keys (K) and a type for its values (V). The generic binding is constrained by the fact that the type which is used for the keys must support the IComparable interface.

3. MODELING AND SIMULATION FRAMEWORK

Many efforts have been invested and several contributions have been proposed for system-on-chip modeling and simulation. Current designers have at their disposal efficient solutions for hardware modeling and simulation (*e.g.* VHDL, Verilog); however, few would argue that these solutions are close to being perfect. Most currently available modeling/simulation solutions fall into one of two categories: those which are implemented using a framework based approach and those which are dedicated languages. SystemC [3], ESys.Net [5], JHDL [13], and Ptolemy 2[12] are representative solutions of the first category which may be referred to as domain-specific embedded languages (DSEL) [14]. SpecC [15], VHDL [16], and SystemVerilog [17] are representative solutions of the second category which may be referred to as domain-specific languages (DSL) [14]. In this paper we are concerned with the first category of solutions. The design of a system-oriented DSEL offers an interesting challenge. Since DSELs are implemented using general-purpose programming languages, DSEL designers are constrained by two elements of the host language when defining the necessary system modeling and simulation concepts: (1) syntactical limitations (lexical and grammatical) and (2) a finite set of semantic building blocks.

3.1. Current Solutions

SystemC [3], announced in September 1999, is very popular for system-on-chip design. It is based on a library/framework approach implemented with C++. At its core is an event-driven simulation kernel. SystemC provides all the basic concepts found in HDLs (*e.g.* modules, ports, signals, time, *etc.*). It also provides additional concepts of higher abstraction such as interfaces, communication channels and events.

ESys.Net is a modeling and simulation framework which is based on SystemC. A research team from the Université de Montréal ported the core concepts of SystemC to .Net in order to capitalize on many interesting capabilities of the platform such as threading, reflection and attribute programming.

Ptolemy II is a software framework developed as part of the Ptolemy Project. It is a Java-based component assembly framework with a graphical user interface called Vergil. Vergil itself is a component assembly defined in Ptolemy II. The Ptolemy project studies modeling, simulation, and design of concurrent, real-time, embedded systems. Its focus is on the assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interactions between components. A major problem area being addressed by the project is the use of heterogeneous mixtures of models of computation.

3.2. Lack of Separation in Current Framework Based Solutions

Traditional HDLs such as Verilog and VHDL were developed from the start with modeling in mind; the simulation of models describe with those languages was a secondary objective. This had a great influence on those standards for there is very little simulation semantics in them. This separation of concerns between modeling and simulation semantics is at the very opposite of environments such as SystemC, ESys.Net, and Ptolemy. We intentionally omit the terminology of "language" to describe these solutions for they are truly simulation solutions and not modeling languages. Our reluctance to qualify the later solutions as modeling languages lies in the fact that there exists no clear boundary between the aspects for simulations and modeling; one cannot model with these solutions and easily change the simulation implementation, especially after the model has been compiled with the simulation framework. In a perfect object-oriented framework, a model should "at all times" be dependent only on the modeling aspects of the framework and not the simulation aspects. The "glue" element between the model and the simulation framework would be the modeling framework which would serve as a contractual interface. ***Figure 1*** represents the dependency architecture of current simulation/modeling frameworks. ***Figure 2*** represents and idealized dependency architecture.

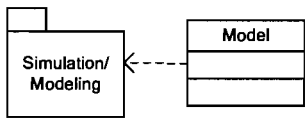


Figure 1. Current frameworks dependencies

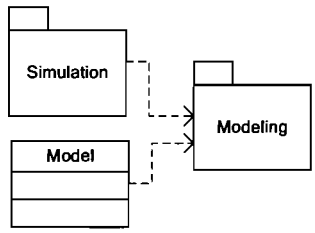


Figure 2. Idealized dependencies

3.2.1. Examples from SystemC and ESys.Net

In SystemC and ESys.Net, a designer has the responsibility to instantiate modeling concepts such as **Event** and **Signal** class instances. The issue here is that since the objects are instantiated by the designer, the implementation of those objects are determined indirectly by the designer. Since the **Event** and **Signal** objects contain code that are directly related to the implementation of the simulation environment, the model created by the designer is indirectly coupled to a simulation implementation. One could argue that it is possible to change the simulation implementation binding of a model changing the implementation to which it is link (such as with Space [18]), but when the model is compiled the binding becomes permanent. As stated earlier, a model should, at all times, only be dependent on the modeling constructs and not the simulation implementation. Since a SystemC and ESys.Net model is directly and indirectly dependant on a simulation implementation, it does not respect true separation of concerns. Another example from SystemC is the concept of a ResquestUpdate in primitive channels. This class method is used to synchronize a primitive channel instance with the delta cycles of a simulation implementation. The concept of a delta cycle should not be present in a pure model, for it has nothing to do with modeling. The only true advantage, with respect to separation of concerns, which ESys.Net has over SystemC is that the simulation infrastructure is not present once models are compiled.

3.2.2. Examples from Ptolemy

In the Ptolemy environment, domains, which are implementations of models of computation, rely on the concept of actors and a director. The actors implement the computation that must take place and the director orchestrates the implementation of the domain. A designer when creating a model must implement and/or assemble predefined actors of the domain. When implementing an actor, the designer must make calls to the director. This implementation style does not respect true separation of concerns for the director has much more to do with simulation then with modeling; hence, there is not true separation between modeling and simulation semantics.

4. SOC MODELING LANGUAGE (SOCML)

4.1. Overview

SoCML is a modeling framework inspired by SystemC and ESys.Net; its implementation only offers a subset of the modeling semantics of the later but a complete implementation could easily be achieved. The main objective of the SoCML project was not the implementation of a modeling framework but rather the demonstration of software framework design techniques which could achieve separation of concerns between system modeling and simulation aspects. SocML contains the concepts of ports, signals, modules, and channels with posses the same semantics as the same named concepts in SystemC and ESys.Net. The major difference is that the semantics are defined only with the aid of abstract classes, virtual empty methods, interfaces, and attributes. Here is the complete list of all the interfaces defined in the modeling framework :

```
public interface Input { }
public interface Output { }
public interface InOut : Input, Output { }

public interface inPort<t> : Input { t Value { get;} }
public interface outPort<t> : Output { t Value { set;} }
public interface inoutPort<t> : inPort<t>, outPort<t> { }

public interface isensitive { inEvent Sensitive { get;} }
public interface ipositive { inEvent Pedge { get;} }
public interface inegative { inEvent Nedge { get;} }

public interface sinPort<t> : inPort<t>, isensitive { }
public interface sinoutPort<t> : sinPort<t>, outPort<t> { }

public interface binPort : sinPort<bool>, ipositive, inegative {}
public interface binoutPort : binPort, outPort<bool> { }

public interface signal<t> : sinoutPort<t> { }
public interface bsignal : binoutPort { }

public interface clock : binPort {}

public interface inEvent {}
public interface outEvent {
   void Cancel();
   void Notify();
   void Notify(long time);
}

public interface biEvent : inEvent, outEvent{}}
public interface var<t> : isensitive { t Value { get;set} }
```

Here is a complete list of the classes defined in the framework:

```
public abstract class BaseModule {
  //Methods with the name Initialize are special to the environment
  //These methods act like constructor and should only contain call to
  //initialize methods of sub-modules and sub-channels


   public virtual void Initialize(){}
   protected virtual void SectionPortBinding() { }
   protected virtual void Wait() {}
   protected virtual void Wait(long time) { }
   protected virtual void Wait(inEvent ev) { }
}

   public abstract class BaseChannel : BaseModule{}
```

The only semantic differences between SoCML and SystemC that are worth noting are:

- the separation of the ***Event*** concept into three sub/super concepts which have directionality;
- a new concept called ***Var*** which represents a variable that may be updated and read in parallel like a signal;
- the SectionPortBinding which is a class method that should contain only port binding code in user defined modules and channels;

- the special treatment of methods found in modules and channels called Initialize. These methods are equivalent to constructors and should only contain method calls to Initialize methods on sub-modules and sub-channels. Initialize methods should only be called once and class constructors should not be used for reasons that will be explained later.
- A top level module which represents a complete model should use a class constructor; however, the constructor must delegate directly and immediately to an Initialize method.

It should be noted that there is no implementation code at all in the modeling framework. This is intentional for as stated earlier: "a modeling framework should only contain modeling semantics".

Here is a simple example of a produce–consume model:

```
public class Consumer : BaseModule {

  public inEvent sync;
  public sinPort<int> input;

  [NonBlockableProcess, Sensitive("input")]
  protected void Consume() {
    Console.WriteLine(input.Value.ToString());
    Wait(sync);
  }
}
```

The model looks very similar to SystemC and ESys.Net. In the consumer module, the **NonBlockableProcess** attribute is equivalent to an SC_METHOD in SystemC. The **Sensitive** attribute indicates that the process is bound to the sensitive event of the signal bound to the port called input. The sensitive event has a clear semantic meaning which is "sensitive to writes on the signals no matter what the value. In the context of a **NonBlockableProcess**, the Wait method call has the same meaning as a next_trigger in SystemC.

In the producer module, the **Blockable** attribute is equivalent to an SC_THREAD in SystemC. The semantics of the producer should be interpreted in the same manner it would be in SysemC.

```
public class Producer : BaseModule {

  public outEvent sync;
  public binPort clock;
  public outPort<int> output;
  private int i = 0;

  [BlockableProcess , Sensitive("clock")]
  protected void Produce() {
    output.Value = ++i;
    Wait(35);
    sync.Notify(25);
    output.Value = ++i;
  }
}
```

```
[ClockDomain(20)]
public class Model : BaseModule {
  private signal<int> sig;
  private clock clk;
  private Producer producer;
  private Consumer consumer;
  private biEvent sync;

  protected override void SectionPortBinding() {
    producer.clock = clk;
    producer.output = sig;
    consumer.input = sig;
    producer.sync = sync;
    consumer.sync = sync;
  }
}
```

Here is an example of a model which used the producer and consumer modules, we can notice the use of the SectionPortBinding method. The **ClockDomain** attribute defines a clock frequence for all clocks defined in its scope. A **ClockDomain** attribute may be assigned to a particular clock in order to override its parent's scope. The objective of this example is not to demonstrate all the possibilities of the modeling framework but to show how semantic found in SystemC and Esys.NET may be defined.

It should be noticed that there are no object instantiations in the model. This information is intentionally left out for two main reasons:

- Object instantiation adds no information to the model. One only has only to imagine that as in C++ the objects are instantiated on the stack and not on the heap because the new keywork is not used.
- By delaying the instantiation of the objects until need (such as at simulation time), we allow the implementation of the semantics to be chosen depending on the context; for example this allows a simulator to instantiate its implementation of the semantic in order to construct the model.

4.2. Design Constraints imposed on SoCML

As mentioned earlier, the main objective of SoCML was the implementation of clear and unambiguous modeling semantics through the use of an object-oriented framework-based approach. In order to keep the framework "clean" of all simulation semantics and artefacts we started with the analysis of ESys.Net in order to determine the implementation elements that had to be eliminated from the framework.

There were two main types of elements which had to be eliminated: method bodies within the framework which where biased towards a certain simulation implementation and framework classes which had to be instantiated which contained code which was biased towards a certain simulation implementation. The two types of elements are clearly exemplified by the code in the Wait methods of the **BaseModule** class and the **Event** class.

The Wait method contains code which pauses the current executing thread in order to switch to the simulation kernel thread. The method also contains code which makes calls to internal methods of the simulation kernel. The **Event** class contains code which permits the scheduling of event instances by making calls to internal methods of the simulation kernel.

One design approach which could be used to loosely couple the core modeling classes from the core simulation classes is the use of a service contract. With this approach, the service contract offered by the simulation core to the modeling core would have been define using an interface type. The modeling core would use the interface type to interact with the simulation core. The only difficulty with this approach is the passing of an implementation of the service contract to the modeling core, however many implementation strategies exist. We did not adopt this approach because it did not fulfill the need to have a complete separation between the two concerns; it only weakened the coupling between both concerns. In order to achieve the required separation, we used only software interfaces to describe all the modeling semantics which we needed. By using interfaces, we eliminated both the problem of instantiation and of biased code fragments. The only semantics which we decided to keep as classes within the framework were Module and

Channel. We made this choice because the use of class inheritance in order to use those semantics permitted a simple and elegant solution

5. A SIMULATOR FOR SOCML

In order to complete the demonstration of our modeling framework design approach, we created a simulation framework for SoCML. In order to achieve our implementation goals, it was necessary to find solutions to the constraints we imposed on the modeling framework. We had to find a way to instantiate implementations for variables contained within a model which where of interface types defined by the modeling framework. We also had to find a way to implement the method bodies of virtual Wait methods in the **BaseModule** class.

5.1. Class Instantiation Problem

The problem of instantiation of an implementation of a variable of a given type is basically the problem of class instantiation. The software design pattern called *Inversion of Control* is a perfect solution for this kind of problem.

5.1.2. Inversion of Control

The software community has a software design pattern which is a solution for a similar problem: *Inversion of Control*.

IoC is a design pattern which enables the decoupling between types [19]. Decoupling is achieved through (1) the use of explicit contract dependency declarations – the term declaration is used here in an implementation agnostic way; (2) the elimination of direct instantiation of a contract implementation by a type instance; and (3) the consummation of a dependency through an interface. Put simply, a type instance designed according to IoC does not instantiate objects which fulfils its dependency needs but rather delegates the instantiation responsibility to an execution environment and consumes the dependencies through interfaces which hide the implementations of the contracts. The execution environment, through the use of a defined dependency need declaration paradigm, locates an implementation for each required dependency of a class instances and instantiates it. Once the implementation instantiated, the environment gives the requester access to it through another defined convention. The environment portion of the pattern is often referred to as a container. IoC is sometimes referred to as Dependency Injection or The Hollywood Principle (Don't call us we'll call you).

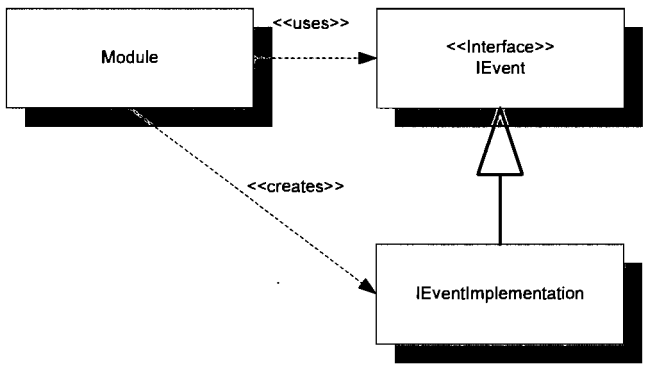


Figure 3. Typical type instantiation

The basic principals of IoC to remember are:

- High level modules should not depend upon low level modules. Both should depend upon abstraction.
- Abstraction should not depend upon details. Details should depend upon abstractions.

Figure 3 represents a typical UML diagram depicting class dependencies between a requestor and a dependency. The **Module** object uses the **IEvent** interface in order to interact with an **Event** object but is must also instated the implementation of the **IEvent** interface. The **Module** object is the requestor and the **Event** object is the dependency. In most current modeling solutions (ESys.Net and SystemC), the **IEvent** interface does not exist but in substituted for the implicit interface of their respective event classes.

Figure 4 represents a UML diagram depicting a modified version of Figure 3 using the IoC pattern.

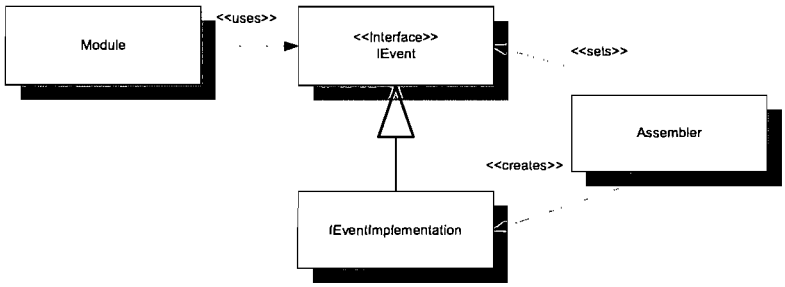


Figure 4. Inversion of control

In the modified diagram, the **Module** object no longer has the responsibility of instantiating the implementation of the interface. The instantiation task is delegated to an **Assembler** object.

5.1.3. Typical Implementation Techniques

Three mainstream techniques exist in order to implementation the declaration of dependencies and the passing of dependency instances between a dependency requestor and a container: contextualized lookup, constructor injection and setter injection. The three techniques use the concept of a container which is an execution environment for the requestor which acts as the **Assembler**.

Contextualized Lookup

Contextualized Lookup is a technique in which the container makes dependencies available via an interface/method that the requestor implements to indicate that is has dependencies. The interface/method, which is implemented by the requester, typically receives a reference to a lookup service.

```
public class Module : Serviceable{
    IEvent event;
    public void service(ServiceManager sm) {
        event = (event) sm.lookup("IEvent");
    }
}
```

The above implementation uses a lookup service which instantiates the **IEvent** object for the **Module**; dependency information is often stored in a configuration file for the lookup service to use.

Constructor Injection

Constructor Injection is a technique in which the container makes dependencies available to a requestor via a class constructor. Dependency declaration information is often retrieved via reflection on the constructor. The container is responsible for instantiating objects and passing dependency implementations.

```
public class Module {
    private IEvent event;
    public Module (IEvent event) {
        this. event = event;
    }
 }
```

Setter Injection

Setter Injection is a technique in which the container makes dependencies available via setter methods after instantiation.

```
public class Module {
   private IEvent event;


/**
    * @service name="IEvent"
    */
   public void setIEvent(IEvent event) {
       this. event = event;
   }
}
```

By its very nature IoC involves loss coupling between service requesters and services providers. This loss coupling promotes easier code maintenance, easier code reuse, and a lot of flexibility.

5.1.4. Inversion of Control With Reflection

The mainstream techniques used to implement IoC are usually satisfactory in the context of business applications, but they are not sufficiently transparent to be used in the context of a modeling framework. It would be necessary to "pollute" the modeling framework with IoC implementation mechanisms which have nothing to do with modeling.

If we come back to the IoC design pattern, the basis of the pattern is to create a contract with the aid of an interface, which permits a service requestor to declare the need of a service whose implementation will be chosen by a container. This pattern applies very well to the model/modeling/simulation tuplet :

- the model may be seen as the requestor;
- the modeling framework may be seen as the service interface definitions;
- the simulator may be seen has the container;
- the interface declaration in the model may be seen as a service requests.

Through the used of reflection [5, 20] , a simulator can dynamically discover the interface declarations and understand them as service requests.

Our SoCML simulation frameworks uses model analysis through reflection in order to act has a container. The core of the analysis is very similar to the one used in ESys.Net, the only significant difference is the instantiation upon detection of the modeling semantic declarations. The analysis and implementation strategy used by the simulator of the simulation framework is made possible by the runtime resolution of generic in the .Net framework. The strategy could not have been used by an implementation in Java or C++.

Here is a fragment of the pseudo-code used by the core model building algorithm of our simulator.

```
BuildModel(BaseModule module) {
      mduleType = GetType(module);
      fieldDefinitions = GetFields(moduleType);
  foreach   field in fieldDefinitions {
    fieldInstance = GetInstance(field,module);
    fieldType = GetType(field)
    If(fieldInstance is unset){
    Select(fieldType){
       Case inEvent,outEvent,biEvent :
         FieldInstance = new EventImplementation;
       Case clock :
         fieldInstance = new ClockImplementation;
       Case signal :
         boundType = GetGenericBoundType(fieldType)
         customeSignalType =
            CreateSignalType(boundType,ImplSignalType)
         fieldInstance = new customeSignalType
       Case module
         proxy = CreateProxy(fieldInstance)
         fieldInstance = proxy;
    }
  }
  Foreach subModule in module BuildModel(subModule)
}
```

We can clearly see the use of the IoC design pattern in the above code fragment. We may view the model as the service requestor and the simulator as the container. The handing over of the service is done in an alternative way from the ones presented earlier. The model declares its need of a service by declaring a variable of a type defined in the modeling framework. Through introspection of the model, the simulator finds the service declarations and sets them with an instance of an appropriate implementation. The simplicity and elegancy of the solution is made possible by the reflective and dynamic generic capabilities of .Net.

5.2. Virtual Method Implementation Problem

The problem of implementing a virtual method without the consumer being aware often arises in the context of distributed applications. In traditional distributed applications, consumers use an object which impersonates a remote object. The responsibility of the impersonating object is to offer a simple interface — usually an interface which is identical to the remote object — to the consumer and marshal the calls to the remote object. The same basic technique can be use for a virtual method implementation problem. The technique is based on the proxy design pattern.

5.2.1. Proxy Design Pattern [8]

The proxy design pattern is one the structural patterns defined by the GoF. The GoF defines structural patterns as:

"Structural patterns are concerned with how classes and objects are composed to form larger structures. Structural class patterns use inheritance to compose interfaces or implementations. As a simple example, consider how multiple inheritances mixes two or more classes into one. The result is a class that combines the properties of its parent classes. This pattern is particularly useful for making independently developed class libraries work together."

The intent of a proxy is to provide a surrogate or placeholder for another object to control access to it. There are various flavors of proxies depending on their usages such as:

- remote, where you represent a remote object through a local object;
- virtual, which provides on demand creation of expensive objects;
- protection, which controls access to the original object;

- smart reference, also known as a smart pointer, which provides "decorated" functionality to the proxied object (such as a smart pointer, persisted object loader, or wrapper object for multithreaded operations to a single-threaded object.)

Here is the UML diagram of the pattern:

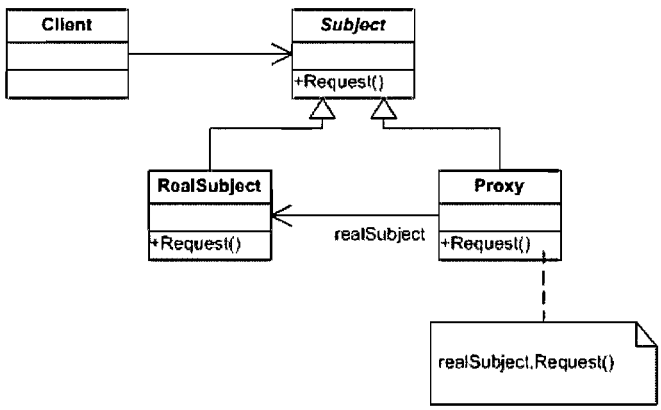


Figure 5. Proxy pattern

The proxy maintains a reference that lets the proxy access the real subject. The proxy may refer to a Subject if the RealSubject and Subject interfaces are the same. It also provides an interface identical to Subject's so that a proxy can by substituted for real subject and controls access to the real subject and may be responsible for creating and deleting it. The proxy may have other responsibilities depending on the kind of proxy.

The Subject defines the common interface for RealSubject and Proxy so that a Proxy can be used anywhere a RealSubject is expected.

RealSubject defines the real object that the proxy represents.

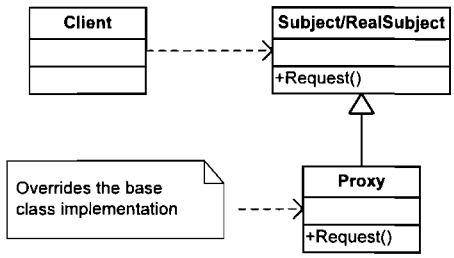
5.2.2. Combining IoC and Proxy Design Pattern

In traditional distribution applications, the developer is aware that he is using a proxy object for he usually either requests an instance of the proxy from a factory style software layer or he must explicitly instantiate it. Both of these approaches were not suitable for the implementation of the simulation framework for we did not want the system modeler to have to be aware of the underlying mechanics of the simulator in order to achieve separation of concerns.

In order to solve the problem, we used a combination of the IoC and proxy design patterns. In the simulation implementation that we propose, the implementation of the Wait methods of a **Module** or **Channel** are achieve by the use of a proxy which catches the calls and delegates them to the simulator. The interception of the calls is achieved by:

- Creating dynamically a derive type from a user-defined module/channel type at runtime.
- Overriding all the virtual Wait methods with an implementation which delegates the call to directly/indirectly the simulator.

In our implementation, the Subject and RealSubject are both the same type; a user defined module or channel type. To be more precise, we can state that the Subject is the public interface of a user-defined module/channel and the RealSubject is the implementation of that public interface. In order to obtain a type which derives from the Subject we must derive from the user-defined type.



We achieve the creation of the proxy class by using the DynamicProxy.Net framework distributed by the Castle project [21]. The framework supports the creation of proxy type from generic types.

```
proxyGenerator.CreateClassProxy(type, new MyInterceptor(this)));
```

The above line of the code is the true method call which creates the dynamic proxy, the method call takes as parameters a type and an interceptor object. The interceptor object will receive all the methods calls made on virtual methods overridden in the proxy class.

```
internal class MyInterceptor : StandardInterceptor {
     private Simulator manager;
     public MyInterceptor(Simulator manager) {
       this.manager = manager;
     }

     public override object Intercept(IInvocation invocation,

       params object[] args) {
       if (invocation.Method.Name == "Wait") {
          if (args.Length == 0) {
            manager.Wait();
          } else if (args.Length == 1) {
            if (args[0] is long) {
              long t = (long)args[0];
              manager Wait(t);
            } else {
              manager.Wait(args[0] as Event);
            }
          }
       } else {
          base.Proceed(invocation, args);
       }
       return null;
     }
}
```

The above code is the interceptor type we use to catch the wait method calls. In the DynamicProxy.Net framework, all the calls made on a generate proxy are delegate to the Intercept method of a user-defined interceptor from management. Our decision to use the DynamicProxy.Net framework was made because it was the simplest and quickest way for us to achieve are proof of concept implementation.

6. BENEFITS OF OUR APPROACH

The design approach we used for the implementation of our modeling and simulation solutions has many subtle but very important benefits that we will present in this section. The benefits that we will presents do not add significant value to the semantic capabilities of the modeling solution or the efficiency of the simulation solution but rather "opens the

door" to news possibilities. These new possibilities are enabled because of the separation of concerns that we have achieved between modeling and simulation aspects.

6.1. Perfect Separation of Concerns

As mentioned earlier, to our knowledge, all current modeling/simulation solutions that are based on a framework-oriented approach do not posses a clear boundary between modeling and simulations aspects in their design and/or implementation.

The modeling framework that we have presented possesses no dependencies on a simulation solution implementation. It depends only on a well defined set of modeling concepts and a particular model of computation. Because of this separation, system models that used the modeling framework, by transitivity, are themselves independent of a simulation solution. These models, by the means of .Net, may have multiple "clean" representations as depicted in the diagram below. The importance of these clean models of representation is explained in [5].

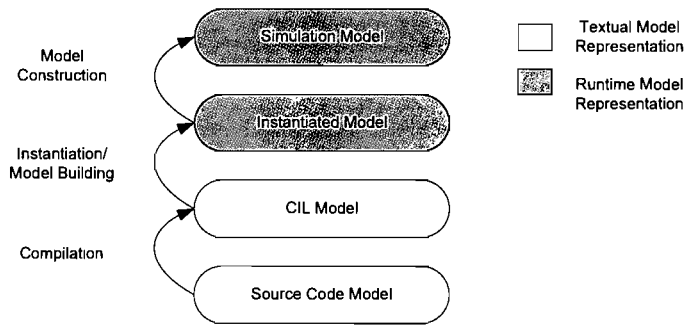


Figure 6. Model representation stacking

We believe that compiled versions of these "clean" models and modules may serve as the backbone for simulation implementation agnostic intellectual property (IP) block-based designs. It is very difficult to achieve true IP block reuse in solutions such as SystemC, for once compiled, IP blocks are statically bound to a specific simulation solution implementation. Separation from specific simulation solutions is fairly important, especially in the context of SystemC, for multiple implementations of the SystemC framework exist and each supports a different toolset. In this context, an IP block which is not independent of a simulation implementation may probably not be used with certain tools. The artificial binding to specific simulation implementations hinders the creation of custom design flows.

6.2. Verification by Interception

The utilization of design patterns such as IoC and Proxy enable a simulation solution to create chains of interceptors which can monitor different aspects of a model under simulation without having to add the verification elements in the model itself or having to create a complex verification enabled layer (API) such as SystemC's SVC.

One can easily imagine an implementation of the **Var** modeling interface we presented which allows a tool external to the simulator to be notified on modification in order to drive linear temporal logic (LTL) expression verification. Another example could be a simulation implementation which allows a verification tool to chain interceptors in a proxy chain in order to monitor the number of method calls on a channel. The combination of a flexible framework design and the reflective capabilities of the .Net framework offers many possibilities for the creation of effective tools at low cost.

6.3. Alternative Model Simulations

Since all models created with the modeling framework are independent of a particular simulation solution implementation, it becomes possible to use a model with different simulators in order to take advantages of alternative implementations. Alternative simulation implementations could offer:

- different performance characteristics

- different monitoring characteristics
- different verification characteristics
- different tool support
- distributed simulation
- software vs hardware simulation for Co-design (Space)
- support heterogeneous Model Of Computation simulation [12, 23].

It is difficult to imagine all the possible simulation solution implementations but what is clear is that a model could be transparently used with various simulation implementations without having to be modified or recompiled, as long as both use the same modeling semantic contract.

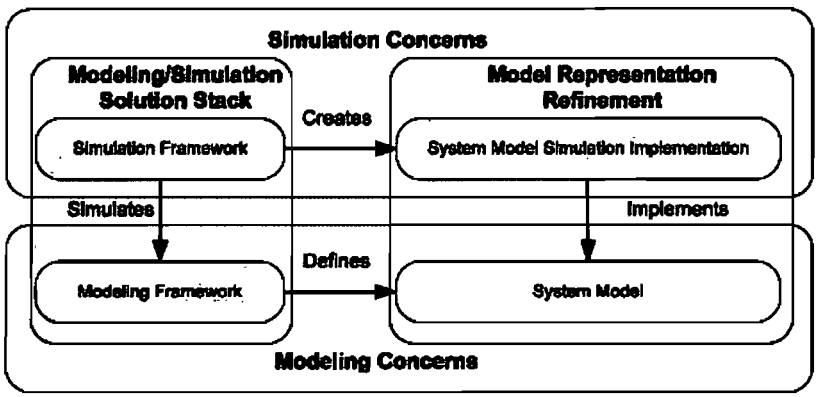


Figure 1. Modeling concerns

Figure 1 illustrates the various relations which exist between the elements of our approach. By using a stack-based approach between the modeling and simulation concerns, it is possible to create various modeling/simulation solutions without affecting models. An emulation solution could be implemented by using the same model analysis techniques based on reflection and IoC but could instantiate the user-defined modules types. The simulation would be achieve not by using a proxy pattern but by orchestrating the execution of multiple hardware elements in the simulation kernel and by updating the variables of the instantiate model, giving the illusion that it is the user model which is being simulated by the kernel. In this context, the instantiate model serves the role of a projection of the emulation.

6.4. Simplicity and Elegancy of Modeling

In our opinion, a model designer should only have to worry about the structural and behavioral aspects of his model and not the implementation constraints of tools which will process the model (*e.g.* simulator). Our statement might seen trivial but in frameworks such as ESys.Net and SystemC, a designer must respectively use the RequestUpdate and Update class methods of the **Channel** base class in order to schedule a primitive channel to be updated during delta cycles in order to perform variable "housekeeping". Should a system designer have to know that the model might be executed with a simulator which implements the concept of a delta cycle? What he really wants is the concept of a variable that can support concurrent reading and writing.

```
class APChannel<t> :PChannel<t>{
    public Event sensitive = new
Event;
    private t currentvalue;
    private t oldvalue;
    private t newvalue;

    public t Value {
      get { return currentvalue; }
      set {
        newvalue = value;
        RequestUpdate(this);
      }
    }

    public override void Update() {
      if (currentvalue != null) {
        if
(!currentvalue.Equals(newvalue)) {
          oldvalue = currentvalue;
          currentvalue = newvalue;
          sensitive.Notify(0);
        }
      } else {
        oldvalue = currentvalue;
        currentvalue = newvalue;
        sensitive.Notify(0);
      }
    }
  }
```

Esys.Net

```
class APChannel <t> : signal<t>{
    private var<t> value;

    public t Value {
      get { return value.Value; }
      set { value.Value = value;}
    }

    public inEvent Sensitive {
      get {
          return value.Sensitive;
        }
    }

  }
```

SoCML

The above side-by-side code comparison demonstrates the simplicity with which a primitive channel may be modeled using our example modeling framework SoCML. The SoCML based model only contains modeling semantics; no simulation related elements are present. As mentioned earlier, the **Var** data type has the semantic meaning of a variable which supports concurrent reading and writing. The implementation of the **Var** concept would probably be similar to implementation approach of the RequestUpdate/Update protocol used in ESys.Net.

Having a simple modeling framework which is free of simulation implementation related information will definitely help designers be more productive. Software tools will also have an easier time analyzing models for they will not have to discard non-modeling related elements. In a perfect dedicated modeling language, each necessary semantic notion would probably be expressed using a small and simple list of key words. We believe that a modeling framework based only on attribute programming, interfaces, and abstract class containing only virtual methods brings modeling framework-based languages much closer to a dedicated modeling language than traditional solutions.

7. FUTURE RESEARCH

The ideas in this article illustrate our vision for the next generation of framework-oriented system modeling languages. This next generation of solutions will be characterized by perfect separation of concerns between modeling and simulation aspects, which may be achieved by applying our design guidelines.

This work opens the door to many other projects such as creating a new SystemC-style modeling solution. It would be interesting to explore the impact of our design guidelines on the design and implementation of a heterogeneous model of computation environment. It would also be interesting to revise the syntax of Metropolis according to the guideline of this work.

Our team is currently working on a redesign of the ESys.Net modeling/simulation framework according to the guidelines we have presented and we are investigating different backend implementation for the simulation engine. We

are also working on the same kind of separation of concerns concept but applied to verification and model constraint aspects.

8. CONCLUSION

Many believe that framework-based modeling/simulation solutions will allow designers to model systems more effectively and will facilitate the creation of custom design flows. Moreover, many believe that framework-based solutions are a good approach to heterogeneous "model of computation" (MOC) and "co-design" simulation. In the mist of all the work which has been done in order to create such framework-based solutions, very few have worked on the design guidelines that such solutions should follow in order to create effective object-oriented frameworks.

In this article, we presented the current lack of separation of concerns which is present in most mainstream modeling/simulation framework-based solutions. We argue about the importance of such separation and its benefits. We present "state of the art" software design patterns and technologies which can promote better separation of concerns between modeling and simulation aspects when applied to modeling/simulation frameworks. Finally we present a novel modeling framework called SoCML which follows our guidelines. By its design, SoCML presents all the benefits which were enabled by the approach:

- simplicity and elegance of the models which use the framework;
- independence of models from a particular simulation implementation allowing their reuse with alternative simulation approaches;
- verification by inception enabled;
- "clean" model representations.

In order to demonstrate the feasibility of our approach, we present a simulation implementation for SocML.

REFERENCES

[1] ITRS 2005. *International Technology Roadmap for Semiconductors, Edition 2003.* Available from: http://public.itrs.net/

[2] David I. Rich, "The Evolution of SystemVerilog," *IEEE Journal Design & Test of Computer*, **20**(2003), p. 82.

[3] THE OPEN SYSTEMC INITIATIVE (OSCI) 2005. *SystemC 2.1 Language Reference Manual.* Available from: http://www.systemc.org/

[4] SYNOPSYS, *COCENTRIC® SYSTEM STUDIO*, 2001.

[5] James Lapalme, El Mostapha Aboulhamid, and Gabriela Nicolescu, "A New Efficient EDA Tool Design Methodology", *Journal of ACM Transactions on Embedded Computing Systems (TECS)*, **5**(2006), p. 408

[6] Nicolas Gorse, Michel Metzger, James Lapalme, El Mostapha Aboulhamid, Yvaon Savaria and Gabriela Nicolescu, "Enhancing ESys.Net with a Semi-Formal Verification Layer", *in Proceedings of the 16th IEEE Intl Conference on Microelectronics (ICM'04)*, 2004 , p. 388.

[7] James Lapalme, El Mostapha Aboulhamid, Gabriela Nicolescu, Luc Charest, Jean-Pierre David, Francois Boyer, and Guy Bois, "ESys.NET: A New Solution for Embedded Systems Modeling and Simulation," *in Proceedings of the ACM SIGPLAN/SIGBED 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04)*, 2004, p. 107.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Massachusetts : Addison–Wesley, 1994.

[9] Edsger W. Dijkstra, *Selected Writings on Computing: A Personal Perspective.* New York: Springer-Verlag, 1982, p. 66.

[10] MICROSOFT 2005. *Microsoft .NET Framework.* Available from: http://msdn.microsoft.com/netframework/

[11] ECMA/ISO 2006. ECMA (334-335) and ISO/IEC (23270-23271), *C# and Common Language Infrastructure Standards*. Available from: http://www.ecma-international.org/

[12] *Ptolemy Project.* http://ptolemy.eecs.berkeley.edu/

[13] Peter Bellows and Brad Hutchings, "JHDL: AN HDL for Reconfigurable Systems," *in Proceedings of the IEEE Symposium on FPGAs For Custom Computer Machines*, 1998, p. 175.

[14] Marjan Mernik, Jan Heering, and Anthony M. Sloan, "When and How to Develop Domain-Specific Languages", *in ACM Computing Surveys*. New York : ACM, 2005, p. 316.

[15] Daniel D. Gajski, Jianwen Zhu, Rainer Dömer, Andreas Gerstlauer, and Shuqing Zhao, *SpecC: Specification Language and Methodolog*. New York:, Springer, 2000, p. 336.

[16] IEEE 2006. IEEE 1076.3 VHDL.

[17] IEEE 2006. IEEE 1800 SystemVerilog.

[18] Jerome Chevalier, Olivier Benny, Mathieu Rondonneau, Guy Bois, El Mostapha Aboulhamid, and Francois-Raymond Boyer, "Space: A Hardware/Software SystemC Modeling Platform Including an RTOS", *in Source Languages for System Specification*. Massachusetts: Kluwer Academic Publishers, 2004, p. 91.

[19] Griffin Caprio, "Dependency Injection", *MSDN Magazine*, **20**(2005).

[20] Frederic Doucet, Sandeep Shukla, and Rejest Gupta, "Introspection in System-Level Language Frameworks: Meta-Level vs. Integrated.", *in Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2003, p. 382.

[21] *The Castle Project - DynamicProxy.Net*. http://www.castleproject.org/

[22] Gabriela Nicolescu, Sungjoo Yoo, Aimen Bouchhima, and Ahmed A. Jerraya, "Validation in a Component-Based Design Flow for Multicore SoCs.", in *Proceedings of the 15th international Symposium on System Synthesis (ISSS)*, 2002, p. 162.

[23] Ahmed Jerraya and Rolf Ernst, "Multi-Language System Design.", in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 1999, p. 696.

L'impact du Web sémantique sur la conception des systèmes assistée par ordinateur

La vision du Web sémantique a été définie, pour la première fois en 1999, par Tim Berners-Lee [10]. Cette vision décrit la prochaine évolution du Web. Un Web qui s'autodéfinit et dont la consommation est aussi simple pour les machines que pour les humains. Le Web sémantique n'est pas une remise en question du Web « Classique » mais une extension de celui-ci qui va au-delà de la simple publication et consultation de documents. L'objectif principal du Web sémantique est le partage de la connaissance. Ce partage de la connaissance ne se fait pas seulement par le partage des données et des formats d'encodage mais par le partage de métadonnées non ambiguës.

La vision du Web sémantique a aidé les domaines des sciences de la vie et de la pharmaceutique [7][61]. L'utilisation du Web sémantique a permis l'émergence de nouvelles connaissances par l'utilisation d'une base de connaissances collective. Cette émergence du savoir par le partage des connaissances a permis la découverte de nouveaux mécanismes d'interactions génomiques et pharmaceutiques.

Dans le domaine de la CAO, la gestion des données et des métadonnées joue un rôle clé dans le contexte de la conception niveau système. Plusieurs flux de conception utilisent un patron consistant à faire l'assemblage de composantes relativement simple — sous-systèmes et IP — afin de définir des systèmes complexes. Ce processus d'assemblage est très dépendant sur les métadonnées qui décrivent les composantes assemblables.

Depuis l'adoption des technologies XML par l'industrie, plusieurs utilisent celles-ci pour la gestion des données et des métadonnées. Cette adoption est grandement due

au fait que XML a démocratisé l'implémentation de format de représentation de données.

Dans le domaine de la conception des systèmes embarquées, plusieurs projets tel que Colif [17]et MoML [51] ont utilisé XML pour représenter des architectures de systèmes embarqués. XML est une solution très efficace pour la définition et la manipulation de syntaxe pour la représentation d'information, par contre elle n'a pas été conçue pour la gestion des métadonnées ainsi que leur sémantique. Par conséquent, les solutions qui se fondent sur XML pour l'échange de données sémantiquement non ambiguëes doivent traiter beaucoup de points faibles. Puisque le Web sémantique est la prochaine étape en technologies de représentation de métadonnées, elle peut être un outil de grande valeur pour l'industrie de la CAO. La gestion de métadonnées basée sur RDF et OWL [4], deux technologies du Web sémantique, bénéficie d'une fondation très robuste dédiée à la sémantique. Les travaux tel que [70] démontrent que les technologies du Web sémantique peuvent être effectivement appliquées au domaine des systèmes embarquées. [70] présente comment les technologies du Web sémantique peuvent être utilisées conjointement avec les technologies orientées services afin de créer un environnement de développement intégré flexible pour la conception de système embarqués.

Le projet SPIRIT est un autre exemple d'efforts investis dans la définition d'un environnement de développement pour la conception de systèmes embarqués [46]. Son principal objectif est la définition de normes et de standards pour la conception d'un environnement de développement (ide) intégré pour la conception de systèmes basés sur les IP. Actuellement, le projet SPIRIT a adopté une approche plus traditionnelle basée sur XML que [70] pour la gestion des métadonnées et des données connexes par IP. Cette approche a inutilement complexifié les normes et les standards produits, ainsi que la gestion de leurs versions.

Les contributions principales de cet article sont:

- la présentation détaillée des concepts du Web sémantique dans le contexte de la CAO;

- la présentation et la discussion des avantages des technologies du Web sémantiques par rapport aux technologies XML pour la gestion des données et des métadonnées dans le contexte de la conception de systèmes;
- la présentation de l'application des technologies du Web sémantique aux standards du projet SPIRIT, ainsi que la présentation des bénéfices de celle-ci;

Les pages suivantes contiennent une copie du chapitre de livre [48], dans son format original (sauf la numérotation des pages), qui sera publié dans *System level design with .Net technology, E.M. Aboulhamid and F. Rousseau Eds., CRC Press.*

1

The Semantic Web Applied to IP-Based Design : A Discussion on IP-XACT

James Lapalme
DIRO Université de Montréal - Canada

El Mostapha Aboulhamid
DIRO Université de Montréal - Canada

Gabriela Nicolescu
Polytechnique de Montréal - Canada

1.1 Introduction

The Semantic Web [7] was first envisioned by Tim Berners-Lee in 1999 as the next evolution of the Web. A Web that was self-describing and easily consumable by machines, not just humans. The Semantic Web is all about knowledge sharing. The sharing of knowledge cannot be achieved solely with the sharing of data and encoding formats but through the sharing of unambiguous metadata and meaning.

The Semantic Web vision has already helped the domains of life sciences and pharmaceutics [3, 27]. The Semantic Web has allowed researchers to inferred new knowledge and understanding by creating a collective knowledgebase. This sharing of knowledge has permitted the discovery of new genes and drug interactions. The Semantic Web has not only helped the sciences but also the field of resource manage-

ment. The NASA has implemented a successful project [18] to manage the expertise profiles of human resources. In this project the Semantic Web was a key enabler to integrate information across multiple systems.

In the Electronic Design Automation (EDA) domain, the management of data and metadata plays an important role in the context of system-level design. Many design flow methodologies use a pattern which consists of assembling reusable sub-systems and/or intellectual proprietary blocks (IP) to construct complex system platforms. This design process relies heavily on metadata. The formalisms used to describe system architectures within this process are often referred to as models of architecture (MoA). Over the last decade, special architecture-oriented languages have been developed in order to support these formalisms; we called them architecture description languages (ADL) [26] . Many projects such as Colif [12] and MoML [25] have built the syntax of their ADLs on the XML [15] technology stack as a means for metadata management and its manipulation.

From the perspective of syntax, XML based solutions are very effective. A vast number of free open source solutions are available which support the XML technology stack. However, XML was never intended for the management of metadata and its semantics. Hence solutions which rely on XML for the exchange of semantically unambiguous data must deal with many shortcomings. Since the Semantic Web is the next step in metadata representation technologies, it can be a valuable tool for the EDA industry. Metadata management based on RDF and OWL [2], both technologies of the Semantic Web, benefits from a very robust foundation focused on semantics. Work such as [31] demonstrates that the Semantic Web technologies can be effectively applied to the domain of embedded systems. The former presents how the Semantic Web technologies can be used in conjunction with service-oriented technologies in order to create a flexible integrated development environment for embedded system design.

Another project focused on the definition of a development environment for embedded system design is SPIRIT [24]. The SPIRIT project has gained a lot of momentum over the last couple of years. The project's principal objective is the definitions of standards for the design of an open integrated development environment (IDE) tool for IP-based system design. Currently, the SPIRIT project has taken a more traditional approach based on XML than [31] for the management of IP related metadata and data. This approach has overly complicated the standards of the projects, as well as the management of their versioning. This paper continues the discussion started in [31] by presenting the benefits of using the Semantic Web over XML for metadata and data management. It will also discuss how the SPIRIT project could benefit from using the Semantic Web.

The objectives of this paper are to introduce the Semantic Web to the EDA community, as well as discuss how initiatives such as the SPIRIT project can greatly benefit by adopting OWL instead of XML as a medium for metadata and data. The paper is organized as follows: sections 2-4 present background information on MoAs, IP-XACT and the Semantic Web technologies, section 5 discusses the semantic shortcomings of XML, section 6 discusses the advantages of the Semantic Web over XML, section 7 presents how SPIRIT could benefit form using the Semantic

Web technologies and section 8 discuses the cost of adoption.

1.2 Models of Architecture and XML

Models of architecture (MoA) [30, 22] are closely related to models of computation (MoC) [10]. In the same way that MoCs define semantic frameworks which permit the definition of computational-oriented applications, MoAs define semantic frameworks which allow the definition of system platform architectures. Presently, the state-of-the art of MoCs compared to MoAs is fare more mature. Many MoCs have been formally defined and studied in order to understand their inherent characteristics as well those of the applications which are defined with them. A mature concept of MoA has only begun to emerge in the last couple of years. Although no generally accepted definition of the term exist, two major approaches to MoA can be identified:

1. Describe platform components using existing MoCs;

2. Specify platforms using dedicated formalisms.

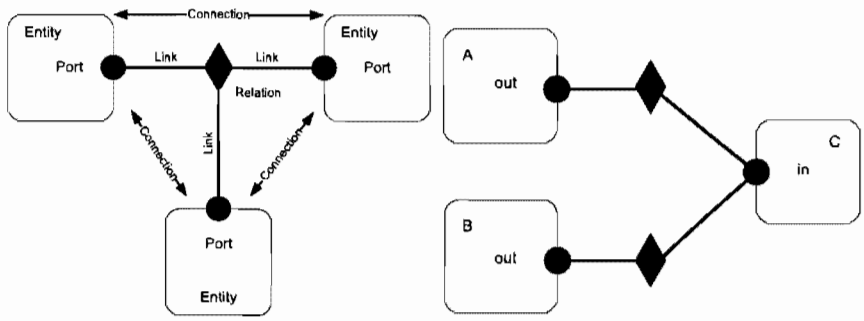
Some examples of the first approach are SystemC [17] , Metropolis [4] and the automated design flow based on Synchronous Dataflow in [29]. Each of these uses the same modeling constructs to define applications and platforms.

MoA is a key concept of some modern system design flow paradigms such as the Y-Chart approach [22]. In the Y-Chart approach, application models are defined using MoCs and platform models are defined using MoAs. A mapping is then defined between both models in order to specify how the application components are executed on the resources offered by the platform.

MoAs and ADLs are usually focused on describing the resources of architectures (platforms), their properties (area, energie, etc.) as well as their interconnection topology. In the context of distributed embedded systems, many researchers observed the existence of processor (e.g., general-purpose processors, accelerators, and dedicated controllers), communication (e.g., busses, network-on-chip, and i/o interfaces) and storage (e.g., memories, and hard disks) resources as elements of a MoA. There are many examples of ADLs which use XML as a meta-language for their definition. Through XML, these ADLs capture de definition of various architectural resources such as computational, communication and storage. They also allow the capture of system designs which are defined as interconnections of these resources. Hence, these ADLs manage both the metadata about resources as well as data about the designs which use the resources. The reminder of this section will present three key examples of ADLs which use XML.

1.2.1 GSRC and MoML

MoML is an XML modeling markup language [25]. It is a concrete syntax for the GSRC abstract syntax which was developed at UC Berkley in the context of the Ptolemy project. GSRC can be perceived as a MoA. GSRC, hence MoML, allows the specification of interconnections of parameterized, hierarchical components. MoML is extensible in that components and their interconnections can be decorated with data specified in some other language (such as another XML language). Figure 1.1(a) illustrated the key elements of the GSRC semantics, hence the elements which are encoded using MoML. The main concepts are entities, port and links.



(a) MoML concepts (b) MoML example

FIGURE 1.1: MoML concepts and example

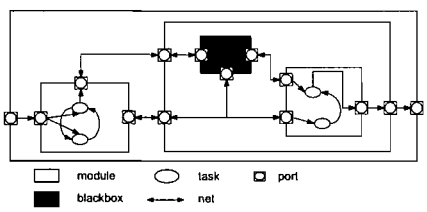
Figure 1.2 is the MoML representation of the example illustrated in Figure 1.1(b).

```
<model name="top" class="classname">
  <entity name="A" class="classname">
    <port name="out"/>
  </entity>
    <port name="out"/>
  <entity name="B" class="classname">
  </entity>
  <entity name="C" class="classname">
    <port name="in">
      <property name="multiport"/>
    </port>
  </entity>
  <relation name="r1" class="classname"/>
  <relation name="r2" class="classname"/>
  <link port="A.out" relation="r1"/>
  <link port="B.out" relation="r2"/>
  <link port="C.in" relation="r1"/>
  <link port="C.in" relation="r2"/>
</model>
```

FIGURE 1.2: MOML XML example

1.2.2 Colif and Middle-ML

In a similar fashion to the GSRC abstract syntax, Colif [12] defines a MoA with a focus on the description of application-specific multiprocessor SOC architectures (ASMSA). A key objective of Colif is to model on-chip communication at different abstraction levels while separating component behavior from the communication infrastructure. The concrete syntax for Colif is Middle ML, an intermediate language defined with XML. Middle-ML offers simple constructs in order to layer another languages on-top; Colif is one such language. Both Colif and Middle-ML have been developed by a research group at TIMA.



(a) Colif concepts

```
<?xml version-"11.0" encoding="UTF-8"?>
<!DOCTYPE middle SYSTEM "middle.dtd">
<middle>
...
  <typedef name="MODULE" structure="set">
    <typefield name="description">
      <typeref base="str"/>
    </typefield>
    <typefield name="entity">
      <typedef structure="ref">
        <typeref base="MODULE_ENTITY"/>
      </typedef>
    </typefield>
    <typefield name="content">
      <typedef structure="ref">
        <typeref base="MODULE_CONTENT"/>
      </typedef>
    </typefield>
  </typedef>
...
</middle>
```

(b) Middle-ML example

FIGURE 1.3: Colif and Middle-ML

Figure 1.3(a) illustrates the main semantic elements of Colif. The core concepts of Colif are: Modules, Ports and Nets. Figure 1.3(b) illustrates the use of Middle-ML to describe a Colif based architecture.

1.2.3 Premodona

Premadona [30] offers a tool for generating abstract performance models of Network-on-Chip based Multi-Processor System-on-Chips (MPSoCs) which are expressed wit the Parallel Object-Oriented Specification Language (POOSL) [32]. POOSL is an object-oriented system specification language which is based on a formal mathematical model. Moreover, is it the specification language for the Software/Hardware Engineering (SHE) methodology developed at the University of Eindhoven [32]. The *Premadona* tool follows the Y-Chart paradigm. Figure 1.4 illustrates the design flow with *Premadona*. Application models are defined using an XML language which uses the SDF3 specification language. SDF3 support the description of appli-

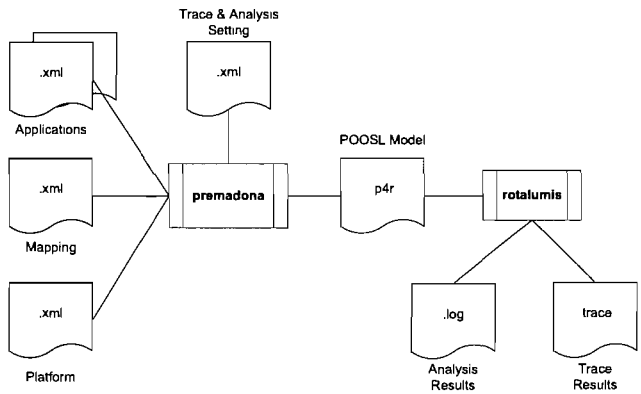


FIGURE 1.4: Premadona tool flow

```
<?xml version="1.0" encoding="UTF-8"?>
<platform name="MPSoC" version="1.0"
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns="uri:platform" xsi:schemaLocation="uri:platform
  http://www.es.ele.tue.nl/premadona/tools/platform.xsd">
  <node name="Node1" processor_type="MIPS"  scheduling_policy="PB"
    voltage_scale_factor="1" local_bandwidth="400000000"
    local_setup_latency="0.00001" />
  <node name="Node2" processor_type="ARM7"  scheduling_policy="PB"
    voltage_scale_factor="1" local_bandwidth="400000000"
    local_setup_latency="0.00001" />
  <node name="Node3" processor_type="TriMedia"  scheduling_policy="PB"
    voltage_scale_factor="1" local_bandwidth="400000000"
    local_setup_latency="0.00001" />
  <node name="Node4" processor_type="MIPS"  scheduling_policy="PB"
    voltage_scale_factor="1" local_bandwidth="400000000"
    local_setup_latency="0.00001" />
  <noc bandwidth="200000000" setup_latency="0.00002" />
  <power storage="0.000001" communication="0.0000005" />
  <processor_type name="MIPS" clock_frequency="167000000"
    context_switching_time="1500" power="0.075"/>
  <processor_type name="ARM7" clock_frequency="100000000"
    context_switching_time="1200" power="0.07"/>
  <processor_type name="TriMedia" clock_frequency="200000000"
    context_switching_time="3200" power="0.085" />
</platform>
```

FIGURE 1.5: Platform model

cation models using one of three MoC: synchronous dataflow, cyclo-static dataflow and scenario-aware dataflow. The specification language for platforms is also an XML based language which support MoA constructs such as processors, storage, etc. The *Premadona* tool generates POOSL models which can be simulated in order to evaluated performance.

Figure 1.5 illustrates a simple platform models composed of 2 MIPS processors, 1 ARM7 processor and 1 TriMedia processor.

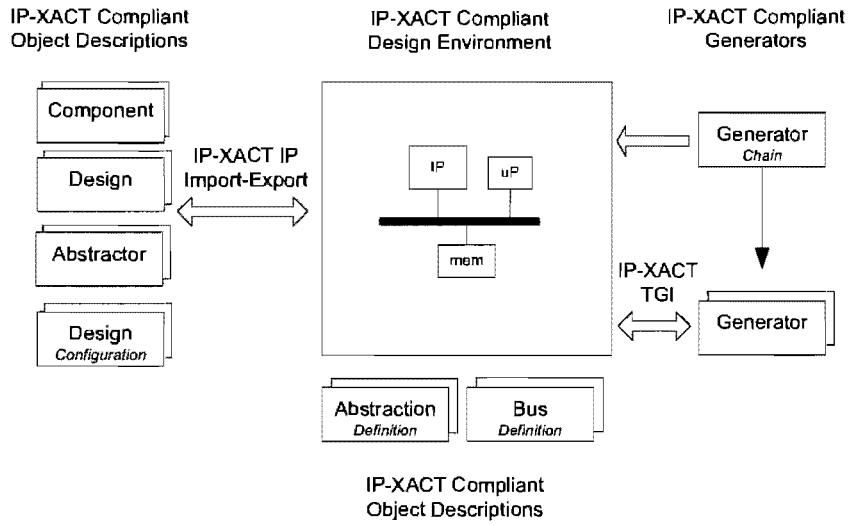


FIGURE 1.6: SPIRIT design environment architecture

1.3 SPIRIT

The SPIRIT Consortium [24] is a non-profit organization dedicated to the development of standards to empower the vision of IP-based development. At the heart of the SPIRIT vision is an open design environment (DE) which can support an IP-based design flow for the elaboration of embedded systems. The necessity of the SPIRIT vision has emerged because of the absence of standards for the packaging of IP descriptions and their related metadata. Currently, there is no design environment which can support IP descriptions across all vendors as well as integrate the necessary tools to supports them. Figure 1.6 illustrates the architecture of the design environment which is part of the SPIRIT vision.

In order to realize its vision, the consortium has defined a specification called IP-XACT which defined 3 mains sub-specifications: the IP-XACT metadata format, the Tight Generator Interface (TGI) and the Semantic Constraint Rules (SCRs). There are two obvious interfaces expressed in Figure 1.6: from the DE to the external object description libraries and from the DE to the generators. The IP-XACT metadata format is used for the interface between the DE and the object description libraries. The TGI is used between the DE and generators.

FIGURE 1.7: IP-XACT object description types

Objects	Meaning
Bus Definition Description	Defines the type attributes of a bus.
Abstraction Definition Description	Defines the representation attributes of a bus.
Component Description	Defines an IP or interconnect structure.
Design Description	Defines the configuration of an interconnection between components.
Abstractor Description	Defines an adaptor between interfaces of two different abstractions.
Generator Chain Description	Defines the grouping and ordering of generators.
Design Configuration Description	Defines configuration information.

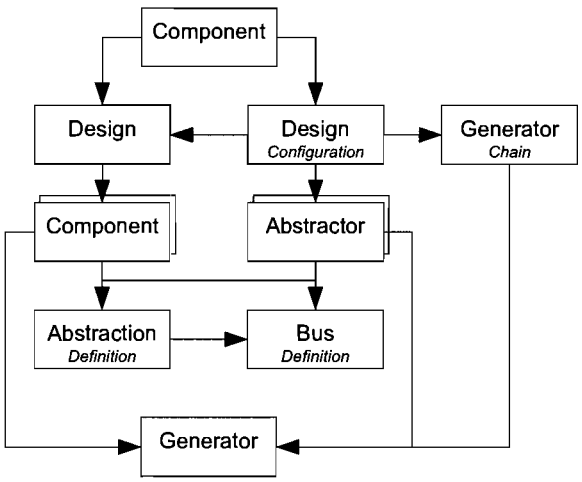


FIGURE 1.8: IP-XACT object descriptions

1.3.1 IP-XACT metadata format

IP-XACT metadata format specification is a metadata description for documenting IPs. The metadata format is an XML schema which creates a common and language-neutral way to describe IPs compatible with automated integration techniques and enabling integrators to use IPs from multiple sources with IP-XACT enabled tools. IP-XACT enabled tools are able to interpret, configure, integrate and manipulate IP blocks that comply with the proposed IP metadata description. The current version is 1.4. The XML schema which defines the metadata format is composed of seven top-level schema definitions. Each schema definition can be used to create object descriptions of the corresponding type. Figure 1.7 gives an overview of the main concepts defined with the IP-XACT metadata format.

```
<?xml version="1.0" encoding="UTF-8" ?>
<spirit:busDefinition xmlns:spirit=
  http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:schemaLocation="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4
  http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.4/index.xsd">
    <spirit:vendor>amba.com</spirit:vendor>
    <spirit:library>AMBA</spirit:library>
    <spirit:name>AHB</spirit:name>
    <spirit:version>v1.0</spirit:version>
    <spirit:directConnection>false</spirit:directConnection>
    <spirit:isAddressable>true</spirit:isAddressable>
    <spirit:extends spirit:vendor="amba.com" spirit:library="AMBA"
      spirit:name="AHBlite" spirit:name="v1.0" />
    <spirit:maxMasters>16</spirit:maxMasters>
    <spirit:maxSlaves>16</spirit:maxSlaves>
    <spirit:systemGroupNames>
    <spirit:systemGroupName>ahb_clk</spirit:systemGroupName>
    <spirit:systemGroupName>ahb_reset</spirit:systemGroupName>
    </spirit:systemGroupNames>
</spirit:busDefinition>
```

FIGURE 1.9: AHB bus definition

The links between he main schema objects are illustrated in Figure 1.8. The arrows illustrate a reference of one object to another (e.g., reference of object B from object A). Figure 1.9 is an example of the AHB portion of the AMBA specification [1] described using the IP-XACT metadata format.

1.3.2 TGI

The second interface of the SPIRIT Design Environment Architecture is one which defines the interaction API between the DE and Generators. This interface is defined by the TGI portion of the IP-XACT specification. Generators are an important part of the design environment architecture. They are executable objects (e.g., scripts or binary programs) which may be integrated within a design environment (referred to as internal) or provided separately as an executable (referred to as external). Generators may be provided as part of an IP package (e.g., for configurable IP, such as a bus-matrix generator) or as a way of wrapping point tools for interaction with a design environment (e.g., an external design netlister, external design checker, etc.). An internal generator may perform a wide variety of tasks and may access IP-XACT compliant metadata by any method a DE supports. IP-XACT does not describe these protocols.

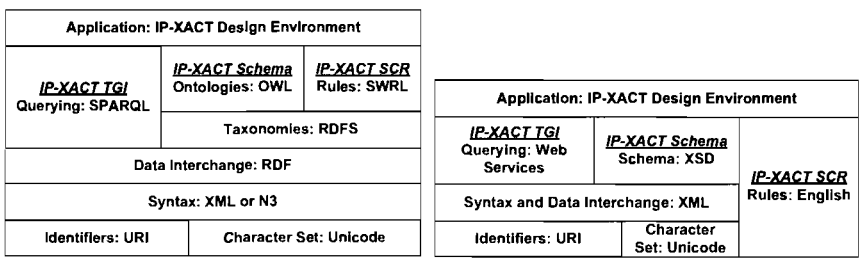
The DE and the generator communicate with each other by sending messages utilizing the Simple Object Access Protocol (SOAP) standard specified in the Web Services Description Language (WSDL). SOAP provides a simple means for sending XML format messages using the Hyper Text Transfer Protocol (HTTP) or other transport protocols.

1.3.3 SCRs

Since the schema of the IP-XACT metadata format is defined using the XML schema technology, it is bound by the expressive limits of this technology. There are a certain number of consistency rules that are important for the coherence of the metadata schema and conforming documents which can not be expressed with XML schemas. In order to define these rules, the IP-XACT specification defines a list consistency rules called the Semantic Consistency Rules (SCR) which complements the IP-XACT metadata schema. The intent is that tools implement these rules in order to validate the coherency of documents which use the IP-XACT metadata schema.

1.4 The Semantic Web

The core technology for knowledge representation in the Semantic Web is RDF. Figure 1.10a illustrates the position of RDF and all the other Semantic Web technologies relative to XML technologies. The Semantic Web is often seen as a layer above XML but as Figure 1.10a illustrates, it can be layered on other standards such as N3 notation [6]; hence it is **independent** of XML. Figure 1.10a also illustrates how the various IP-XACT standards can be implemented with the SW and positions a design environment as an application which uses the stack. Figure 1.10b serves to illustrate how the SW approach compares to the current XML approach of implementing the IP-XACT standards.



(a) SW based stack (b) Current XML based stack

FIGURE 1.10: Semantic Web and IP-XACT stack

1.4.1 RDF

RDF [2] perceives the world as a collection of resources. A resource can be anything (a web page, a fragment of a web page, a person, an object, etc.) and is referred to in the Semantic Web with a Uniform Resource Identifier (URI). RDF is built on

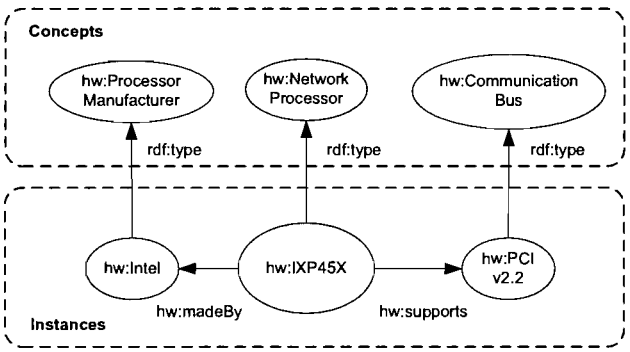


FIGURE 1.11: RDF example

3 concepts: resources, properties (relations) and statements. As mentioned earlier, a resource can be anything which is referred to by an URI. A property (or relation) is a resource which gives information on an aspect of a resource. Since a property is a resource, all properties have a unique URI. A statement is a triple of the form <Subject, Property, Object> which puts a *Ressouce* (the subject of the statement) in relation with another *Ressouce* (the object of the statement). The property (relation) of a statement indicates the aspects which the statement is giving information about. A set of statements form an RDF graph. RDF defines a small set of standard resources. The most important is the *type* property (relation) which expresses a "is a" relation. A triple which uses the *type* relation such as <aSubject, type, anObject> generally indicates that the subject of the triple is a conceptualization and the object is an instance of the conceptualization. Figure 1.11 is a simple RDF graph which describes some common hardware conceptualizations and instances of those conceptualizations which relate to the IXP45X Intel network processor [21]. Each oval depicts an RDF resource which represents a conceptualization. The arrows depict property resources. All resources are identified with a URI (URI prefix are used for conciseness). The model states that an IXP45X is a network processor (a special kind of processor) which is made by a specific manufacturer called Intel and supports a specific communication bus called PCI v2.2

The RDF specification defines a standard serialization format called RDF/XML for its abstract syntax. As mentioned earlier, other serialization formats exist such as N3 [6]. These serialization formats are not typically consumed directly but through tools. Figure 1.12 illustrates the serialization on the example presented in Figure 1.11. In the reminder of this article we shall use N3 notation. The N3 notation expresses each triple on a single line.

```
<rdf:Description rdf:about="#IXPA45X">
  <rdf:type rdf:resource="#NetworkProcessor"/>
  <hw:madeBy rdf:resource="#Intel"/>
  <hw:supports rdf:resource="#PCIv2.2"/>
</rdf:Description>
<rdf:Description rdf:about="#Intel">
  <rdf:type rdf:resource="#ProcessorManufacturer"/>
</rdf:Description>
<rdf:Description rdf:about="#PCIv2.2">
  <rdf:type rdf:resource="#CommunicationBus"/>
</rdf:Description>
```

(a) RDF/XML serialization

```
hw:IXPA45X hw:madeBy hw:Intel.
hw:IXPA45X hw:supports hw:PCIv2.2.
hw:IXPA45X rdf:type hw:NetworkProcessor.
hw:Intel rdf:type hw:ProcessorManufacturer.
hw:PCIv2.2rdf:type hw:CommunicationBus.
```

(b) N3 serialization

FIGURE 1.12: RDF serialization

Class	Meaning	Relation	Meaning
Class	This is the class of resources that are RDF classes. rdfs:Class is an instance of rdfs:Class	Range	The property rdfs:range is an instance of rdf:Property that is used to state that the values of a property are instances of one or more classes.
Property	rdf:Property is the class of RDF properties. rdf:Property is an instance of rdfs:Class.	Domain	The property rdfs:domain is an instance of rdf:Property that is used to state that any resource that has a given property is an instance of one or more classes.
		SubClassOf	The property rdfs:subClassOf is an instance of rdf:Property that is used to state that all the instances of one class are instances of another.
		SubPropertyOf	The property rdfs:subPropertyOf is an instance of rdf:Property that is used to state that all resources related by one property are also related by another.

FIGURE 1.13: Main RDFS concepts

1.4.2 RDF Schema

RDF Schema (RDFS) [2] extends RDF along 2 axes: it defines a precise list of resources (meaning and URIs) and a set of entailment rules which allow the inference of new triples from RDFS graphs. RDFS allows the definition of classes of resources as well as their organization. The concept of a class in RDFS must be understood as a set. Figure 1.13 summaries the main concepts found in RDFS.

Figure 1.14 is an extension of Figure 1.11 which defines relations between conceptualizations. For example, Figure 1.14 expresses that the set of all network processors is a sub-set of the set processors. We often refer to networks of conceptualizations as schema hence the name RDF Schema. Based on the semantic of RDFS and underlying entailment rules, 2 implicit triples should be understood : <hw:Intel, rdf:type, hw:Manufacturer> and <hw:IXP45X, rdf:type, hw:Processor>.

The previous implicit triples may be inferred because of the semantic of the rdfs:subClassOf property. Since the rdfs:subClassOf expresses the relation of parent set and sub-set between 2 sets, all individuals in the sub-set are necessarily individuals of the parent set.

1.4.3 Web Ontology Language (OWL)

OWL [2] is a knowledge representation language which can be used to represent the terminological knowledge of a domain in a structured and formally well-understood way. More specifically, OWL is a description logic language. Description logics

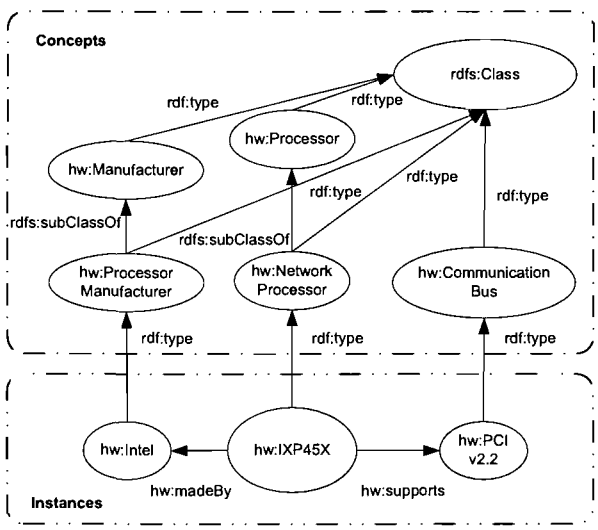


FIGURE 1.14: RDFS example

express conceptual descriptions with first-order predicate logic. OWL is defined on top of RDFS in the same way RDFS extends RDF. OWL defines a list of resources (meaning and URIs) and a set of entailment rules which allow the inference of knowledge in the form of new triples. In particular, OWL adds to RDFS the capability to express the admissibility criteria of a given class (set). It is possible to define a class not only by defining its relationship with other classes but also by defining the criteria which a resource must respect to be classified as an individual of the class. There are many semantically rich elements in the OWL specification but for the context of this paper we will focus on the concept of a *Restriction*. The concept of a *Restriction* defines a resource of type owl:Class. This class defines the set of all individuals which express restriction (criteria) specifications with are used to define other owl:Class. These criteria are typically on the existence of relations (property statement) which an individual resource must be part of. Figure 1.15 defines in greater detail the conceptualization of a Processor and a Network Processor. The example defines the class Processor as all resources which are part of exactly 1 relation madeBy as well at least 1 relation supports. The class Network is defined to be a subset of all Processor which are part of exactly 1 relation madeFor and the object of this relation must be "networking". The example retakes the IXPA45X resource of the previous examples but does not express anything about it association with a class. Based on the semantic of OWL, reasoning over the example would conclude that the resource hw:IXPA45X is a Processor and a Network Processor because it fulfils all the criteria for both sets.

The OWL specification defines 3 sub-sets of the language which extent each oth-

```
hw:Processor rdf:type owl:Class.
hw:Processor rdfs:subClassOf _:processorRestriction1.
_:processorRestriction1 rdf:type owl:Restriction.
_:processorRestriction1 owl:onProperty  hw:madeBy.
_:processorRestriction1 owl:Cardinality "1"^^xsd:int.

hw:Processor rdfs:subClassOf _:processorRestriction2.
_:processorRestriction2 rdf:type owl:Restriction.
_:processorRestriction2 owl:onProperty  hw:supports.
_:processorRestriction2 owl:min "1"^^xsd:int.

hw:NetworkProcessor rdf:type owl:Class.
hw:NetworkProcessor rdfs:subClassOf hw:Processor
hw:NetworkProcessor rdfs:subClassOf _:processorRestriction3.
_:processorRestriction3 rdf:type owl:Restriction.
_:processorRestriction3 owl:onProperty  hw:madeFor.
_:processorRestriction3 owl:hasValue "networking"^^xsd:string.
_:processorRestriction3 owl:Cardinality "1"^^xsd:int.

hw:IXPA45X hw:madeBy hw:Intel.
hw:IXPA45X hw:madeFor "networking"^^xsd:string.
hw:IXPA45X hw:supports hw:PCIv2.2.
```

FIGURE 1.15: OWL example in N3

ers: OWL Lite, OWL DL and OWL Full. Each subset is balanced between expressivity and the computational complexity to reason over a model defined with the subset. Since OWL builds upon RDFS and RDF, it uses the same serialization formats.

1.4.4 SPARQL

SPARQL [28] is a query language for RDF graphs. SPARQL is very much to RDF what SQL [14] is to relational databases. SPARQL is based on a pattern matching paradigm like XPath [5]. In the same way that an XPath describes an XML pattern which is usually hierarchical, a SPARQL query describes a graph pattern. A basic SPARQL query has 2 portions: a **SELECT** portion that defines a list of variables returned by the query and a **WHERE** portion that defines a list of triple statements used to matched. The variables in the **SELECT** portion are used as unbound elements (subject, property or object) in the statements. The query defined in Figure 1.16 searches for all NetworkProcessor which support PCIv2.2. If executed on the example illustrated in Figure 1.14, the result set would contain hw:IXP45X.

If the query had requested all Processors which support PCIv2.2, the result set would have been empty because there is no explicit relation (rdf:type) between hw:IXP45X and Processor. A SPARQL engine will only search for matches based on what is explicitly present in the queried graph. As discussed previously, the OWL language has precise semantics which includes entailment rules. By processing an OWL model with an inference engine such as Pellet [16] , new statements can be added to the

Select Query

SELECT ?x
WHERE{
?x rdf:type hw:NetworkProcessor.
?x hw:supports hw:PCIv2.2.
}

hw:IXP45X

FIGURE 1.16: Select query

Construct Query	Ask Query

CONSTRUCT{
?x rdf type ?c1.
}WHERE{
?c2 rdfs:subClassOf ?c1
?x rdf:type ?c2.
}
	ASK {
	hw:IXP45X hw:madeBy
	hw:Intel.
	}

hw:IXP45X rdf·type hw:Processor.	yes

FIGURE 1.17: Construct and Ask queries

model based on the entailment rules. If we apply entailment rules to the OWL example, the RDF statement <hw:IXP45X, rdf:type, hw:Processor> will be added. The execution of the query would now return the expected result. This capability to infer new knowledge from a given model is a great added value over typical approaches.

The SPARQL specification also defines three other query types :

1. The **CONSTRUCT** query form returns a single RDF graph specified by a graph template. The result is an RDF graph formed by taking each query solution in the solution sequence, substituting for the variables in the graph template, and combining the triples into a single RDF graph by set union.

2. The **ASK** query is used to test whether or not a query pattern has a solution. No information is returned about the possible query solutions, just whether or not a solution exists.

3. The **DESCRIBE** form returns a single result RDF graph containing RDF data about resources. This data is not prescribed by a SPARQL query, where the query client would need to know the structure of the RDF in the data source, but, instead, is determined by the SPARQL query processor. The query pattern is used to create a result set.

Figure 1.17 give examples of the **CONSTRUCT** and an **ASK** query as well as the results of the queries if execute on Figure 1.14. In conjunction to the SPARQL specification, the SPARQL Protocol specification [13] was established in order to define a communication interface over HTTP for remote SPARLQ query execution.

In addition to the operators defined by the official SPARQL standard, others have been proposed and implemented by tools. A specification called SPARQL/Update [28] has been propose to the W3C for standardization. This specification defines two operators which enable to insert and delete triples, hence given write, update and delete

Update Query

```
DELETE{
  ?x hw:madeBy hw:Intel.
}INSERT{
  ?x hw:madeBy hw:AMD.
}WHERE{
  ?x hw:madeBy hw:Intel.
  ?x rdf:type hw:NetworkProcessor.
}
```

FIGURE 1.18: Update query

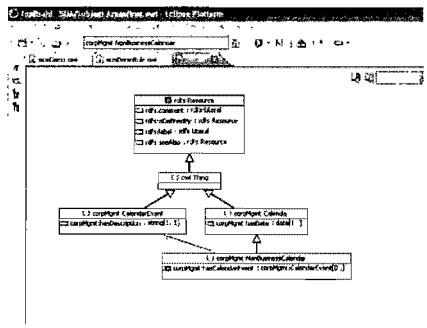


FIGURE 1.19: TopBraid Composer

capabilities. Figure 1.18 illustrates the use of the SPARQL/Update extension in order to change the madeBy poperty of all NetworkProcessor from "Intel" to "AMD".

1.4.5 Tool for the Semantic Web : Editors and Jena

The Jena framework [11] is a Java-based open source toolkit for the Semantic Web. The current version implements a programmatic framework for RDF, RDFS, OWL and SPARQL. Jena also provides some interesting features such as a rule-based inference engine and a persistence storage framework for large RDF graph. Jena also provides some very powerful extensions to the SPARQL languages such as free text searches and property functions. In the context of this paper, the most important extension is the SPARQL/Update specification.

Web Semantic development is usually done using an editor. Multiple commercial and academic editors are available such as as Protege [23] and TopBraid Composer (see http://www.topbraidcomposer.com/). These editors, in addition to facilitating model edition, they commonly support visualization of semantic models, integration with inference engines, SPARQL integration and various kinds of analysis.

```
rdfs:subClassOf(?x,?y)rdfs:subClassOf(?y,?z)->rdfs:subClassOf(?x,?z)
```

FIGURE 1.20: Rule example

1.4.6 SWRL and Jena rules

The Semantic Web Rule Language (SWRL) [20] is a W3C member submitted standard since 2004. It is based on a combination of the OWL DL and the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language. SWRL extends OWL with Horn-like rules [19]. As such, SWRL defines a high-level abstract syntax for the definition of rules as well has a formal semantic definition for the interpretation of these rules in the context of an OWL ontology. SWRL rules are in take the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as:

"Whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold."

Both the antecedent (body) and consequent (head) consist of zero or more atoms. An empty antecedent is treated as trivially true (i.e. satisfied by every interpretation), so the consequent must also be satisfied by every interpretation; an empty consequent is treated as trivially false (i.e., not satisfied by any interpretation), so the antecedent must also not be satisfied by any interpretation. Multiple atoms are treated as a conjunction. Atoms in these rules can be of the form **C(x)**, **P(x,y)**, **sameAs(x,y)**, **differentFrom(x,y)** or *built-in*(x,y,z,...) where **C** is an OWL description, **P** is an OWL property, and **x,y** are either variables, OWL individuals or OWL data values. The specification proposes a library of functions which reuses the existing built-ins in XQuery [8] and XPath. The list of built-ins may be extended by users.

Figure 1.20 illustrates the use of SWRL to define an entailment rule for the rdfs:subClassOf property of RDFS. The rule state that if x is a subset of y and that y is a subset of z then x is also a subset of z. Figure 1.21 illustrates the concrete syntaxes of the example of Figure 1.20. The Jena framework also includes a general purpose rule-based reasoner which is used to implement both its RDFS and OWL reasoners, however it is also available for general use. This reasoner supports rule-based inference over RDF graphs and provides forward chaining, backward chaining and a hybrid execution model. The syntax of the rules is very similar to the abstract syntax of SWRL. Jena also defines a similar built-in library such as the one defined by SWRL. Tool such as TopBraid Composer support both the edition of SWRL and Jena rules. Some inference engines such as Pellet support the execution of SWRL rules. When using Top Braid composer, the editor translate SWRL rules to Jena rules and then uses the Jena reasoner; this approach means that the editor only support the sub-set of SWRL which maps to Jena rules.

XML syntax

```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#example1"/>
    <ruleml:_body>
      <swrlx:individualPropertyAtom
        swrlx:property="hasParent">
        <ruleml:var>x1</ruleml:var>
        <ruleml:var>x2</ruleml:var>
      </swrlx:individualPropertyAtom>
      <swrlx:individualPropertyAtom
        swrlx:property="hasBrother">
        <ruleml:var>x2</ruleml:var>
        <ruleml:var>x3</ruleml:var>
      </swrlx:individualPropertyAtom>
    </ruleml:_body>
    <ruleml:_head>
      <swrlx:individualPropertyAtom
        swrlx:property="hasUncle">
        <ruleml:var>x1</ruleml:var>
        <ruleml:var>x3</ruleml:var>
      </swrlx:individualPropertyAtom>
    </ruleml:_head>
</ruleml:imp>
```

RDF/XML syntax

```
<swrl:Variable rdf:ID="x1"/>
<swrl:Variable rdf:ID="x2"/>
<swrl:Variable rdf:ID="x3"/>
<ruleml:Imp>
  <ruleml:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate
        rdf:resource="&eg;hasParent"/>
      <swrl:argument1 rdf:resource="#x1"/>
      <swrl:argument2 rdf:resource="#x2"/>
    </swrl:IndividualPropertyAtom>
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate
        rdf:resource="&eg;hasBorther"/>
      <swrl:argument1 rdf:resource="#x2"/>
      <swrl:argument2 rdf:resource="#x3"/>
    </swrl:IndividualPropertyAtom>
  </ruleml:body>
  <ruleml:head rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
      <swrl:propertyPredicate
        rdf:resource="&eg;hasUncle"/>
      <swrl:argument1 rdf:resource="#x1"/>
      <swrl:argument2 rdf:resource="#x3"/>
    </swrl:IndividualPropertyAtom>
  </ruleml:head>
</ruleml:Imp>
```

FIGURE 1.21: SWRL serializarion examples

1.5 XML and its shortcomings

Pre-XML data exchange was characterized by a vast amount of proprietary file formats; most of which were either binary or flat (comma-delimited, tab-delimited, etc.). Consumption of these files came at a high cost, each software system had to implement a parser and interpreter for each file format; very little reuse was possible because of the diversity of data encoding and data structuring. Hence, enabling M software systems to exchange information bi-directionality with one another required M*(M-1) parser/interpreter bridges. This high number of software data exchange bridges made data exchange and data interoperability between systems a fairly challenging and expenses endeavor.

The advent of the XML technology stack (XML, XSD, XPatn, XSLT and XQuery) [15] has democratized the exchange and consumption of data because XML-based data exchanges uses a predefined data encoding scheme (Unicode) as well as a meta-structure for syntax. This has enabled the development of generic file parsers which tools can embed and reuse. Moreover, since all the technologies in the XML stack are based on open standards, many free implementations are available which has greatly lowered the cost of data exchange and interoperability.

Current data exchanges based on XML only require parties to define a precise XML data model which is defined using the XML schema standard. Once a data model has been defined, software systems only have to implement an interpreter which extracts data fragments (using XPath or XQuery) and consume them in a

Example A	Example B
<networkProcessor> <name>IPX45X</name> <supports> <communicationBus> <name>PCIv2.2</name> </communicationBus> </supports> </networkProcessor>	<networkProcessor name="IPX45X"> <supports> <communicationBus name="PCIv2.2"/> </Supports> </networkProcessor>

FIGURE 1.22: Multiple possible syntaxes

fashion which is coherent with the meaning or intent of the fragments. Even though XML stack has solved a vast number of problems with regards to data exchange, it has some very important shortcomings.

1.5.1 Multiple Grammars

XML allows multiple valid syntaxes for a particular semantic model. An XML data model consists of two aspects one is syntax (syntax model) and the other is semantic (semantic model). To achieve a consensus around a data model requires a consensus on both models as well as the mapping between them. To be more precise, when defining an XML based format, it is necessary to define the schema that will define the structure which all XML files based on the schema must respect, the *syntax model*. The *syntax model*consists of the element definitions, the attribute definitions and the nesting rules for elements just to name a few. The consensus with regards to semantics is the definition of "what" the data contained in the file structure means independently of how it is expressed in the file. This consensus is important because even if a data structure is clearly defined and accessing the data in the structure is simple, if multiple parties interpret the meaning (semantic) of the information differently, data interoperability has not been achieved. For example, if a data model defines an element called POWER which contains an integer value; it is possible that one parties interprets the value in KiloWatts and other party in MegaWatts; this is called a semantic gap.

The problem with having multiple syntax models is that it complicates achieving a consensus on the syntax model for multiple schools of thought exist which advocate different styles. Also as the need for data exchange evolves, the structuring of the information will change for maintenance raisons in order to facilitate the integration of new data in the exchange. This causes incompatibilities between syntax models which requires software systems to be modified even if the semantic models are fully compatible because new data concepts are added or refined which does not invalidate earlier interpretation. For example, between SPIRIT 1.2 and 1.4 some attributes have become elements. We must consume meaning and not encoding and syntax. Figure 1.22 is an example of multiple syntaxes for the same meaning if the

statement "IPX45X, a network processor, supports PCIv2.2, a communication bus."

1.5.2 Documentation-centric

By design, XML is intended for simple message-oriented data exchanges, it is document-centric. Hence the capabilities of storing data in multiple files – files which have no knowledge of one another but which the data they contain is complementary - and then easily combine these files dynamically in order to query the consolidated data set as a virtual file is not possible. For example, it is difficult to have an XML document which defines the structure of a system which is composed of modules, have multiple documents which contain detailed information of each module and then combine all of the documents and query across them.

Another problem with the documentation-oriented nature of the XML is that if one wishes to use XPath, XQuery, or XSLT it is necessary to know the physical location of a file in order to load it in memory and then manipulate it. Hence software systems which consume XML documentations must have inherent knowledge of the physical file names and locations in order to consume their contents. This makes XML consumption brittle from a physical data exchange point of view. For example, just changing the name of a file or its location can easily break data exchanges. Software system are tied to the physical location of files even though it is the consumption of their content semantics which is important.

Typically, both the consolidation and the location transparency problems are avoided by using a consolidation repository which stores all the information in the XML documents. This repository is usually based on relational database systems because of the omnipresence of the technology and supporting tools. This is a very effective way to consolidate data and to offer a mostly transparent access to the information (one must still know the connection string of the database) but it bring a new problem : it is necessary to define a new data model – a relational data model- which bring the same two consensus aspects we described with XML data models. Moreover, a new technologies stack must be learned and exploited. SPIRIT uses this consolidation repository approach.

Again, in order to avoid this new problem, a typical solution is to hide the data store behind a Web Services layer which exposes an XML data model. By reusing the initial data model, no new consensus must be achieved, only the communication and the maintenance of the APIs which is not necessarily an ease task. Moreover, hiding the consolidated documents behind an API typically narrows greatly the consumption capabilities of a software because APIs usually only exposes specific consolidated fragments so querying can only be done on those fragment and not on everything in the data store. SPIRIT uses this approach for the TGI.

1.5.3 Biased Grammar model

Database technologies have evolved from tree-based data models (hierarchically database) to network-based models (network databases) to relation data models over the last 30 years; the XML data model brings us back to the beginning.

The meta-model of XML data models is biased toward tree-like structures; the nesting of elements is the principal mechanism which allow syntax model definitions. Using graph structures such as in UML [9] class models and table structures in relation models is a more natural ways of modeling the world. Since an XML data model requires a single top level root, it is necessary to either promote a concept in the data model to the top element or create an artificial one which has no semantic meaning and is present for only syntactic reasons.

Through the use the ID/IDREF from XSD 1.0 or key/keyref pairs from XSD 2.0 it is possible to create implicit graph-like structure attributing identifiers to nodes in an XML document and allowing nodes to refer other nodes by using their IDs. However, these are implicit graphs and querying through implicit graph is not trivial. With XPath and XQuery, it is not possible to request the XML node witch is referenced by another node, it is necessary to explicitly search for the referred node by using query predicates on the keys which identify the node.

1.5.4 Limited Metadata

XML offer more semantics than flat-file because metadata is present in the form of element name and attribute name, however semantic expressiveness is limited. For example the nesting of an element in another elements has no semantics, e.i. if element **a** has a nested element **b**, does it mean that **a** poses **b** or that **b** belongs to **a** etc.

File formats such as comma-delimited posses no inherent metadata, only the data is present. Hence, the consumption of these files requires an intrinsic knowledge of the content of the file and its meaning. XML documents, by the presence of element and attribute names, deliver metadata as well as data. Moreover, the scoping of elements and attributes with namespaces makes them unique because there are based on URIs. A consensus on an XML data models implies that a consensus of the mapping of the semantic model to the syntax model as been achieved, which implies that a consensus on each XML elements – which is unique because of the URIs scheme- and its meaning as been reach. This allows a software system to consume a previously unknown XML file because it may search the file for elements and attributes with specific URIs which the software systems knows how to correctly interpret. Having said this, XML does not convey all the necessary metadata which is implied by an XML data model. The mechanism of nesting has no semantics; an XML element may be nested under another XML element in order to represent very different meanings. For example, given the concept of a hardware module which is represented as an XML element and the fact that we wish to express two lists of modules which are related to a specific module:

1. the list of modules which this module is backward compatible with
2. the list of modules which the module is not backward compatible with

We cannot simply list each module in both of these lists under the module of discussion because it will not be possible to distinguish the modules in the compatible

list from the incompatible list. This problem is typically solved by either creating two XML elements that respectively represent both of these lists under which we nest the appropriate module entries. We then nest these two XML elements under the module under discussion. In this solution nesting still does not have any semantics but we have created an unambiguous data structure which can be interpreted by a software system.

UML class models and ER data models do not have this issue, for in both, associations between either classes or entities are named. Moreover, in a UML classes it is possible to define role for each class which participates in an association. Hence, in UML class models and ER models, the semantics of associations are precise and clear. If we wished to represent the above problem in a UML class diagram we would have only one class definition and two recursive associations, one called is compatible whit and the other is incompatible with. The cardinality of both associations would probably be many to many

1.6 Advantages of the Semantic Web

The advent of XML technologies enabled a simpler data manipulation paradigm than the one supported by flat file approaches. It also brought data manipulation at a higher-level of abstraction. Data manipulation was no longer focused on encoding and parsing but on grammar and data exchange. The Semantic technologies do the same with respect to the XML technologies. These new technologies bring many important benefits over its predecessor with regards to data manipulation and also bring the bring data manipulation at a higher-level of abstraction. The focus now is on semantics, data integration and queries over consolidated information sources. Figure 1.23 is a summary of this section.

1.6.1 Richer Semantic Expressivity

OWL offers a rich, formal and non-ambiguous language to describe information and knowledge. In many regards, its expressive capabilities go well beyond that of XML. By defining information using OWL, it is possible to infer additional information, hence creating knowledge; this is not possible with XML. This capability is enhanced when combining OWL with SWRL rules. Another distinction advantage that OWL, RDFS and RDF has over XML is that the schemas are also information to which inferencing may be applied.

As discussed earlier, most modern data model schemas such as UML and Relation schemas are inherently graph-oriented; XML is tree-oriented which can often be an inconvenience. By the very nature of the triple basis of the RDF, semantic data models are directed graphs. In addition, the meaning of a model which used semantic technologies is unambiguous. Each element of data that is in a relation with another

XML	OWL
1. Syntax and grammar focused	1. Semantic focused which abstracts syntax
2. Informal Semantics	2. Formal Semantics
3. Document and centralized oriented	3. Distributed oriented
4. Supports federation with but not transparently	4. Supports federation by design and transparently
5. Hierarchical data model with implicit graph structures	5. Explicit Graph data model
6. Multiple grammar for a specific semantic model	6. Single grammar for a specific semantic model
7. Support syntax transformation	7. Support semantic integration
8. Doesn't support entailment	8. Supports entailment

FIGURE 1.23: IP-XACT object description types

element of data is done so with a predicate; hence the meaning of the relation is unambiguous. RDF based models do not suffer from the semantic gaps which XML does such as element imbrications which defines ambiguous relations between data element.

1.6.2 Separation between semantics and encoding

The Semantic Web stack is focused on the modeling/expression of semantics and not encoding/data structure. The serialization of OWL to an RDF/XML encoding and grammar is defined in the OWL specification, hence only one grammar model exists of a specific semantic model for that particular encoding/structuring scheme. Even though multiple encoding/structuring schemes exist for OWL, consumption is always done at the semantic level through tools such as SPARQL and Jena. Since the consumption of OWL at the semantic level and Jena, software applications do not know which file format was used to express the OWL model, hence are not affected by encoding changes or file location etc. Moreover, software applications are also isolated from changes to the semantic models which are backward compatible. As long as semantic definitions are not removed or the meaning (semantic element should never change meaning), software application will not break.

1.6.3 Federated data model

The semantic technology stack is based on the premise that "anybody can say anything about anything" on the web which implies that data is scattered throughout the Web. Because of this premise, technologies of the semantic stack have been developed in order to federate this information and allow its consumption in a manner which is agnostic of this distribution. Through federation, consumers have the perception that all the information is local and storage in a single "virutal file". Tools such as Jena and SPARQL engines are design to stitch RDF statements from mul-

tiple sources together in order to achieve a global picture, a single graph structure. This focus on federation is at the opposite of XML which is mostly focus data exchanges patterns in the context of peer to peer communication were distribution is not a concern. Hence, semantic technologies have a distinct advantage in scenarios which require data consolidation (file consolidation) even when the consolidation is not on the Web. The newest generation of XML technologies (XQuery 1.0, XPath 2.0 and XSLT 2.0) has the capability to manipulate data from multiple files, hence offering a certain level of federation capabilities. However, the physical local of files has not been abstracted and the consolidated view is achieved by the user defining the necessary joins between data elements. With the Semantic Web technologies, file location is not important because it is managed by SPARQL engines which also stitches data element base on semantics without human intervention.

1.6.4 Simpler Data Manipulation

Data manipulation can often be reduced to three simple areas of focus: data queries, data updating and data transformation. In the context of the XML technologies, the XQuery and XSLT languages support these areas. Even though they are simpler than previous technologies, these technologies still have various aspects which are overly complicated.

XQuery offers querying capabilities over XML documents. It has a syntax which is similar to combination of SQL and XPath. It is not a very complicated language but because of the underlying data model which is XML, queries must always take in consideration the tree-oriented structure of the data which is not as simple as a graph structure. Moreover, has discussed earlier, it is difficult to navigate in the implicit graph structure of an XML document. Both of these aspects are simpler with the Semantic Web technologies because all nodes have an explicit identifier which is universally unique and which is independent of the relation in which the node is part of. It is this node which is used to define the explicit graph structure of the data model. Because of this explicit graph structure based on simple identifiers it is simple to navigate along relations by means of the name of the relation and the identifier of the starting node.

An extension for the XQuery language has been proposed in order to manage update operations. Prior to this, updates could only be achieved by either using transformations to generate a new document from an older one with the updates or to use a coding library which supported DOM which is a standard API for XML document manipulation. Both of these older approaches were overly complicated for a simple attribute value update. The new XQuery extension offers a number of update operators which allows the insertion, modification, deleting of XML content. This approach is much simpler than the older approaches but it still exposes users to the tree structure requirements of the manipulation such as:

1. under which element must another element be added
2. if other element are already present between which elements should the inserting be done.

With the semantic technology, additions are as simple as adding them to the "cloud" of existing statements because each statement is independent of the others. Removal or modification of statements is just a manner of finding the statements which must be manipulated and applying the manipulation, all the operations are done at the semantic level.

With regards to the area of transformation, transformations which are focused on the massaging of various data sources into a canonical format is where the Semantic web is at its best. Because semantic data models use a formal model to define their structure and their means, it is possible to uses declarative axioms to define the semantic equivalence between various sources. Examples of such axioms are:

1. Equivalence class axiom;
2. SubClassOf axiom;
3. Equivent property axiom;
4. SubPropertyOf axiom.

These types of transformations are referred in the semantic web as data or semantic integrations. [2] discusses in great details the capabilities of the semantic web to fulfill this task. The execution of the declarative statements is achieved with an inference engine. Using SWRL rules can also augment the expressive capabilities in order to define transformations. The Semantic Web is only focused on semantics; hence it is only interested in transformation with regards to semantics in order to either:

1. do semantic alignment (semantic integration);
2. define rules which will help deduce new knowledge in a semantic model from knowledge in another semantic model.

The Semantic Web does not address transforming information in a certain encoding format into another encoding format. This capability is XSLT's strong point. This is to be expected because the XML technologies were designed to facilitate manipulations at the syntax and encoding level. However, this same focus on syntax and encoding makes semantic data integration more complex because it must be achieved at a lower level of abstraction. Because this article is mainly concerned with semantics, the advantage is given to the semantic technologies.

1.7 Case Study – SPIRIT

SPIRIT, as discussed earlier, uses XML to represent and share information. IP vendors use the IP-XACT metadata format for the definition of metadata which describes

Change Type	Examples	Impacted Files
Attribute removal	The spirit:choiceStyle and spirit:direction attributes have been removed from all element configurable elements	autoConfigure.xsd.
Element removal	The spirit:maxMasters and spirit:maxSlaves elements have been removed from the spirit:channel element	busInterface.xsd
Splitting of concepts	The busDefinition was slipt in two concepts: busDefinition and abstractDefinition	busDefinition.xsd
Cardinality Changes	The number of spirit:busInterface elements under spirit:busInterfaces has changed from 1..n to 0..n.	busInterface.xsd
Type changing	The type serviceTypeDef has changed from xs:Name to xs:string .	port.xsd

FIGURE 1.24: Breaking semantic changes summary

their IPs. The SPIRIT development environment uses the IP-XACT metadata format schema as a MoA formalism in order to define system-model designs based on IP aggregation. If the SPIRIT consortium was to based the IP-XACT specification on the Semantic Web technologies, IP-XACT would benefit in a number of ways. This section will discuss these advantages.

1.7.1 Advantages applied to version management (SPIRIT 1.2 to SPIRIT 1.4)

Like most specifications and standards, the IP-XACT specification will be rectified over time. This evolution process generally will come at the cost of incompatibilities between version increments. On many occasions these incompatibilities will be unavoidable because changes are made to the semantics of the specification. These changes will require consuming tools to be modified in order to interpret the new version correctly. For example, changes to the IP-XACT metadata format have been made between versions 1.2 and 1.4. Many of these changes are semantic in nature hence modification to the semantics and syntax of the metadata format have been made. Figure 1.24 summaries semantic breaking changes between versions 1.2 and 1.4 of IP-XACT. These changes cause incompatibilities which are unavoidable and this independently of the specification technology which is used.

On the other hand, figure 1.25 summaries changes between both versions which do not break semantics but only syntax; if semantic level data exchange was used this would have been avoided.

1.7.2 Advantages applied to modeling

The semantic modeling technologies such as RDF, RDFS and OWL have an advantage over XML when modeling because many encoding details which have no semantic significance are abstracted. This abstraction of encoding details allows for a simpler modeling experience.

Change Type	Examples	Impacted Files
Attribute Renaming	The spirit:signalName is renamed to spirit:portMap inside spirit:signalMap	busInterface.xsd
Element Renaming	The spirit:remapSignal element has been renamed to spirit:remapPort	busInterface.xsd
Changes from attribute to element	The spirit:name attribut of the spirit:adHocConnection element has become a sub-element.	subInstances.xsd
New collections tags	A container element called spirit:parameters has be created in order to organize multiple following spirit:parameter elements.	Global

FIGURE 1.25: Non-breaking semantic changes summary

A common practice in XML is to use container style elements in order to organize file content. An example of this technique is the uses of parameters elements which contain parameter elements in the IP_XACT specification. These container tags add no semantic meaning, they only facilitate human readability. From a modeling and computational processing perspective, these tags only add "noise" to the model.

A similar subject of great debate which is most often stylistic in nature is the use of element vs attribute in order to encode data. As discussed earlier, XML support the use of elements or attributes for the encoding of properties. When an entity can be associated with multiple values for a same property it is often necessary to use elements, because only elements may be repeated. In mostly all other situations, from a modeling perspective, there is no semantic difference between both approaches. Hence, this again just complexifies the modeling process and leavers room for unnecessary debates.

Another area where XML has added complexity is then management of element cardinality when using nesting. XML schema offers two options for defining nesting rules: **sequence** and **all**. The **sequence** option allows nesting an unlimited sequence of elements (order is important), the number of times an element may be present can be specified using a minimum and maximum cardinality constraints. The **all** option allows nesting of an unlimited set of elements (order is not important) but each element can appear at most once. There is no option which allows the nesting of a set of elements (order not important) and that allows the specification of occurrence using minimum and maximum constraints. As a consequence, XML schemas when using the **sequence** option in order to manage cardinality without constraints becomes sensitive to element reordering and addition. This added complexity is only because of encoding concerns and not for semantic concerns. Since the IP-XACT specification uses the **sequence** it is overly sensitive to element adding and re-ordering both of which do not break semantic compatibility but which break grammar compatibility.

The concept of uniqueness is at the core of the Semantic Web technologies and this across files. RDF has implemented this requirement with URIs, a very simple but effective construct. Since XML is document oriented, it does not have any construct to

```
spirit:AbstractDefinition rdf:type owl:Class.
spirit:Port rdf:type owl:Class.
spirit:hasPort rdf:type owl:ObjectProperty.
spirit:hasIdentifier rdf:type owl:DataProperty.
spirit:hasLogicalName rdf:type owl:DataPorperty.
spirit:isAddress rdf:type owl:DataProperty.
spirit:AbstractDefinitionInstance rdf:type AbstractDefinition.
spirit:PortInstance rdf:type :Port.
spirit:AbstractDefinitionInstance :hasPort :PortInstance.
spirit:PortInstance :hasIdentifier "myID".
spirit:PortInstance :isAddress true.
spirit:PortInstance :hasLogicalName "myName".
```

```
SELECT ?logicalName
WHERE{
  ?aPort rdf:type spirit:Port.
  ?aPort spirit:hasIdentifier "myID".
  ?aPort spirit:hasLogicalName.
  ?logicalName.
}
```

```
ASK{
  ?aPort rdf:type :Port.
  ?aPort spirit:hasIdentifier "myID".
  ?aPort spirit:isAddress true.
}
```

FIGURE 1.26: SPARQL implementation of TGI example

manage unique references across files. Moreover, even if XML schema offers capabilities to manage identifiers and references within a file, it is not common practice to uses them. The IP-XACT specification does not use the key capabilities of XML schema, it has defined it own concept called VLNVs, a 4 part identifier. Using custom identifiers shemes adds unnecessary complexity and well has added developed in order for tool to very consistency. The 4 part scheme of IP-XACT could be easily encoded within a URI.

1.7.3 Impact on TGI

The TGI portion of the SPIRIT standard exemplifies the Web Services approach to the XML location and consolidation problem. By using the Semantic Web technologies, the entire API could be eliminated for federated SPARQL queries over a collection of RDF files.

The quasi-totaly of the TGI API is composed of "getter" and "setter" operations. There are two types of "getter" operation in the API, those that return values contained in the model and those that return computed values based on values in the model. In the current version, all of the operations that return computed values are test operations which return Boolean values. The first class of "getter" operations can be substituted with simple SPARQL queries using the SELECT construct. The second class of "getter" operation can be substituted with simple SPARQL queries using the ASK construct. Respectively the getAbstractionDefPortLogicalName and getAbstractionDefPortIsAddress are examples of the two types of "getter" operations. Figure 1.26 illustrates the implementation of these two operations.

The TGI API also has a number of "setter" type operations which allow the modification of the data contained in the data store. These methods cannot be implemented with basic SPARQL queries however they could be implemented using the JENA framework library or the UPDATE SPARQL extensions which Jena supports. The

```
INSERT{
  ?aPort spirit:hasLogicalName "newName".}
DELETE{
  ?aPort spirit:hasLogicalName ?oldName.}
WHERE{
  ?aPort spirit:hasIdentifier "myID".
}
```

FIGURE 1.27: SPARQL Update implementation of TGI example

two other type of operation are the "add" and "remove" which can be implemented in the same way as the "setter" operation. Figure 1.27 is an example of the using the SPARQL/UPDATE for the setAbstractionDefPortLogicalName operation.

1.7.4 Implications for SPIRIT Semantic Constraint Rules (SCRs)

The SPIRIT 1.4 specification contains a list of SCRs which define constraints that cannot be expressed or easily expressed with XML Schemas. By using semantic technologies such as SWRL or JENA rules, SCRs which could be expressed using these technologies could be used to verify the consistency of designs. Verification would be achieved by processing the design using an inference engine which supports rules. This utilization of the semantic technologies would bring two key benefits:

1. Unless specified using controlled vocabularies, rules expresses in natural languages may be ambiguous. This ambiguity may result in different interpretations of the rules, hence different validations. By using formal rule languages such as SWRL, rules can be specified in a non ambiguous fashion.

2. Rules describe in a specification document, hence not as a constraint in a schema language such as XML schema, must be translated into a program in order to be applied to models for validation purposes. By using a formalism which is executable, this extra translation step is eliminated, thus adding value.

The reminder of this section will present a number of SCRs rules which may be expressed using OWL and/or SWRL. We will also present rules which cannot be expressed formally in order to present possible limitations. The objective of this section is not to discuss thoroughly all the SCRs but rather to demonstrate that it is possible to define formally some of the rules with OWL and SWRL which cannot be expressed with XML Schema. Even if only a fraction of the SCRs can be expressed formally this capabilities is a benefit over the current XML implementation which cannot.

The specification contains a certain number of rules which pertain to referential integrity between design elements. The main objective of these rules is to express that a design element which references another design elements must reference a valid element. The XML Schema 1.0 standard allows the definition of element attributes as identifier by using the ID type. These identifiers must be unique with the context of

```
spirit:BusInterface rdf:type owl:Class.
spirit:AbstractionDefinition rdf:type owl:Class.
spirit:BusDefinition rdf:type owl:Class.
spirit:hasBusType rdf:type owl:ObjectProperty.
spirit:BusInterface rdfs:subClassOf _:BusInterfaceRestriction1.
_:BusInterfaceRestriction1 owl:onProperty spirit:hasBusType.
_:BusInterfaceRestriction1 owl:allValuesFrom spirit: spirit:BusDefinition.
spirit: AbstractionDefinition rdfs:subClassOf
    _: AbstractionDefinitionRestriction1.
_: AbstractionDefinitionRestriction1 owl:onProperty spirit:hasBusType.
_: AbstractionDefinitionRestriction1 owl:allValuesFrom
    spirit:spirit:BusDefinition.
```

FIGURE 1.28: Implementing SCR 1.4 using OWL

a document. These identifiers may be used as values for element attributes which are declared as type IDEF. Using this feature, it is not possible to declare that a certain element may refer, by means of its ID attribute, to another element of a specific type. It is only possible to declare that an element may refer to another element by means of its ID. With OWL, all resources must have a unique ID (its URI) and resources may be associated with specific owl:Class. By means of "allValuesFrom" axioms it is possible to define precise class criteria on range constraints from relations. The SCR 1.4 is defined as follows : *The VLNV in a busType element in a bus interface or abstraction definition shall be a reference to a bus definition.*

Figure 1.28 illustrates how this rule could be implemented using OWL. The model defines that all instances which are in a relation with an **busInterface** or an **astractionDefinition** instances using the hasBusType property must be a **busDefinition**. The owl:allValuesFrom declares this constraint on the range of the hasBusType property. This example also demonstrates the substitution of VLNVs by URIs.

SCRs such has SCR 2.4-2.9 express more complex conditional constraint on values. These rules define constraints on the allowed combinations of interfaces that may be connected together using an interconnection. For example SCR 2.4 states: *An interconnection element shall only connect a master interface to a slave interface or a mirrored-master.* These rules could be express using only OWL and restriction criteria. Figure 1.29 illustrates the implementation of SCR 2.4.

SCRs which express constraint on values which are in simple equality relations can be expressed using OWL and SWRL. For example SCR 2.10 states: *In a direct master to slave connection, the value if bitsInLAU in the master's address space shall match the value of the bitsInLAU in the slave's memory space.* Figure 1.30 illustrates the implementation of SCR 2.10 using OWL and SWRL.

As stated earlier, no all rules may be expressed using OWL and/or Jena rules. An example of such a rule is SCR 3.3 which states: *A channel can be connected to no more mirrored-master busInterfaces then the least value of maxMAsters in the busDefinitions referenced by the connected busInterfaced.* The main issue is that since OWL and Jena are based on first-order logic, there is no direct way to express rules which require counting. Hence we cannot express a rule that states that the sum of the relations which an instance participates in must be lower then a value which is

```
spirit:MasterInterface rdf:type owl:Class.
spirit:SlaveInterface rdf:type owl:Class.
spirit:MirroredMasterInterface rdf:type owl:Class.
spirit:MirroredSlaveInterface rdf:type owl:Class.
spirit:MirroredSystemInterface rdf:type owl:Class.
spirit:DirectInterface rdf:type owl:Class.
spirit:MasterInterconnection rdf:type owl:Class.
spirit:hasMainInterface rdf:type owl:ObjectProperty.
spirit:hasSecondaryInterface rdf:type owl:ObjectProperty.
spirit:MasterInterconnection rdfs:subClassOf
_:MasterInterconnectionRestriction1.
_:MasterInterconnectionRestriction1 owl:onProperty spirit:hasMainInterface.
_:MasterInterconnectionRestriction1 owl:allValuesFrom
    spirit:spirit:MasterInterface.
spirit:MasterInterconnection rdfs:subClassOf
    _:MasterInterconnectionRestriction2.
_:MasterInterconnectionRestriction2 owl:onProperty
    spirit:hasSecondaryInterface.
_:MasterInterconnectionRestriction2 owl:allValuesFrom _:Union1.
_:Union1 owl:unionOf (spirit:SlaveInterface spirit:MirroredMasterInterface).
```

FIGURE 1.29: Implementing SCR 2.4 using OWL

itself defined by another relation. Our examples with the concept of cardinality have always been with an absolute value which is part of the schema, hence all instance must respect the same cardinality.

1.7.5 Dependency XPATH

The IP-XACT metadata specification allows design models to contain values which are defined with mathematical equations based values present in the design models. These equations are expressed using XPath 1.0 expressions. The specification also defines a list of XPATH functions which extend the default library for expressing these equations. Figure 1.31 illustrates a typical example for the use of dependency expressions. The example defines the "base address of a certain memory map" as a function of parameters of another memory map. By using SWRL rules, such dependency expression may be defined. Execution of the SWRL rule by an engine will result in the evaluation of the expressions. Figure 1.31 illustrates the XML oriented approach defined by IP-XACT as well as the equivalent using SWRL. The SWRL portion defines custom built-ins which have a prefix of spiritb, these are functions which are not part of the default SWRL built-ins; they have the same meaning as their equivalent in the XML version. The main differences between both approaches are:

1. In the SWRL version, because of the predicate nature of the rules, it is not possible to define expressions which use imbricate built-ins. It is necessary to define variables for each intermediate calculation.

2. In the IP-XACT version, it is necessary to use spirit:id for values in order to references them in calculations.

```
spirit:AddressSpace rdf:type owl:Class.
spirit:hasBitsInLAU rdf:type rdf:DataTypeProperty.
spirit:hasAddressSpace rdf:type rdf:ObjectProperty.
spirit:Interface rdfs:subClassOf _:InterfaceRestriction1.
_:InterfaceRestriction1 owl:onProperty spirit:hasAddressSpace.
_:InterfaceRestriction1 owl:cardinality 1.
spirit:AddressSpace rdfs:subClassOf _:AddressSpaceRestriction1.
_:AddressSpaceRestriction1 owl:onProperty spirit:hasBitsInLAU.
_:AddressSpaceRestriction1 owl:cardinality 1.
spirit:MasterInterface rdfs:subClassOf spirit:Interface.
spirit:SlaveInterface rdfs:subClassOf spirit:Interface.
```

```
spirit:MasterInterconnection(?connection)
spirit:hasMainInterface(?connection, ?x)
spirit:hasSecondaryInterface(?connection, ?y)
spirit:MasterInterface(?x)
spirit:SlaveInterface(?y)
spirit:AddressSpace(?space1)
spirit:hasAddressSpace(?x, ?space1)
spirit:hasBitsInLAU(?space1?, bits_x)
spirit:AddressSpace(?space2)
spirit:hasAddressSpace(?y, ?space2)
spirit:hasBitsInLAU(?space2?, bits_y)
->
swrlb:equal(?bits_x, bits_y)
```

FIGURE 1.30: Implementing SCR 2.10 using OWL

The SWRL approach is a more verbose than the XPATH approach, but it has the added advantage of:

1. Not requiring spirit:id to be define, hence it is possible to use values which were not initially intended to be referred.

2. Not requiring the implementation of a pre-processing stage of XML models. The IP-XACT approach requires custom code to written to interpret the embedded XPATH expressions

1.8 Cost of adoption

Migration of the SPIRIT standards to the Semantic Web would offer many benefits which are important with regards to expressivity, simplicity and flexibility. However, nothing comes without a price. The migration of the SPIRIT standard as well as tools which have been developed by vendors which have adopted the standard would consist of three primary tasks:

```
<spirit:memoryMaps>
  <spirit:memoryMap>
    <spirit:name>mmap</spirit:name>
    <spirit:addressBlock>
      <spirit:name>abl</spirit:name>
      <spirit:baseAddress spirit:resolve="user"
        spirit:id="baseAddr">0</spirit:baseAddress>
      <spirit:range spirit:id="range">786432</spirit:range>
      <spirit:width>32</spirit:width>
      <spirit:usage>memory</spirit:usage>
      <spirit:access>read-write</spirit:access>
    </spirit:addressBlock>
  </spirit:memoryMap>
<spirit:memoryMap>
  <spirit:name>dependent_mmap</spirit:name>
    <spirit:addressBlock>
    <spirit:baseAddress spirit:resolve="dependent"
      spirit:dependency="spirit:
                pow(2,
                  floor(spirit:log(2,spirit:decode(id('baseAddr'))+
                    spirit:decode(id('range')))+1))"
              spirit:id="dependentBaseAddress">0</spirit:baseAddress>
      <spirit:range>4096</spirit:range>
      <spirit:width>32</spirit:width>
      <spirit:usage>register</spirit:usage>
      <spirit:access>read-write</spirit:access>
    </spirit:addressBlock>
  </spirit:memoryMap>
</spirit:memoryMaps>
```

```
spirit:MemoryMap(?mm1)
spirit:hasName(?mm,"mmap")
spirit:MemoryMap(?mm2)
spirit:hasName (?mm, "dependent_mmap")
spirit:hasRange(?mm,?range)
spirit:hasBaseAddress(?mm,?baseAddr)
spiritb:decode(?decRange,?range)
spiritb:decode(?decBaseAddr,?baseAddr)
swrlb:add(?tmpRange,?decRange 1)
spiritb:log(?tmpLog, 2, ?decBaseAddr)
swrlb:floor(?floorTmp, ?tmpLog)
swrlb:pow(?powTmp, 2, ?floorTmp)
swrlb:add(?dependent_baseAddr, ?powTmp, ?tmpRange)
->
spirit:hasBaseAddress(?mm2, ?dependent_baseAddr)
```

FIGURE 1.31: Dependency XPATH example

1. Develop an OWL and SWRL based model for the IP-XACT standards;
2. Migrate all IP descriptions and design models to the new ontology;
3. Replace the XML consumption portion of current tools with an implementation based either SPARQL or on the JENA toolkit.

The first task would not be very difficult for the most demanding portion of creation an OWL model is determining the required semantics and achieving consensus. This has already been achieved through the development of the IP-XACT metadata format. The second task will probably be the most demanding, however by using a

combination of XSLT scripts and Perl scripts, it would probably be possible to automate a large portion of the migration. The third task, despite being straight forward once an OWL model has been established, will require a fair amount of development, however we believe that the mid to long term benefits out-weigh by far the cost. Some of the secondary tasks would be the selection of an inferences engine as well as the development of the necessary extension functions for SWRL rules.

1.9 Future Research

This article discusses a possible path for the use of the Semantic Web technologies in the context of EDA. Based on this work, many other aspects are left to be explored. The development of a complete ontology for the IP-XACT standards would offer many interesting challenges with regards to semantic modeling. It would also be vary interesting to apply the ideas in the paper to other work such as Colif and MoML. A discussion on the quantity of code required to implement a Semantic Web approach versus a traditional XML approach would be interesting in order to guide further implements. The whole aspect of performance benchmarking is also to be explored and discussed. Moreover, comparing the effectiveness of modeling with regards to time and complexity would be interesting in order to measure designers comfort with this approach compared to approaches based on XML.

1.10 Conclusion

The XML technology stack has significantly helped the EDA industry over the last decade by simplifying the exchange of information between tools. It has also given developers an effective mean for the creation of simple markup-based languages. We believe that the Semantic Web technology stack is the next step. The next generation of EDA tools will benefit in multiples ways by adopting a technology which is focused solely on semantics and not syntax. This paper has presented the major benefits of the Semantic Web technologies over XML in general. It also discussed key benefits for the IP-XACT standard if it adopts these new technologies. We believe that the benefits of adopting the semantic web technologies outweigh by far it inconveniences.

2

References

[1] Amba specification (rev2.0) and multi layer ahb specification, 2001.

[2] Dean Allemang and James A. Hendler. *Semantic web for the working ontologist : modeling in RDF, RDFS and OWL.* Morgan Kaufmann Publishers/Elsevier, Amsterdam; Boston, 2008.

[3] Christopher J.O. Baker and Kei-Hoi Cheung, editors. *Semantic Web : Revolutionizing Knowledge Discovery in the Life Sciences.* Springer, 2007.

[4] Felice Balarin, Yosinori Watanabe, Harry Hsieh, Luciano Lavagno, Claudio Passerone, and Alberto Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *Computer*, 36(4):45–52, 2003.

[5] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernndez, Michael Kay, Jonathan Robie, and Jrme Simon. Xml path language (xpath) 2.0. Technical report, 23 January 2007.

[6] T. Berners-Lee. N3 notation - http://www.w3.org/designissues/notation3.

[7] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):28–37, 2001.

[8] Scott Boag, Don Chamberlin, Mary F. Fernndez, Daniela Florescu, Jonathan Robie, and Jrme Simon. Xquery 1.0: An xml query language. Technical report, W3C Recommendation - http://www.w3.org/TR/xquery/, 23 January 2007.

[9] Rumbaugh J. Jacobson I. Booch, G. *The Unified Modeling Language User Guide 2 ed.* Addison Wesley Professional, 2005.

[10] Jerry Burch, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. Overcoming heterophobia: Modeling concurrency in heterogeneous systems. *Application of Concurrency to System Design, International Conference on*, 0:13, 2001.

[11] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83, New York, NY, USA, 2004. ACM.

[12] W.O. Cesario, G. Nicolescu, L. Gauthier, D. Lyonnard, and A.A. Jerraya. Colif: A design representation for application-specific multiprocessor socs. *Design and Test of Computers, IEEE*, 18(5):8–20, Sep-Oct 2001.

[13] Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. Sparql protocol for rdf. Technical report, W3C Recommendation, 15 January 2008.

[14] C. J. Date. *An Introduction to Database Systems 7ed.* Addison Wesley Longman, 2000.

[15] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The semantic web: the roles of xml and rdf. *Internet Computing, IEEE*, 4(5):63–73, Sep/Oct 2000.

[16] E. Sirin et al. Pellet: A practical owl-dl reasoner. In *Web Semantics: Science, Services and Agents on the World Wide Web*, pages 51–53, 2007.

[17] T. Grotker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC.* Kluwer Academic Publications, Boston, 2002.

[18] Michael Grove and Andrew Schain. Pops nasas expertise location service powered by semantic web technologies. Technical report, W3C Semantic Web Case Studies and Use Cases - http://www.w3.org/2001/sw/sweo/public/UseCases/Nasa/Nasa.pdf, 2008.

[19] A. Horn. On sentences which are true of direct unions of algebras. *Journal of symbolic logic*, pages 14–21, 1951.

[20] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml. Technical report, W3C Member Submission - http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/, 21 May 2004.

[21] Intel. Ixp45x datasheet, http://www.intel.com/design/network/datashts/306261.htm.

[22] Albert Carl Jan KIENHUIS Kienhuis. *Design Space Exploration of Stream-based Dataflow Architctures: Methods and Tools.* PhD thesis, Delft University of Technology, 1999.

[23] Holger Knublauch, Mark A Musen, and Alan L Rector. Editing description logic ontologies with the protege owl plugin. In *In Description Logics*, 2004.

[24] Wido Kruijtzer, Pieter van der Wolf, Erwin de Kock, Jan Stuyt, Wolfgang Ecker, Albrecht Mayer, Serge Hustin, Christophe Amerijckx, Serge de Paoli, and Emmanuel Vaumorin. Industrial ip integration flows based on ip-xact standards. *Design, Automation and Test in Europe, 2008. DATE '08*, pages 32–37, March 2008.

[25] E. A. Lee and S. Neuendorffer. Moml a modeling markup language in xml, version 0.4. Technical Report ERL/UCB M 00/12, University of California at Berkeley,, 2000.

[26] Nenad Medvidovic and Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[27] E. K. Neumann and D. Quan. Biodash: A semantic web dashboard for drug development. In *Pacific Symposium on Biocomputing*, pages 176–187, 2006.

[28] E. Prudhommeaux and A. Seaborne. Sparql query language for rdf. Technical Report REC-rdf-schema-20040210, World Wide Web Consortium, Jan. 2008.

[29] S. Stuijk. *Predictable Mapping of Streaming Applications on Multiprocessors*. PhD thesis, Eindhoven University of Technology, 2007.

[30] B.D. Theelen. Performance model generation for mpsoc design-space exploration. *Quantitative Evaluation of Systems, 2008. QEST '08. Fifth International Conference on*, pages 39–40, Sept. 2008.

[31] K. C. Thramboulidis, G. Doukas, and G. Koumoutsos. A soa-based embedded systems development environment for industrial automation. *EURASIP J. Embedded Syst.*, 2008(1):1–15, 2008.

[32] P.H.A. van der Putten and J.P.M. Voeten. *Specification of Reactive Hardware/Software Systems: The method Software/Hardware Engineering (SHE)*. Ph.d., Eindhoven University of Technology, 1997.

Conclusion et Travaux Futurs

Malgré les complexités liées à la conception des systèmes embarqués, ces petits « bidules » jouent un rôle clé dans notre quotidien en raison de leur omniprésence. De plus, il serait fort peu probable que leurs rôles cessent de croître dans le futur. Au contraire, on peut facilement imaginer une évolution exponentielle de leur présence. Afin de soutenir cette forte croissance, il est impératif de définir de nouvelles approches pour leur réalisation supportant adéquatement les exigences spécifiques de conception de chacun. Ces approches devront donc être facilement personnalisables afin de répondre à cette diversité d'exigences.

La génération courante d'outils CAO, malgré tous les bénéfices qu'ils apportent, ne supporte pas, généralement, complètement les méthodologies de conception. Les discontinuités engendrées dans le flux de conception causées par ce support incomplet rend inefficace le travail des concepteurs de systèmes. De plus, les outils actuels, pour de multiples raisons, supportent médiocrement la personnalisation ainsi que l'intégration avec des outils tierce partie afin de permettre la spécialisation des flux de conception.

La communauté du logiciel est riche en techniques et technologies pour gérer la complexité du processus de développement de systèmes logiciels d'envergure ainsi que leur intégration. Ce savoir-faire a émergé en raison de facteurs très similaires que ceux présents dans la communauté des systèmes embarqués : la complexité croissante des systèmes et la décroissance du temps de mise en marché attendue. Le domaine de la conception des systèmes embarqués peut gagner beaucoup en mettant à profit le savoir-faire de la communauté du logiciel afin de concevoir une nouvelle génération d'outils CAO. Ces outils seront plus personnalisables et supporteront d'avantage les méthodologies de conception ainsi que leurs spécialisations.

Plus particulièrement, en combinant l'utilisation d'une plateforme moderne de développement de logiciels tel que la plateforme .Net et des patrons de conception de

logiciels, il est possible de créer une nouvelle génération d'outils de modélisation et de simulation de systèmes avec des caractéristique fort intéressantes. Ces outils prennent la forme de langages de spécification orientée bibliothèque n'ayant pas certain des inconvénients majeurs des outils actuels (i.e. SystemC). Ils offrent :

- des langages hôtes plus simple en matière de syntaxe et de gestion de mémoire rendant la modélisation plus efficace et moins prompt à l'erreur;
- des environnent d'exécution fonctionnellement riche (i.e. introspection, intégration web, multitâches, etc.) permettant :
    - le développement rapide d'outils d'analyse,
    - le support de plusieurs langages hôtes,
    - l'intégration d'outils de tierce partie plus efficace,
    - la simulation parallèle plus simple;
- une séparation d'aspect entre la modélisation et la simulation permettant :
    - un meilleur support pour la conception par IP,
    - l'intégration d'outils de tierce partie plus efficace,
    - une exploration de l'espace de conception plus simple.

Les points ci-haut sont des avantages importants pour cette nouvelle génération d'outils comparativement à la génération précédente et ceci malgré une perte de rapidité d'exécution lors de la simulation, le désavantage principal de l'approche. Ce désavantage devrait s'amoindrir dans le temps avec le raffinement de la compilation dynamique ainsi que l'implémentation plus efficace des engins de simulation.

L'incorporation des technologies du Web Sémantique dans la conception des outils d'aide à la conception est une autre approche prometteuse. Cette approche permettra de concevoir des outils supportant plus efficacement la conception par IP. Cette efficacité est attribuable à :

- une utilisation et un partage plus simple des métadonnées décrivant les IP et les systèmes (les connaissances);

- la conception collaborative via la fédération de connaissances distribuées;
- l'incorporation simple du support à la prise de décisions via l'inférence de connaissances ainsi que la validation formelle.

Les points ci-haut permettent d'entrevoir une génération d'outils supportant mieux la conception par IP ainsi que la démocratisation des technologies de gestion des connaissances.

Les contributions de ce travail se sont effectuées sur deux grandes lignes. La première est une étude à caractère pédagogique sur l'approche « Y-Chart ». Dans un premier temps, cette étude présente l'influence de la conception dirigée par les modèles, une technique de la communauté du logiciel sur l'approche « Y-Chart ». Dans un deuxième temps, cette étude présente et discute des diverses facettes de cette approche : son histoire, ses principaux concepts, son implémentation et son utilisation dans divers domaines d'applications. Une grande partie de cette présentation/discussion se fait par l'entremise d'une comparaison de trois méthodologies qui ont implanté l'approche soit Metropolis, SHE-POOSL et DOL-MPA.

La deuxième grande ligne est sur trois travaux qui transposent les technologies et techniques de pointes du domaine du génie logiciel au domaine de la CAO des systèmes embarqués. Cette transposition a permis l'élaboration de nouvelles approches innovatrices. Dans un premier temps, une nouvelle méthodologie pour la conception d'outils CAO a été présentée. Cette méthodologie s'appuie sur: (i) un nouveau flux de conception ciblé sur les représentations de modèle et (ii) l'utilisation des technologies .Net pour concevoir de meilleures architectures d'outils CAO. Afin de démontrer cette méthodologie, une réalisation de celle-ci a été faite : Esys.Net.

Dans un deuxième temps, les idées de ce dernier travail ont été approfondies afin de proposer une architecture cible innovatrice pour la prochaine génération d'outils CAO. Cette architecture cible réussit à créer une séparation parfaite des aspects entourant la conception des systèmes embarqués. Cette architecture a été possible grâce à la nouvelle génération des technologies .Net. Afin de démontrer cette architecture, une réalisation de celle-ci a été faite : SoCML.

Dans un troisième temps, les technologies du Web sémantique ont été introduites à la communauté du matériel, ainsi qu'une étude de cas sur l'utilisation de celle-ci pour la conception à base d'IP. Dans ce travail, une présentation à caractère pédagogique des technologies du Web sémantique est proposée afin d'introduire tous les concepts et technologies clés. Ensuite, une discussion sur les avantages de ces technologies (vs les technologies XML) pour gestion de la sémantique et la conception à base d'IP est présentée. Le travail conclut sur une étude de cas dans lequel la spécification IP-XACT est utilisée afin de démontrer les bénéfices que ce standard pourrait tirer s'il utilisait une implémentation basée sur les technologies du Web sémantique.

1.8 Développements possibles

Les travaux effectués dans le cadre de cette thèse peuvent être approfondis selon deux grandes voies. Une première voie de recherche serait de pousser les travaux de SoCML plus loin afin de :

- définir une solution de modélisation et de simulation plus complète et d'optimiser les divers aspects d'implémentation de celles-ci (syntaxe, performance, etc.) ;
- expérimenter avec des solutions de simulation intégrant des capacités d'émulation, de vérification, de « CoDesign », etc.

La deuxième voie de recherche serait de continuer les travaux reliés au Web sémantique afin d'élaborer une solution complète pour le standard IP-XACT basée sur OWL et SWRL. Une fois cette solution développée, une étude comparative pourrait être faite afin de discuter de l'ensemble des implications de l'approche.

De plus, il serait fort intéressant de développer un environnement intégré à la conception des systèmes embarqués s'appuyant sur l'ensemble des idées discutées dans cette thèse. Cet environnent aurait au minimum les caractéristiques suivantes :

- le cœur de la solution serait basé sur une plateforme de type « Framework » pour la modélisation et la simulation de système matériel/logiciel offrant une séparation des aspects parfaites comme SoCML. Cette plateforme serait développée avec les technologies .Net;
- l'environnement et la plateforme de modélisation/simulation supporteraient les trois méthodologies de conceptions suivantes: (i) l'élaboration par

raffinements successifs, (ii) le paradigme « Y-Chart » et la conception à base d'IP;

- la gestion et le partage des métadonnées de l'environnent serait fait avec une ontologie élaborée avec les technologies sémantiques, ainsi que l'API de l'environnent.

Sources documentaires

[1] .NET Framework Home Page, http://www.microsoft.com/net, 2003

[2] "Moores Law", http://www.webopedia.com/TERM/M/Moores_Law.html, 2003

[3] Albahari B. A, Comparative Overview of C#. Available from: ttp://genamics.com/developer/csharp comparative.htm., 2000

[4] Allemang D. and Hendler J., *"Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL "*, Morgan Kaufmann , 2008

[5] Bailey S. *"VHDL-200X improves design and verification"*, EEDesign, Nov 7, 2003

[6] Bailey S., *"Comparison of VHDL, Verilog and SystemVerilog"*, Model Technology, Digital Simulation White Paper, 2003

[7] Baker C. J.O., and . Cheung K.-H (Eds.), *"Semantic Web : Revolutionizing Knowledge Discovery in the Life Sciences"*, Springer, 2007

[8] Balarin F. et al., *"Metropolis: An Integrated Electronic System Design Environment"*. IEEE Computer, 36(4), pp. 45-52, 2003

[9] Bellows P. and Hutchings B., *"JHDL - an HDL for reconfigurable systems"*, IEEE Symposium on FPGAs for Custom Computing Machines, pp. 175-184, 1998

[10] Berners-Lee T., Hendler J. and Lassila O., *"The Semantic Web."* New York: Scientific America, 284(5), pp. 34-43, 2001

[11] Bezivin, J, *"On the Unification Power of Models"*, Software and System Modeling 4(2), pp. 171–188, 2005

[12] Booch G., *"Object-Oriented Design"*, Ada Letters Volume 1 (3), pp. 64-76, 1982

[13] Borrione D. et al, *"Three decades of HDLs. II. Conlan through Verilog"*, Design & Test of Computers, IEEE, vol. 9, pp. 54-63, 1992

[14] Buchenrieder K., Pyttel A. and Sedlmeier A., *"A powerful system design methodology combining OCAPI and Handel-C for concept engineering"*, Design, Automation and Test in Europe, pp. 870-874, 2002

[15] Burch J. R., Passerone R. and Sangiovanni-Vincentelli A., *"Overcoming Heterophobia: Modeling Concurrency in Heterogeneous Systems."*, Proceedings of ACSD, pp. 13-32, 2001

[16] Cai L, Verma S. andGajski D., *"Comparison of SpecC and SystemC Languages for System Design"* ,Technical Report CECS-03-11, 2003

[17] Cesario W.O., et al, *"Colif: A Design Representation for Application-Specific Multiprocessor SOCs"*, IEEE Design and Test of Computers, pp. 8-19, 2001

[18] Chang H et al, *"Surviving the SOC Revolution : A Guide to Platfrom-Based Design,"*, Klumer Academic, 1999

[19] Charest L. and Aboulhamid E.M., *"A VHDL/SystemC Comparison in Handling Design Reuse"*, International Workshop on System-on-Chip for Real-Time Applications, pp. 79-85, 2002

[20] Charest L., Aboulhamid E.M., and Bois G., *"Applying patterns and multi-paradigm approaches to hardware/software design and reuse"*, in Patterns And Skeletons For Parallel And Distributed Computing, F. Rabhi and S. Gorlatch, Eds. London: Springer-Verlag, pp. 297-325, 2002

[21] Chu Y. et al, *"Three decades of HDLs. I. CDL through TI-HDL"*, Design & Test of Computers, IEEE, vol. 9, pp. 54-63, 1992

[22] Delpasso M., Bogliolo A. and Benini L., *"Virtual Simulation of Distributed IP-Based Designs"*, Design Automation Conference, 1999

[23] DeMarco T., *"Object Structured Analysis and System Specification"* Prentice-Hall, 1979

[24] Donlin A., "*Transaction Level Modeling: Flows and Use Models*", CODES+ISSS'O4, pp. 75-80, 2004

[25] Doucet F., Shukla S. and Gupta R., *"Introspection in system-level language frameworks: meta-level vs. integrated"*, Design, Automation and Test in Europe, 2003

[26] Doulos, "*SystemC In Europe: Current Usage and Future Requirements*", http://www.doulos.com/, 2003

[27] Drucker L., *"SystemC Verification Library speeds transaction-based verification"*, EEdesign, Feb 24, 2003

[28] ECMA International, *"ECMA and ISO/IEC C# and Common Language Infrastructure Standards"*, http://msdn.microsoft.com/net/ecma/ ISO/IEC 23270 to ISO/IEC 23272, 2003

[29] Ferrandi F., *"Functional verification for SystemC descriptions using constraint solving"*, Design, Automation and Test in Europe , pp. 744-751, 2002

[30] Gajski D. and Kuhn R.H., *"New VLSI Tools"*, IEEE Computer, pp. 11-14, 1983

[31] Gamma E., et al, *"Design Patterns: Elements of Reusable Object-Oriented Software*", Addison-Wesley, 1994

[32] Gorse N, et al., *"Enhancing ESys.Net with a semi-formal verification layer"*. In Proceedings of the 16th IEEE Intl Conference on Microelectronics (ICM'04), pp. 388–391. 2004

[33] Gough K. J., *"Stacking them up: a comparison of virtual machines"*, Computer Systems Architecture Conference, pp. 55-61, 2001

[34] Grove M., et al, *"Case Study: POPS – NASA's Expertise Location Service Powered by Semantic Web Technologies"*, available at : http://www.w3.org/2001/sw/sweo/public/UseCases/Nasa/Nasa.pdf, 2008

[35] Hara Y. , *"Researchers describe embedded processor design tool"*, EEDesign, May 9, 2002

[36] Hutchings, B. et al, *"Developing and debugging FPGA applications in hardware with JHDL"*, Thirty-Third Asilomar Conference on Signals, Systems, and Computers, Vol 1, pp. 554-558 1999

[37] IEEE, "IEEE standard hardware description language based on the Verilog(R) hardware description language", in IEEE Std 1364-1995, 1996

[38] IEEE, "IEEE standard VHDL analog and mixed-signal extensions", in IEEE Std 1076.1-1999, 1999

[39] IEEE, "IEEE standard VHDL language reference manual", in IEEE Std 1076-1987, 1988

[40] ITRS, "*International Technology Roadmap for Semiconductors, Design*", 2007

[41] Jerraya A. and Ernst R., *"Multi-language system design"*, Design, Automation and Test in Europe, 1999

[42] Keating M. and Bricaud P., *"Reuse Methodology Manual"*, Kluwer Academic Publisher, 1999

[43] Kienhuis B., *" Design Space Exploration of Stream-based Dataflow Architctures: Methods and Tools."*, Unpublished doctoral dissertation, Delft Univesity Technology, The Netherlands, 1999

[44] Kilgore R.A., *"Multi-language, open-source modeling using the Microsoft .NET architecture"*, Winter Simulation Conference, 2002

[45] Kleppe A., Warmer J. and Bast W, *"MDA Explained, The Model Driven Architecture : Practice and Promise"*, Addison-Wesley, 2003

[46] Kruijtzer W., et al., *"Industrial IP Integration Flows based on IP-XACT Standards"*, Design, Automation and Test in Europe, pp. 26-31, 2008

[47] Lapalme J., Aboulhamid EM., Nicolescu G., and Rousseau F., « *Separating Modeling and Simulation Aspects in Hardware/Software Framework-Based*

Modeling Languages », The Arabian Journal for Science and Engineering, 2007.

[48] Lapalme J., Aboulahmid E.M. and Nicolescu G. « *The Semantic Web Applied to IP-Based Design : A Discussion on IP-XACT* » in System level design with .Net technology, E.M. Aboulhamid and F. Rousseau Eds., CRC Press. (to be published)

[49] Lapalme J., Aboulahmid E.M. and Nicolescu G., « *A New Efficient EDA Tool Design Methodology* », ACM Transactions on Embedded Computing Systems Special Issue on Concurrent Hardware-Software Design Methods for MPSoC, 2006

[50] Lapalme J., Theelen B., Stoimenov N., Voeten J., Thiele L. and Aboulhamid EM., " *Y-Chart Based System Design : A Discussion on Approaches*", ACM Transactions on Embedded Computing Systems. (submitted)

[51] Lee E. A. and Neuendorffer S., *"MoML - A Modeling Markup Language in XML, Version 0.4"*, Technical Memorandum UCB/ERL M00/12, U. of California, Berkeley, 2000

[52] Lutz M. H. L. and Laplante P.A., *"C# and the .NET framework: ready for real time?"*, IEEE Software , vol. 20, pp. 74-80, 2003

[53] Manola F. and Miller E., "RDF Primer", W3C Recommendation, 2004

[54] Martignano M. D., Fummi N. and Martini S., *"A combined approach to validate the design of embedded network devices"*, IEEE International Symposium on Circuits and Systems (ISCAS), pp. III-169-III-172 vol.3, 2002

[55] Martin G. *"Design Methodologies for System Level IP"*, Design, Automation and Testin Europe, pp 286-289, 1998

[56] Martin G. *"SystemC and the Future of Design Languages: Opportunities for Users and Research"*, SBCC, 2003.

[57] McGuinness D. L. and Harmelen F. v., *"OWL Web Ontology Language Overview"*, W3C Recommendation, Feb, 2004

[58] Medvidovic N. and Taylor R. N. *"A Classification and Comparison", Framework for Software Architecture Description Languages*. Transactions on Software Engineering, 26 (1), pp. 70-93, 2000

[59] Meijer E., Miller J. Technical *"Overview of the Common Language Runtime"* research.microsoft.com/~emeijer/Papers/CLR.pdf, 2003

[60] Mernik M. et al, *"When and How to Develop Domain-Specific Languages"*, in ACM Computing Surveys. New York : ACM, 2005, p. 316.

[61] Neumann E. K., and Quan D., *"Biodash: A Semantic Web Dashboard for Drug Development"*, Pacific Symposium on Biocomputing, 2006, pp. 176-187

[62] Newkirk J. V. and Vorontsov A.A., *"How .NET's custom attributes affect design"*, IEEE Software, vol. 19, pp. 18-20, 2002

[63] Nicolescu G. et al., *"Validation in a Component-Based Design Flow for Multicore SoCs"*, International Symposium on Systems Synthesis, 2002

[64] OSCI. SystemC 2.1 Language Reference Manual. Available from: http://www.systemc.org/., 2005

[65] Rich D.I , *"The evolution of SystemVerilog"*, IEEE Design & Test of Computers, Volume: 20 Issue: 4, July-August, 2003

[66] Shaler S. and Mellor S.J., *"Object Lifecycles: Modeling the World in States"*, Yourdon Press, 1991

[67] Singer J. ,*"JVM versus CLR: A Comparative Study"*, International Conference on Principles and Practice of Programming in Java, p. 167-169, 2003

[68] SpecC Home Page, http://www.ics.uci.edu/~specc/index.html, 2003

[69] Theelen B.D., *"Performance Model Generation for MPSoC Design-Space Exploration."*, Proceedings of QEST. IEEE., 2008

[70] Thramboulidis K.C., Doukas G., and Koumoutsos G., *"A SOA-Based Embedded Systems Devleopment Environment for Industrial Automation"*, EURASIP Jounal on Embedded Systems, 2008

[71] Vahid F. et al, *"Specification and design of embedded systems"*, Prentice-Hall, 1994

[72] Van Der Putten P.H.A., and Voeten J.P.M. *"Specification of Reactive Hardware/Software Systems."*, Unpublished doctoral dissertation, Eindhoven University of Technology, Eindhoven, The Netherlands,1997

[73] W3C. Extensible Markup Language (XML). Available from: http://www.w3.org/XML., 2005

[74] Wikipedia, *"Hardware Description Language Definition"*, http://en2.wikipedia.org/wiki/Hardware_description_language, 2003

[75] Yoo S. et al., *"Building Fast and Accurate SW Simulation Models Based on Hardware Abstraction Layer and Simulation Environment Abstraction Layer"*, Design, Automation and Test in Europe, 2003

Liste des contributions

Sections de livre

- F. Rousseau, E.M. Aboulhamid and J. Lapalme, « Introduction » in System level design with .Net technology, E.M. Aboulhamid and F. Rousseau Eds., CRC Press. (To be published)
- J. Lapalme, E.M. Aboulahmid and G. Nicolescu « The Semantic Web Applied to IP-Based Design : A Discussion on IP-XACT » in System level design with .Net technology, E.M. Aboulhamid and F. Rousseau Eds., CRC Press. (To be published)
- J. Lapalme « Esys.Net Environment» in System level design with .Net technology, E.M. Aboulhamid and F. Rousseau Eds., CRC Press. (To be published)

Journals

- J. Lapalme, B. Theelen, N. Stoimenov, J. Voeten, L. Thiele and EM. Aboulhamid, « Y-Chart Based System Design : A Discussion on Approaches », ACM Transactions on Embedded Computing Systems, 34 pages (submitted)
- J. Lapalme, EM. Aboulhamid, G. Nicolescu, and F. Rousseau,« Separating Modeling and Simulation Aspects in Hardware/Software Framework-Based Modeling Languages », The Arabian Journal for Science and Engineering, 2007, 20 pages
- J. Lapalme, EM. Aboulhamid, and G. Nicolescu, « A New Efficient EDA Tool Design Methodology », ACM Transactions on Embedded Computing Systems Special Issue on Concurrent Hardware-Software Design Methods for MPSoC, 2006, 23 pages

Actes de conférence

- M. Kastle, J. Lapalme and EM. Aboulhamid, « Dynamic proxy generation for a Service-Oriented Architecture simulator », NEWCAS-TAISA, 2008, 4 pages
- H. Balen and J. Lapalme , «Panel on : Domain Specific Languages - Another Silver Bullet? », 22th ACM Object-Oriented Programming, Systems, Languages & Application 2007 (OOPSLA), Montreal, Quebec, Canada, Oct 21- 25, 2007, 2 pages
- J.Lapalme, E.M. Aboulhamid, G. Nicolescu and F. Rousseau, « Separating Modeling and Simulation Aspects in Hardware/Software System Design », *18th International Conference on Microelectronics (ICM) IEEE*, Dec 16-19, Dhahran, Saudi Arabia, 2006, 6 pages
- J. Lapalme, EM. Aboulhamid et G. Nicolescu « Leveraging Model Representations for System Level Design Tools. »,16th IEEE International Workshop on Rapid System Prototyping, June 8-10, Montreal, Canada, 2005, 6 pages
- N. Gorse, EM. Metzger, J. Lapalme, E.M. Aboulhamid, Y. Savaria and G. Nicolescu « Enhancing ESys.Net With a Semi-Formal Verification Layer », 16th International Conference on Microelectronics (ICM) IEEE, Dec 6-8, Tunis, Tunisia, 2004, 6 pages
- J. Lapalme, EM. Aboulhamid, G. Nicolescu, L. Charest, F. Boyer, J-P. David et G. Bois, « ESys.Net – A New Solution for Embedded Systems Modeling and Simulation », Proceedings of ACM SIGPLAN/SIGBED Conference onLanguages, Compilers, and Tools for Embedded Systems (LCTES), Washington, DC, USA, June 11-13, 2004, 6 pages

Autres

- J. Lapalme, « Separating DSL Semantics form Implementation », MSDN (http://msdn.microsoft.com/en-us/library/bb896746.aspx), 2007, 13 pages