Université de Montréal

Timing verification in Transaction Modeling

par
Alena Tsikhanovich

Département d'Informatique et de Recherche Opérationnelle
Faculté des Arts et des Sciences

Thèse présentée à la Faculté des Arts et des Sciences en vue de l'obtention du grade de Philosophiae Doctor (Ph.D.) en Informatique

Décembre 2009

Université de Montréal
Faculté des Arts et des Sciences

Cette thèse intitulée:

Timing verification in Transaction Modeling

Présentée par :
Alena Tsikhanovich

a été évaluée par un jury composé des personnes suivantes :

Abdelhakim Hafid, président-rapporteur
El Mostapha Aboulhamid, directeur de recherche
Guy Bois, co-directeur
Jean-Pierre David, membre du jury
Smail Niar, examinateur externe
Jean-François Angers, représentant du doyen de la FES

Résumé

Les systèmes Matériels/Logiciels deviennent indispensables dans tous les aspects de la vie quotidienne. La présence croissante de ces systèmes dans les différents produits et services incite à trouver des méthodes pour les développer efficacement. Mais une conception efficace de ces systèmes est limitée par plusieurs facteurs, certains d'entre eux sont: la complexité croissante des applications, une augmentation de la densité d'intégration, la nature hétérogène des produits et services, la diminution de temps d'accès au marché. Une modélisation transactionnelle (TLM) est considérée comme un paradigme prometteur permettant de gérer la complexité de conception et fournissant des moyens d'exploration et de validation d'alternatives de conception à des niveaux d'abstraction élevés.

Cette recherche propose une méthodologie d'expression de temps dans TLM basée sur une analyse de contraintes temporelles. Nous proposons d'utiliser une combinaison de deux paradigmes de développement pour accélérer la conception: le TLM d'une part et une méthodologie d'expression de temps entre différentes transactions d'autre part. Cette synergie nous permet de combiner dans un seul environnement des méthodes de simulation performantes et des méthodes analytiques formelles. Nous avons proposé un nouvel algorithme de vérification temporelle basé sur la procédure de linéarisation des contraintes de type *min/max* et une technique d'optimisation afin d'améliorer l'efficacité de l'algorithme. Nous avons complété la description mathématique de tous les types de contraintes présentées dans la littérature. Nous avons développé des méthodes d'exploration et raffinement de système de communication qui nous a permis d'utiliser les algorithmes de vérification temporelle à différents niveaux TLM.

Comme il existe plusieurs définitions du TLM, dans le cadre de notre recherche, nous avons défini une méthodologie de spécification et simulation pour des systèmes Matériel/Logiciel basée sur le paradigme de TLM. Dans cette méthodologie plusieurs concepts de modélisation peuvent être considérés séparément. Basée sur l'utilisation des technologies modernes de génie logiciel telles

que XML, XSLT, XSD, la programmation orientée objet et plusieurs autres fournies par l'environnement .Net, la méthodologie proposée présente une approche qui rend possible une réutilisation des modèles intermédiaires afin de faire face à la contrainte de temps d'accès au marché. Elle fournit une approche générale dans la modélisation du système qui sépare les différents aspects de conception tels que des modèles de calculs utilisés pour décrire le système à des niveaux d'abstraction multiples. En conséquence, dans le modèle du système nous pouvons clairement identifier la fonctionnalité du système sans les détails reliés aux plateformes de développement et ceci mènera à améliorer la "portabilité" du modèle d'application.

Mots-clés : Systèmes Matériels/Logiciels, vérification temporelle, analyse temporelle, modélisation transactionnelle, modélisation des systèmes à différents niveaux d'abstraction.

Abstract

Hardware/Software (Hw/Sw) systems are likely to become essential in all aspects of everyday life. The increasing penetration of Hw/Sw systems in products and services creates a necessity of their efficient development. However, the productive design of these systems is limited by several factors, some of them being the increasing complexity of applications, the increasing degree of integration, the heterogeneous nature of products and services as well as the shrinking of the time-to-market delay. Transaction Level Modeling (TLM) paradigm is considered as one of the most promising simulation paradigms to break down the design complexity by allowing the exploration and validation of design alternatives at high levels of abstraction.

This research proposes a timing expression methodology in TLM based on temporal constraints analysis. We propose to use a combination of two paradigms to accelerate the design process: TLM on one hand and a methodology to express timing between different transactions on the other hand. Using a timing specification model and underlining timing constraints verification algorithms can decrease the time needed for verification by simulation. Combining in one framework the simulation and analytical design exploration methods can improve the analytical power of design verification and validation. We have proposed a new timing verification algorithm based on the linearization procedure and an optimization technique to improve its efficiency. We have completed the mathematical representation of all constraint types discussed in the literature creating in this way a unified timing specification methodology that can be used in the expression of a wider class of applications than previously presented ones. We have developed the methods for communication structure exploration and refinement that permitted us to apply the timing verification algorithms in system exploration at different TLM levels.

As there are many definitions of TLM and many development environments proposing TLM in their design cycle with several pro and contra, in the context of our research we define a hardware/software (Hw/Sw) specification and simulation

methodology which supports TLM in such a way that several modeling concepts can be seen separately. Relying on the use of modern software engineering technologies such as XML, XSLT, XSD, object oriented programming and others supported by the .Net Framework, an approach that makes an intermediate design model reuse possible in order to cope with time-to-market constraint is presented. The proposed TLM design methodology provides a general approach in system modeling that separates various application modeling aspects from system specification: computational models, used in application modeling, supported by the language used for the functional specification and provided by simulator. As a result, in the system model we can clearly identify system functionality without details related to the development platform thereby leading to a better “portability” of the application model.

Keywords: Hardware/Software system modeling, timing verification, temporal constraints analysis, transaction level modeling, multiple abstraction levels.

Table of contents

List of tables

List of figures

List of acronyms

ALG	***Alg**orithmic*
ALAP	***A**s **L**ate **A**s **P**ossible*
API	***A**pplication **P**rogramming **I**nterface*
ASIC	***A**pplication **S**pecific **I**ntegrated **C**ircuit*
ASSP	***A**pplication-**S**pecific **S**tandard **P**roducts*
CA	***C**ycle **A**ccurate*
CC	***C**ycle **C**allable*
CIL	***C**ommon **I**ntermediate **L**anguage*
CLR	***C**ommon **L**anguage **R**untime*
CNF	***C**onjunctive **N**ormal **F**orm*
CP	***C**ommunicating **P**rocesses*
CPU	***C**entral **P**rocessing **U**nit*
CSP	***C**ommunicating **S**equential **P**rocesses*
DOM	***D**ocument **O**bject **M**odel*
DT	***D**iscrete **T**ime*
DTD	***D**ocument **T**ype **D**efinition*
EDA	***E**lectronic **D**esign **A**utomation*
EG	***E**vent **G**raph*
FIFO	***F**irst **I**n **F**irst **O**ut*
FSM	***F**inite **S**tate **M**achine*
HDF	***H**eterochronous **D**ata**F**low*
HDL	***H**ardware **D**escription **L**anguage*
Hw/Sw	***H**ard**w**are/**S**oft**w**are*
IP	***I**ntellectual **P**roperty*
ITRS	***I**nternational **T**echnology **R**oadmap for **S**emiconductors*

JIT	***J**ust-**I**n-**T**ime*
LOTOS	***L**anguage **O**f **T**emporal **O**rdering **S**pecifications*
OCP-IP	***O**pen **C**ore **P**rotocol – **I**nternational **P**artnership*
OSCI	***O**pen **S**ystem**C** **I**nitiative*
PV	***P**rogrammer's **V**iew*
ROM	***R**ead **O**nly **M**emory*
RTL	***R**egister **T**ransfer **L**evel*
SAX	***S**imple **A**PI for **X**ML*
SCE	***S**ystem on **C**hip **E**nvironment*
SDF	***S**ynchronous **D**ata**F**low*
SDL	***S**pecification and **D**escription **L**anguage*
SLM	***S**ystem **L**evel **M**odeling*
SOC	***S**ystem **O**n **C**hip*
SPIRIT	**S**tructure for **P**ackaging, **I**ntegrating and **R**e-using **I**P within **T**ool-flows
TLM	***T**ransaction **L**evel **M**odeling*
UM	***U**se- **M**odel*
UML	***U**nified **M**odeling **L**anguage*
VHDL	***V**HSIC **H**ardware **D**escription **L**anguage*
VHSIC	***V**ery **H**igh **S**peed **I**ntegrated **C**ircuits*
W3C	***W**orld **W**ide **W**eb **C**onsortium*
XML	*E**X**tensible **M**urkup **L**anguage*
XSLT	*E**X**tensible **S**tylesheet **L**anguage **T**ransformations*
XSD	***X**ML **S**chema **D**efinition Language*
YML	***Y**-chart **M**odeling **L**anguage*

Acknowledgments

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. In particular, I would like to thank to my thesis adviser, Professor El Mostapha Aboulhamid. I will never be able to thank you enough for the great support during my studies. My special thanks I would like to address to my co-adviser, Professor Guy Bois, for his suggestions and knowledge. Finally, I'd also like to thank all peoples with whom I had chance to work.

Chapter 1. Introduction

1.1 Motivation

Hardware/Software (Hw/Sw) systems are widely presented nowadays. Their impact on our life is very important as these systems are present in a variety of applications such as audio and video consumer products, communications, desktop and mobile computers, as well as professional areas like traffic control, environmental applications, driving and car control, medical systems, etc.

The increasing penetration of Hw/Sw systems in products and services creates a necessity of their efficient development. Several facts, which make the achievement of this goal quite complex, are:

- Increasing complexity of applications
- Increasing degree of integration
- Heterogenic nature of products and services
- Wide diversity of non-functional constraints and applications
- Shrinking time-to-market time
- Increasing importance of system flexibility [71].

According to Moore's law, the complexity of integrated circuits in terms of the number of transistors on a chip will double every 18 month. This growth of complexity of micro-electronic components will correspondingly lead to exponential growth of the complexity of Hw/Sw systems. Furthermore, as the complexity of applications is increasing continuously and modern applications describe often the functionality of different nature, the complexity growth in the number of different technologies needed in building an Hw/Sw system is evident.

The facts mentioned above render the productive design difficult, leading to a necessity of resolving the following challenges in system design:

- Assuring the reuse in different forms

- Assuring the design of the right system on target, without over and without under specification
- Supporting in the design flow a passage from executable specification to implementation
- Providing improved verification and validation techniques as these tools remain a significant bottleneck in system modeling.

All system design challenges listed above presume several opportunities for actions detailed in the embedded systems roadmap [71] and the semiconductor design roadmap [72].

1.1.1 Reuse

Reuse is an important factor in making the development process efficient. The amount of reuse in different forms has to increase drastically. In the embedded systems roadmap [71] it is indicated that the amount of reuse of existing hardware, software and Hw/Sw components has to increase from 20% to at least 80%. In this direction it is necessary to promote, facilitate and develop the reuse of intellectual property (IP) blocks comprising both hardware and software components. For example, current embedded software methods and tools do not support reuse [71]. IP block descriptions have to be standardized in order to facilitate their exchange and integration in different design methodologies. Furthermore, the IP blocks regarding behavior, cost and performance have to be presented at different abstraction levels to achieve the highest degree of reusability.

1.1.2 System specification

The increasing internal and external complexity of systems as well as their diffusion into a multi-disciplinary world makes it very difficult to establish a specification that can serve as input, following a subsequent transformation, to a suitable implementation. The heterogeneity of designed systems will require that several technologies of implementation be combined on a carrier technology. Reusable parts and different kinds of metrics quantifying the design experience must

be developed to aid the designer in balancing constraints of different nature to obtain the final specification. In the specification domain, an urgent need exists for the development of methods that close the gap between requirements and specification, in particular in the expression of different kinds of constraints. Specifications and design representations need to be augmented with means to express properties which go beyond behavior and structure. The languages used in system modeling must make possible to express different time models and different constraint types such as real-time constraints, throughput, power dissipation and silicon area.

1.1.3 Design flow

The design flow of different methodologies can be divided into two parts: the first covers the flow from idea to some form of executable specification and the second covers the flow from executable specification to final implementation. The executable specification of the two parts of the design flow can be based on various models and can be presented at different levels of abstraction varying over a wide range and representing the system description with a different amount of details. This diversity of models and representations imposes an additional demand on the correctness of the design flow, on the capabilities of the design representations and on the correctness of the semantics of these representations, leading to the formalization of the entire design process. This in particular holds for the design flow from executable specification to the final implementation. The different steps in the design flow should be connected together in a more appropriate and less error prone manner. The development methods must put more emphasis on the formal semantics and models on which the design flow and the design representations are founded. Thus, developments in the area of design representations and design languages are needed.

Currently, no design languages or representations do exist that can be used throughout the design process. Moreover, at high levels of specification we need an integral representation of components of different nature, i.e. we need an appropriate representation of physics, mechanics, etc. together with behavior and structure. The design space exploration styles have to allow for requirement analysis in a mixed

technology framework by co-simulation of executable models of heterogeneous building blocks at the appropriate abstraction levels. It is necessary to develop methods and tools to explore design decisions concerning the allocation of computation and communication to resources with the possibility of early evaluation of the consequences of system requirements and obtaining high-quality solutions.

Another aspect of supporting a passage from specification to implementation is further automation of design tools. This trend imposes supplementary exigencies on compilers and translators used in the design. In this context a compiler is any translator that translates one representation into another and performs synthesis and optimization. The ultimate goal of the compiler is automatically map the behavior expressed by an executable specification on hardware. Thus, the compiler can determine both – underlying hardware and software. The derivation of hardware architecture from behavioral specification has to be based on several cost functions. Furthermore, for this hardware the dedicated software has to be generated too. The other kind of compilers urgently needed in Hw/Sw system design is a retarget compiler that efficiently delivers a code to execute on these designs. For these compilers a suitable representation of the target architecture is needed.

1.1.4 Verification and validation techniques

At each step of the design flow it is necessary to check whether the design implements the desired functionality. There are various techniques to do this; these can be separated into formal and non-formal groups. As modern designs are quite complex, there are very large state spaces associated with them. As such, current formal verification tools are not capable of examining these spaces in their entirety. Thus, a large part of verification is effectuated by simulation and emulation. The tendency of the design complexity to grow will maintain a need for simulation. However, as stated in the International Technology Roadmap for Semiconductors (ITRS) [72], the simulation does not scale as designs grow; simulation techniques can only cover a part of the design space. Therefore some new approaches have to be proposed to cope with the design verification issue. The shift from non-formal to formal verification techniques is considered as a breakthrough of the design

verification issue. The integration of formal verification and validation techniques with the design flow are the only ways to solve the growing simulation burden of verifying the correctness of the design steps. Although formal verification and validation are not feasible for all designs, they can be applied successfully in many cases. The analytical power of verification and validation tools has to be improved. Furthermore, much more attention has to be given to the correctness of the design tools. Verification and validation will be needed at low levels of abstraction as well as at higher levels. The relation with the top-level specification is very important to allow for integration of verification techniques in the system design flow. Electronic Design Automation (EDA) tools in general deal with the register transfer level (RTL) of abstraction and the abstraction levels below. At these levels EDA tools are very powerful, but as they demand very detailed system description they are very slow and are not suitable for validation of a whole complex system that could be further implemented in software or hardware. Thus, there is a gap between the system level modeling (SLM) and register transfer level (RTL) design (Figure 1) and a strong need for tools to bridge the system level to RTL gap in design and verification exists.

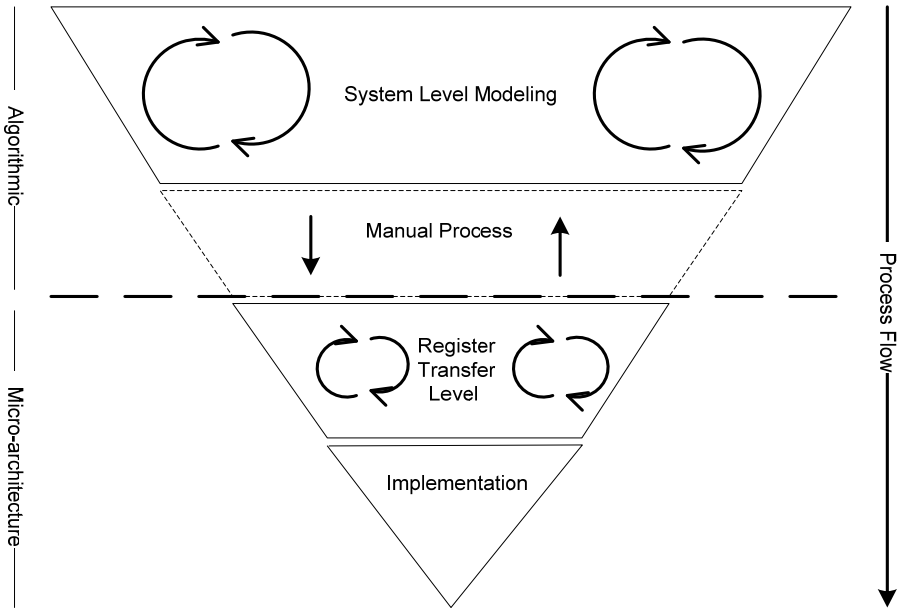

Figure 1: SLM to RTL gap [74]

Analyzing the modern trends and tendencies in hardware and Hw/Sw system design we could summarize in the next paragraph the most promising solutions dealing with the above discussed problems. These solutions have been used in our research providing in this way new methods to cope with modern system design challenges.

1.2 Objectives and proposed approach

Raising the level of design abstraction, the need to increase the design productivity and reuse in all forms and at all levels are a must in designing increasingly complex systems. The necessity of a design methodology covering a passage from executable specification to implementation providing the system description and refinement at different abstraction levels, addressing the design productivity challenge, supporting reuse in different forms, providing sophisticated verification and validation techniques becomes evident. Several solutions aimed at efficient development were recently proposed.

The Transaction Level Modeling (TLM) paradigm supporting multiple higher than register transfer abstraction levels is one of these solutions. In TLM, the computation and communication components are modeled separately. Details of communication and computation are added gradually as necessary, thereby providing acceleration of simulation. The TLM allows the exploration and validation of design alternatives at higher levels of abstraction.

Software engineering technologies and paradigms are actively explored in the Hw/Sw design domain to augment design reuse. Several system level languages such as SystemC and SystemVerilog have some deficiencies that need to be eliminated. For example, these new languages cannot be used in system modeling by both hardware and software designers as they have some limitations in terms of visual descriptions and ease of use at the system level [74]. Some examples of work in this direction are:

- Using UML in conjunction with SystemC as a hardware requirement and description notation [76]-[78].
- Using XML technology for model descriptions [79], as storage format [79], or for description of meta-data for IP documenting [82].

Timing analysis and verification are important parts of system design. Accelerating these activities can drastically speed up the overall design process.

In this work we present a model of timing specification that can be used for acceleration of design space exploration in the Transaction Level Modeling design flow enhancing in this way the most promising simulation technique by analytical methods. The analytic timing specification representation is completely independent from description/modeling languages; this opens another possibility in terms of reuse. The analytical temporal model can be integrated in different design methodologies.

In the current design methodologies it is impossible to decouple the system model from the details imposed by the design framework. There are many aspects that influence the transformation of the initial specification into implementation thus making it non reusable in its intermediate forms by other design methodologies. The aspects, which are used in the design methodology to express the system model at different abstraction levels, are: design flow structure, choice of the specification language, computational models and paradigms. We propose a method permitting to achieve the orthogonalization of different modeling concepts providing in this manner better possibilities of reuse. The method is based on using the XML technology and the .NET Framework as a design environment for system specification.

The .NET Framework has been designed to ease application development in the highly distributed and heterogeneous environment of the Internet, something which assures powerful multi-paradigm support. In the software domain, the .NET Framework is becoming more and more a dominating design environment. These two facts make choosing the .NET Framework as a development platform very suitable and perhaps one of the best currently available solutions. Using .NET design

capabilities such as XML support, interoperability, .NET Framework class library, object-oriented programming, provides new possibilities in system exploration and eases and accelerates Hw/Sw system codesign.

1.3 Contributions

Our contributions in this work are:

- Proposition of a timing expression methodology in TLM flow. Currently, there is no methodology for timing specification representation in TLM. We propose to use a combination of two paradigms to accelerate the design process: TLM on one hand and a methodology to express timing between different transactions on the other hand. Using a timing specification model and underlying timing constraints verification algorithms can decrease the time needed for verification by simulation [65, 66].
- Proposition of methods for communication structure exploration and refinement in system design based on temporal constraints analysis. The proposed communication structure exploration methodology can be used in an automatic protocol generation, in determining temporal specification inconsistencies and in adjusting some parameters in the case of platform-based design methodologies [67]
- Presentation of a new algorithm with an optimization technique for timing verification of systems with deterministic and non-repetitive behavior handling all constraint types described in the literature and using the linearization of min-max inequalities [**Erreur ! Source du renvoi introuvable.**].
- Proposition of a specification/modeling methodology that supports transactional level modeling in the design cycle in order to handle the increasing complexity of systems; closing the gap between system and RTL

levels; speeding up the simulation process, providing in this manner a methodology that increases the design productivity[62].

- Development of a methodology where the specification is not committed to a hardware description language (HDL) nor to a specific programming language. This will allow reuse at different design stages, easier design exploration and exchange of specification between different users having different HDLs and implementation environments.
- Validation of the intermediate models' structures at each abstraction level in order to assure correctness of the modeling. As correctness of design tools is a necessity in modern conditions, the possibility to guide the designer in the designing process through multiple abstraction levels respecting several modeling rules guarantees error reduction in system design.
- Combination of the transaction level modeling paradigm with software engineering technologies in order to provide an efficient method of separation of concerns. Using software engineering technologies in TLM expression eases the creation of high level transaction models and provides a method to productively explore some architectural concerns at a high level of abstraction [63].

1.4 Outline of the Thesis

The thesis is organized as follows: Chapter 2 presents an overview of existing design techniques underlining the diversity of different aspects differentiating them. It covers the existing design flow structure and the computation models used in specification languages and in application modeling. This review serves to demonstrate the enormous influence of the chosen methodology on intermediate representations of the modeling system. Particularly, attention is given to the presentation and evolution of the transaction level modeling paradigm.

Chapter 3 describes current software engineering technologies such as XML/XSLT and .Net paradigms that we use to achieve the separation of some environment related concerns from system representation. An overview of some

Hw/Sw design methodologies that use these software engineering technologies together with a discussion of the similarities and differences between them and the approaches that we have used are also presented in Chapter 3. The TLM Hw/Sw specification and simulation methodology is detailed in Chapter 4. At the beginning of this chapter, we present a description of the abstract model used for application modeling. Further, the XML abstract model representation is given followed by an explanation of methods assuring correct abstract model construction and simulation model generation. Finally, some experimentation results are given and discussed at the end of the chapter.

Chapter 5 covers a timing representation model. At the beginning, related work is presented and discussed. We analyze and summarize the descriptions of all constraint types and establish relationships between several of them. In addition, we present and discuss existing algorithms, introduce a new timing verification algorithm with an optimization technique and demonstrate experimental results. Finally, an application of the presented timing verification methodology in the TLM design flow is given and Chapter 6 concludes the thesis and gives directions for future work.

Chapter 2. Existing design techniques

Design methodology is the key element in product development. Today's systems are often too complex and require a strong methodology for successful product delivery. There exist a number of approaches in hardware and Hw/Sw system modeling. However, all design practices cover the design fully or partly, from system behavior to its detailed implementation, having the same goal: breakdown design complexity and produce better designs in shorter time (Figure 2).

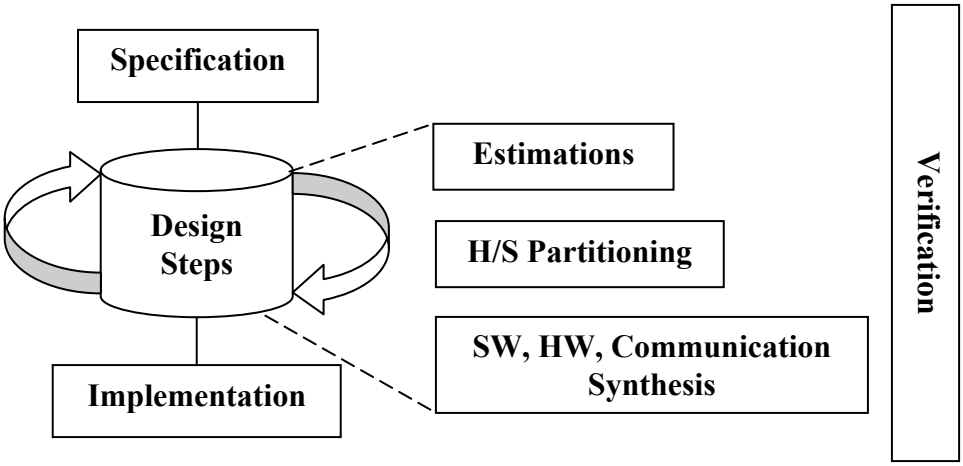

Figure 2: Design process

Specification describes system behavior and non-functional system requirements such as:

- time constraints
- power/energy constraints
- safety requirements
- environmental aspects
- cost, etc.

System specification can be formulated in natural language or using specification languages that make it more detailed and unambiguous. The intended behavior of the designed system can be presented as a relation between inputs and outputs or as an algorithm.

In today's design methodologies it is impossible to decouple the system model from the details imposed by the design framework. There are many aspects that influence the transformation of the initial specification into implementation making it non reusable in its intermediate forms by other design methodologies. These aspects are: design flow structure, choice of the specification language for modeling, computational models and paradigms used in the design methodology to express system model at different abstraction levels.

2.1 Main design flows

There are three general strategies in system modeling based on the design flow structure: top-down, bottom-up and meet-in-the middle.

Top-down strategies, such as that proposed by the System-On-Chip Environment (SCE) [1] start the design flow from system behavior description representing the design's functionality and refine it by adding implementation details until the system gradually reaches the implementation model. In general, system architecture is generated from behavior in these kinds of methodologies.

Bottom-up approaches, such as those used in [5], deal with the existing computation components, which are assembled by means of inserting wrappers among them. Bottom-up strategies focus on component reuse and wrapper generation.

Meet- in-the middle approaches [2] and [3] are a combination of both bottom-up and top-down strategies. In systems developed using the meet-in-the middle approach, system architecture is predefined but the system behavior has to be explored and refined to meet architecture constraints.

2.2 Specification/description languages

The choice of a suitable language for system specification is very important for the design methodology. Generally a trade-off between several criteria such as the expressiveness of the language, the automation capabilities provided by the model

underlying the language as well as the availability of methods and tools supporting the language [34] must be made. Specification/description languages describe the desired functionality of a system in a way that captures system characteristics. The semantics of each language are defined by the underlying model of computation [14]. This model defines the expressiveness of the language, i.e. what kind of systems can be described with the language. Languages with high expressiveness can specify numerous systems with different characteristics. However, they also make formal reasoning and automated synthesis extremely complex and in some cases impossible [14]. Below, we will consider some specification languages used in hardware and Hw/Sw modeling. They can be categorized as follows:

- Programming Languages or system level design languages (SpecC, SystemC, etc.)
    - The SpecC language [20] is defined as an extension of the ANSI-C programming language with the goal of supporting specification and design of digital embedded systems, including hardware and software parts. The SpecC language features concepts essential for embedded system design such as behavioral and structural hierarchies, concurrency, communication, synchronization, state transitions and timing.
    - SystemC [18] is a C++ class library and a simulation kernel that allows the creation of cycle-accurate models and system-level designs. The SystemC class library provides the constructs needed to express hardware timing, concurrency, and reactive behavior that are missing in standard C++.
- Hardware Description Languages (VHDL, Verilog, etc.)
    - VHDL (VHSIC Hardware Description Language) [42] and Verilog [43] are the standard description languages for digital systems. They support very well system description at low levels of abstraction (RTL). System representation at high levels of description is possible but complex and is not widely used by the designers.

- Languages specialized for specification of systems in particular areas and displaying unique features:
    - Formal Description Techniques (LOTOS, SDL, etc.)
        - LOTOS [35] (Language of Temporal Ordering Specifications) is a language that is very suitable in describing concurrency, communication and data structures; however, the concept of time is missing. It is based on Process Algebra. The properties of Process Algebra are used in order to prove correctness of specifications. The language is widely used in protocol and distributed systems specifications.
        - SDL [36] (Specification and Description Language) is a general purpose description language for communicating systems. It is based on the model of extended finite state machines and can be used to model real-time, stimulus-response systems.
    - Real Time System Languages (Esterel, Statecharts)
        - Esterel [48] is a programming language dedicated to programming reactive systems, including real-time systems and control automata. This language provides powerful concepts for expressing time, though the communication model is restricted to the specification of synchronous systems [44].
        - StateCharts [45] is a language that has been developed in order to deal with problems of specification and design of large reactive systems. The basis for the StateChart language is the Hybrid state machine proposed by David Harel. StateCharts is used to depict the behavioral view of a system, which is described by means of hierarchical states with corresponding transitions. Moreover, these transitions are triggered by

conditions and events. The communication model used in StateCharts is broadcasting whereas the execution model is synchronous.

- Parallel Programming Languages (CSP, Occam)
  - CSP (Communicating Sequential Processes language) [46] is Process Algebra designed for the description and analysis of concurrent systems. CSP is based on formal mathematics thereby allowing the designer to specify requirements unambiguously and to satisfyingly prove their implementation.

In the next section, we will consider some of the models of computation commonly used to describe hardware and Hw/Sw systems that have been employed in the above presented specification languages.

2.3 Models of computation

A model of computation in the general sense can be defined as a set of theoretical choices that adequately express some problem. We can distinguish a computational model or models underlying the specification language from computational models that can be constructed on the top of the language.

Any design can be viewed abstractly as a set of components interacting with each other and with their environment. In this context, the model of computation describes the behavior and interaction methods of these components. The aspects that models of computation usually refer to are:

- internal semantics of component functionality related to it computation
- communication
- relationships in terms of concurrency

Concurrency is one of the aspects that differentiates computational models. The concurrency can be

- Data-driven

- Control-driven.

In data-driven concurrency the ordering of executions is not explicitly specified. Parallelism is determined by data dependencies. In control-driven concurrency explicit constructs are used to specify parallel and sequential execution.

Various communication and synchronization mechanisms are used in the different computation models: shared memory, message-passing communications, blocking and non-blocking communications control-depending and data-depending synchronizations. The last very important component of each computation model is a time representation [14].

A variety of models have been proposed for concurrent systems. We will consider some computation models commonly used to describe hardware and Hw/Sw systems:

- Communicating sequential processes (CSP)
- Kahn Process Network
- Dataflow models
- Discrete Event
- Petri nets
- Finite State Machine

2.3.1 Communicating sequential processes [40]

In the CSP computation model, the components are sequential processes that run concurrently and communicate using the synchronous message passing technique. Synchronous communication in CSP means the presence of a mechanism that ensures that, in case of data transfer, the receiver process is in an adequate state to accept the information. The notion of time is absent. Time consuming actions have to be modeled using a pair of events (an event is an atomic action with zero duration). This model of computation is very appropriate to represent applications dealing with resource management problems.

2.3.2 Kahn process networks [8].

In this model, concurrent processes (Figure 3) communicate through unidirectional channels with unbounded capacity using a first-in-first-out (FIFO) policy. A read operation is blocking, i.e. an attempt to read from an empty channel will lead to a process stall. A write operation is non-blocking. Processes perform sequential computation and, at any given time, they may be in two states: a computing state or a waiting for information on one of its input lines state.

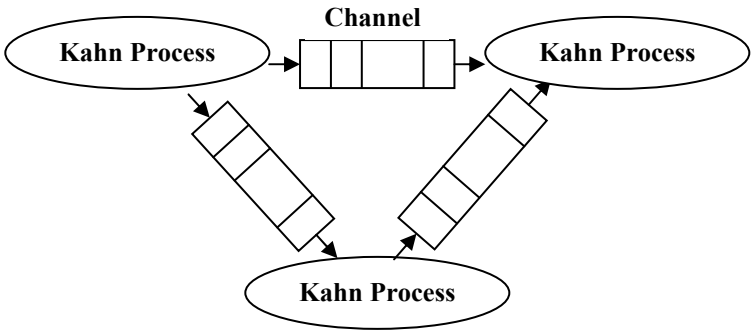

Figure 3: Kahn process network

2.3.3 Dataflow models

- SDF – Synchronous Dataflow [49]. In the SDF model, components execute some actions according to a predetermined schedule, i.e. the modeled system is presented as a directed graph where the nodes describe computations and the arcs data paths. Any node performs its computation (fires) when the input data is available on incoming arcs. When the node fires, several tokens will be consumed or produced on each arc. In the SDF model, the number of tokens on each arc is specified a priori. Communication is strictly controlled and there is no notion of time. This model is very appropriate for representing digital signal processing applications.
- HDF – Heterochronous Dataflow. This is a heterogeneous composition of the SDF and the finite state machine models. The HDF computation model

allows rate changes through state transitions of the FSM, while within each state the system can be considered as an SDF model.

- DT – Discrete Time. This model is an SDF model with the added notion of time. Furthermore, it has a global time and period.

2.3.4 Discrete Event

In discrete event models [50] system components communicate according to the events that are ordered on a global timeline. An event is an instantaneous action which causes a transition from one discrete state to another. The communication between computational tasks (processes) is implemented by means of signals that represent a set of atomic events occurring in some instant of physical time. Thus, each event is associated with a value and a timestamp [47]. Each action can be the event generator or event receiver. This computation model is widely used in digital logic to simulate behavior of digital systems (VHDL, Verilog simulators).

2.3.5 Petri nets

Proposed by C.A. Petri in1966, the Petri nets computation model consists of three elements: the S-elements (places), T-elements (transitions) and tokens. Places and transitions are related to each other by a flow relation. Two important characteristics of Petri nets are concurrency and the asynchronous nature of this model. The asynchronous property signifies that there is no inherent clock mechanism for firing transitions. Petri nets are very useful to represent the control structures of digital systems.

2.3.6 Finite State Machine

The FSM model consists of a set of sequential states linked by transitions. An operation of the system is strictly ordered by a set of corresponding transitions. As the classical FSM representations do not allow concurrency of states, a number of extensions and variations have been suggested:

– Extended-FSM,

– Codesign FSM [51].

2.4 Transaction Level Modeling (TLM)

The TLM modeling approach has been widely discussed in system-level design community as an approach to handle the complexity of system-on-chip and time-to-market pressures [6, 7, 21, 22, 23]. There are many definitions of TLM and many development environments proposing TLM in their design cycle. In this section, we will consider some TLM definitions, the evolution of this modeling paradigm and why it became the first choice for many researchers.

Most definitions denote transaction level models as models where the communication and computation of systems are considered separately. This definition has a vague meaning and there are many types of models that fit this description.

2.4.1 SpecC definition

The TLM definition proposed by L. Cai and D. Gajski [6] is closely connected to the SpecC modeling principles. We have mentioned the SpecC modeling language in subsection 2.2.

TLM in SpecC interpretation defines several transaction level models, each of which is adopted for a different design purpose. These TLM models are:

- Component-assembly model
- Bus-arbitration model
- Bus-functional model
- Cycle-accurate computation model.

The TLM models serve to simplify the design process by slicing the entire design into several smaller design stages. Each design stage has a specific design objective. At each of these design stages, the corresponding models can be simulated and estimated independently. The system modeling graph shown in Figure 4 relates different models. On the graph, the axes represent computation and communication

with a time accuracy of three degrees: untimed, approximate-timed and cycle-timed. The untimed degree signifies computation/communication descriptions without any notion of time and represents the description of only pure functionality. The approximate-timed degree presumes that system level implementation details are added to system descriptions. The models with this degree of time accuracy contain information about the selected system architecture and the specification process mapping to the processing elements of the system architecture. The cycle-timed computation/communication models contain implementation details at system and RTL levels. In the system modeling graph shown in Figure 4, the TLM abstraction models (B, C, D, and E) are shown together with two others, namely the specification and implementation models. This is done to demonstrate the relationships between all models used in design.

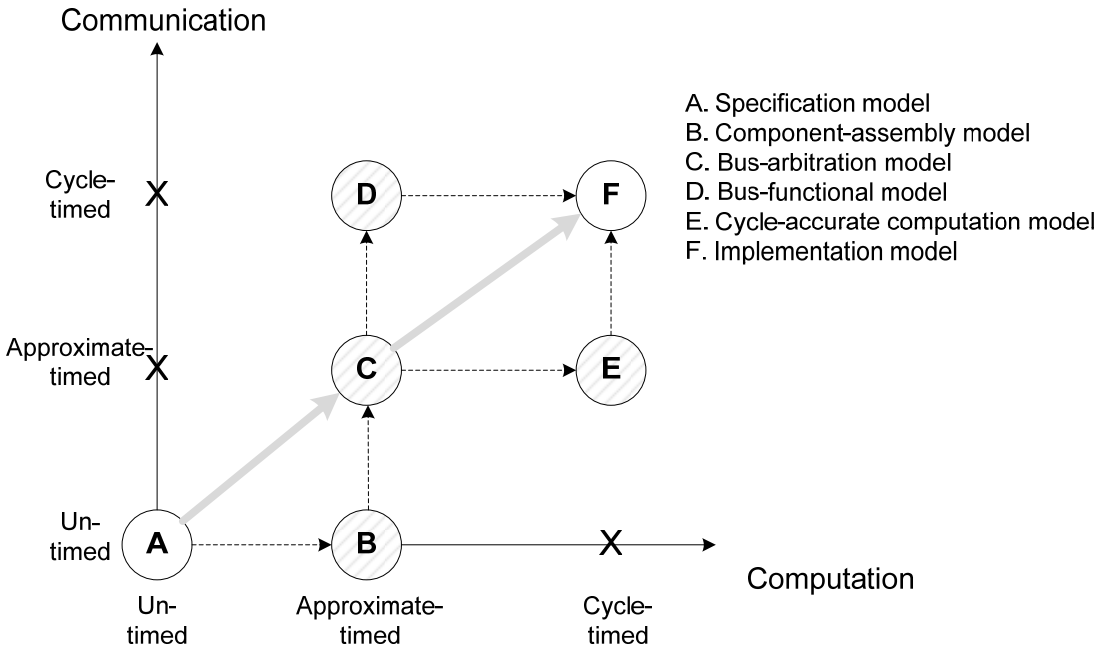

Figure 4: System modeling graph

The specification model is untimed and can describe system functionality without implementation details. The component-assembly model describes concurrently executing processing elements which communicate through channels.

The communication part of the model is untimed while the computation part of the model is timed. The estimated time of computation is computed by a system level estimator. The component-assembly model explicitly specifies the allocated processing elements in the system architecture and the mapping decision of processes to processing elements. In the bus-arbitration model, some bus protocol details are added with approximate timing to the channel message passing mechanism. The bus-functional model contains cycle accurate communication and approximate-timed computation descriptions. Two types of bus-functional models exist: time-accurate and cycle accurate. Time-accurate models specify timing constraints of communication; cycle-accurate models specify the time in terms of the bus master's clock cycles. In the cycle-accurate computation model, the computation is cycle-accurate, i.e. computation components are pin accurate and function cycle-accurately whereas communication is approximate-timed. Finally, the implementation model represents cycle accurate communication and computation. The processing elements are defined in terms of their register-transfer or instruction set architecture.

In Figure 4 a common design flow is indicated by the gray solid arrow going through the specification and bus-arbitration models and finishing with the implementation model. The specification model represents system functionality, the bus arbitration model denotes abstract system architecture and the implementation model deals with cycle-accurate system implementation. The bus-arbitration model (C) divides the system flow in two stages: system design and component design. In the first stage, the system architecture is selected or generated and the system behavior is mapped to that architecture. In the second stage of the design flow, the computation and communication components are refined to a cycle accurate level and possibly synthesized. Different design flows containing different models exist.

2.4.2 SystemC definition

Below, we will consider the SystemC (section 2.2) definition of TLM proposed by A. Donlin [7]. According to [7], TLM refers to a set of abstraction levels each differing from one another in the degree of expression of functional or temporal

details. These levels, and the possible design flows through the TLM space, are presented in Figure 5 together with an indication of their situation between the algorithmic (ALG) and register transfer levels (RTL) that are not considered to be a part of TLM space. Transaction-modeling levels are:

- Communicating Processes (CP)
- Communicating Processes with Time (CP+T)
- Programmer's View (PV)
- Programmer's View with Time (PV+T)
- Cycle Accurate (CA)

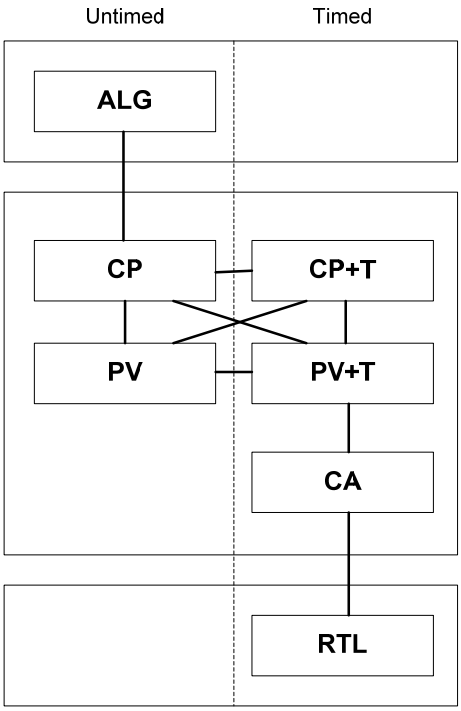

Figure 5: TLM Abstraction Levels and Potential Flows [7]

The Communicating Processes level is characterized by the representation of system behavior as a network of parallel processes that exchange complex, high-level data structures. Processes communicate using point-to-point links. Systems described at this level are generally architecture and implementation independent. However, the separation of functionality into parallel tasks imposes some architectural concerns. The main activity at this abstraction level is functional verification (Figure 6).

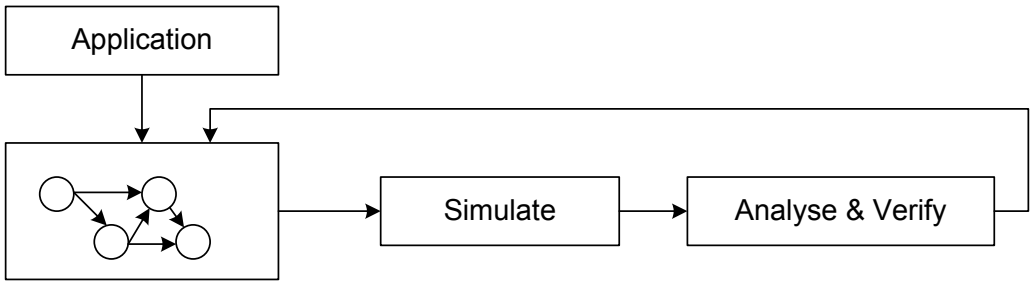

Figure 6: CP level

At the **Communicating Processes with Time level**, some timing information has to be added. Nodes may contain quite accurate or high-level estimates of timing data. Communication timing models are abstract and the exact communication protocol is not defined. The main design activity is design space exploration (Figure 7).

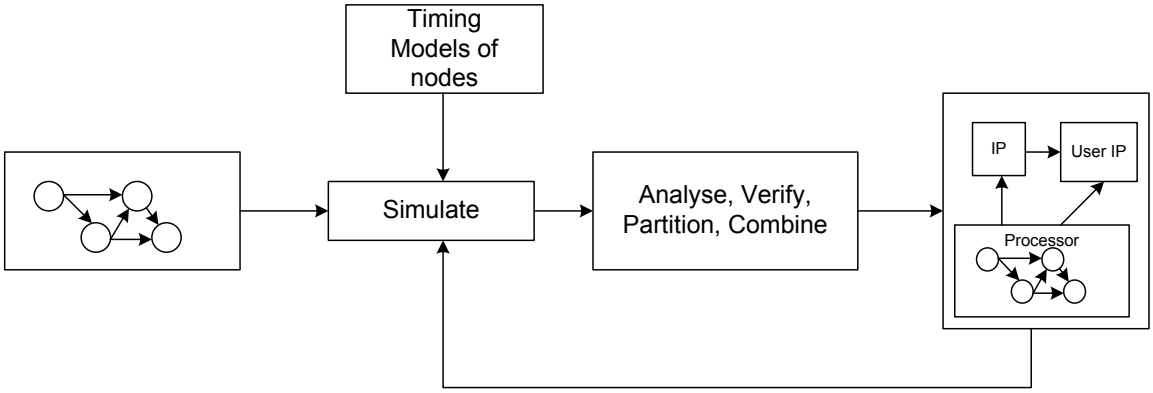

Figure 7: CP+T level

The Programmer's View level defines a transport mechanism between model components with some elements of arbitration. The hardware sub-system at this level may be seen as an accurate programmer's representation by low-level software drivers.

The Programmer's View with Time level is functionally identical to the PV level model with the exception of timing information. The timing model is much more precise. The abstract communication structure is transformed to a given interconnection type and soft-real-time functional verification is enabled (Figure 8).

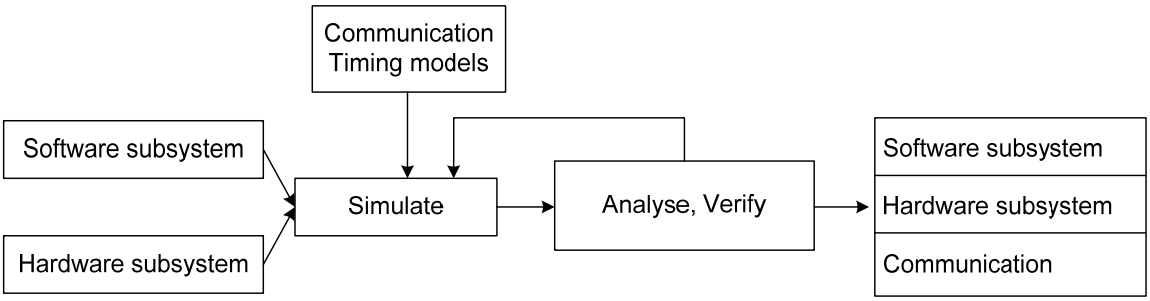

Figure 8: PV, PV+T levels

The Cycle Accurate model contains micro-architectural details. It has all timing annotations and is accurate to the level of individual clock cycles.

According to the features and requirements of particular product-types the author [7] distinguishes five TLM use-models which are suitable for different application domains. Each use model will respect a design flow which is a subset of levels and paths given in the Figure 5:

- Use-Model 1 (UM1) is a model of ASSP-based (Application-Specific Standard Products) systems. The basic characteristic of such systems is their pre-ordained architecture. A designer has to implement an application on a given platform, i.e. in software destined to execute on the ASSP. As UM1 is characterized by its software dominated nature, CP and CP+T levels are not applied. PV and PV+T levels support hardware platform parameter tuning and functional verification. The CA level enables detailed performance analysis of the application using elaborate platform parameter tuning.
- Use-Model 2 (UM2) describes systems based on structured ASIC (Library-oriented micro-architecture design). In these systems, a hardware subsystem is a result of an incremental refinement of an existing reference architecture specification. Customization of the subsystem specification in these models is limited by the next type of variations:
    - IP module replacements
    - Addition of some small number pieces to IP
    - Alternation of the IP provision subset.

In UM2, the CP level captures the application behavior description in an architecture agnostic manner. The CP+T level is not used. The PV level enables software development and functional verification whereas PV+T and CA support the same activities as in UM1.

- Use-Model 3 (UM3) is a model from the Structured ASICs domain with a much greater degree of hardware subsystem customization than UM2. This allows satisfying design performance goals, functional goals or both. In this model, the designer works with a reference hardware architecture and with the ability to add custom-designed IP to the interconnection architecture of the system. In this model, CP, PV, PV+T and CA levels support almost the same activities as in UM2. The increasing degree of customization in UM3 introduces some design space exploration presented at the CP+T level by means of high level performance estimation and partitioning of application functionality.
- Use-Model 4 (UM4) is based on custom designed ASIC. In this model, there is no initial architecture template; some vendor supplied component models are still present. Comparing with UM3, much more application functionality is partitioned into the hardware domain with the aim of realizing it as a custom logic. The design activity in UM4 is equivalent to that in UM3. There is one difference, however, in the shifting of emphasis from PV, PV+T to CP and CP+T levels.
- Finally, Use-Model 5 (UM5) presents a model that allows customization of all system aspects to achieve maximum performance and functionality. In this model design space exploration plays an extremely important role. All abstraction levels are presented in UM5, but the most significant work is concentrated in the CP and CP+T levels.

2.4.3 Open Core Protocol – International Partnership TLM definition

The Open Core Protocol – International Partnership (OCP-IP) TLM defines a set of layers of abstraction that together create a link between architecture exploration and System on chip (SoC) implementation. These layers are: Message Layer (L-3), Transaction Layer (L-2) and Transfer Layer (L-1) [58].

The Message Layer or Layer 3 is the highest level of abstraction which can be used by SoC architects to prove concept tools, to rationalize first order functional partitioning and to explore some system level architectural concerns. This layer is also one of executable specifications. L-3 models are untimed and event driven.

The transaction Layer or Layer 2 serves to make detailed hardware performance analysis and hardware/software partitioning. At this level, low level drivers can be interfaced with hardware simulation models and Operating System simulators can be integrated with hardware emulators. L-2 models are structurally accurate enough to allow modeling a complete system. An event-driven computational model contains approximate timing and is highly parametrical. Computation models at the L-2 level are independent of any bus fabric protocols.

The transfer Layer or Layer 1 is used by designers to perform detailed tasks such as modeling the interfaces of embedded processors, creating cycle accurate test benches and carrying out cycle accurate performance simulations. L-1 models are clocked cycle-accurate.

2.4.4 TLM evolution

We have only presented some TLM definitions from the myriad available. As we can see, the necessity of standards in TLM was urgent [59] in order to provide a possibility for model exchange within companies and across IP producers. Some attempt has been done early trying to link the Open SystemC Initiative (OSCI) and OCP-IP TLM world [58]. Furthermore, the initial SystemC definition [7] has been revised and OSCI TLM levels have become [58]:

- Programmer's View (PV)

- Programmer's View with Timing (PV+T)
- Cycle Callable (CC).

TLM abstracts away the number of events and the amount of information that has to be processed during simulation of the minimum required. In SystemC, the TLM definition of the necessary information is presented to the designer as a TLM API (Application Program Interface). The OSCI TLM API represents a set of interfaces that define how models communicate. At PV level, an interface does not contain communication events, carries little timing information and is implemented as a function call. At PV+T level, the simulation can switch between two interfaces with and without timing. This level is characterized by allowing model refinement without changing the functional description of the model's behavior. The CC level provides a cycle accurate modeling style. The interface explicitly describes the cycle-by-cycle behavior. As the behavior of the model is coupled with the interface, it includes cycle timing. The CC level uses higher level ports rather than pin-accurate signals while still remaining at TLM abstraction.

Recently, a new transaction level modeling standard, TLM-2.0, was announced by OSCI [19]. Standard transaction level modeling approaches aim to enable model interoperability and exchange within companies and between companies. The new standard includes some changes. Models have been categorized according several characteristics such as granularity of time, frequency of model evaluation, functional abstraction, communication abstraction and use cases. The existence of a variety of use cases for transaction level modeling is explicitly recognized and TLM-2.0 uses an approach of distinguishing between APIs on one hand and coding styles on the other rather than defining an abstraction level around each use case as shown in Figure 9. The TLM-2.0 standard defines a set of interfaces and describes a number of coding styles for various use cases. The interfaces are low-level programming mechanisms for implementing transaction level models and form the normative part of the standard thereby ensuring interoperability. A coding style is defined as "a set of programming language idioms that work well together, not a specific abstraction

level or software programming interface" [81]. Each coding style supports a range of abstraction functionality, communication and timing.

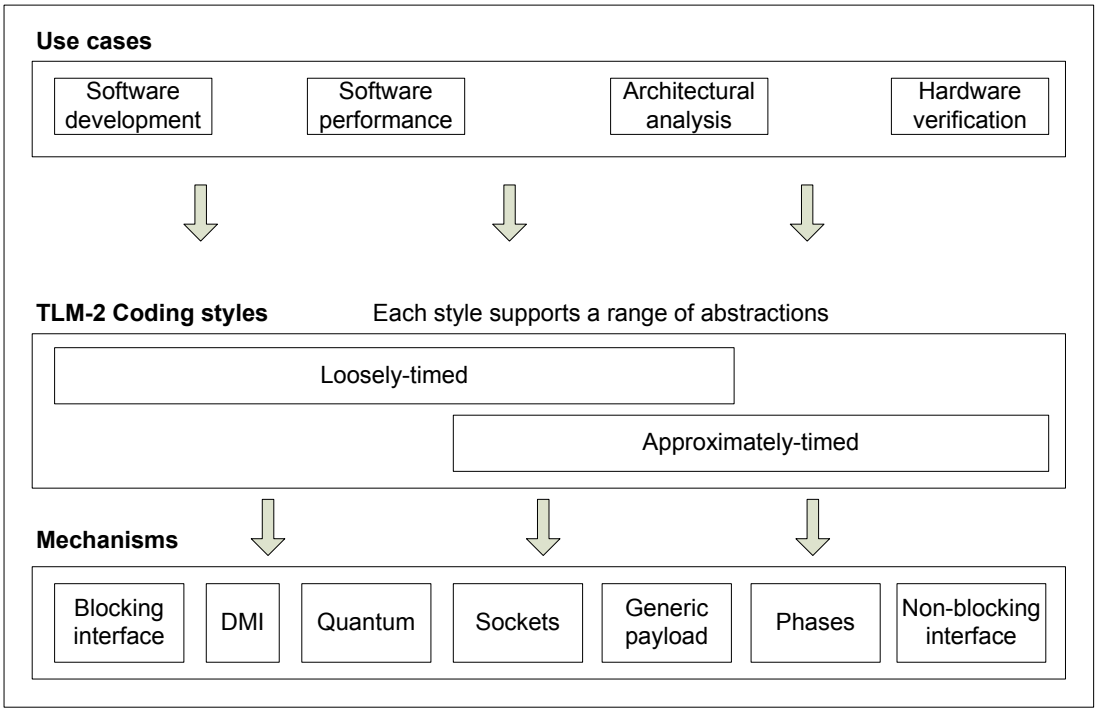

Figure 9: TLM-2.0 approach [81]

In our work, we have chosen the SystemC terminology for TLM as the SystemC modeling language was in its standardization process. Now, the SystemC standardization process has finished with several changes introduced in the new SystemC TLM definition. In our work, we referred to the terminology initially proposed in [7].

2.5 Discussion

There are many of codesign methodologies that use different computational models in system modeling in order to simplify the design process of complex systems [13, 14, 17, 18, 19]. All modeling elements add to the system description a multitude of details which are relevant to the particular environment thus allowing the intermediate system representations to be tightly coupled with the design environment. In Table I, we summarize different design methodology aspects which

influence the transformation process of specification into implementation for some codesign methodologies (the presented list is not exhaustive).

Table I: Codesign methodologies

Codesign Environment	**Application**	**Specification language**	**Model**
Vulcan [52]	Data oriented	HardwareC	DFG Set
Cosyma	Data oriented	C^x	Syntactic tree
Ptolemy [18]	Real Time	Silage	CDFG
Polis [17]	Control-dominated systems	Esterel	CFSM :Codesign Finite State Machines
Cosmos	Control-dominated systems	SDL	SOLAR: communicating extended FSM
CoWare [4], SPACE [85]	Various	SystemC	Discrete Event, TLM
SPADE [57]	Signal processing applications	C	Kahn process networks
SpecSyn [54]	Different	SpecCharts	Hierarchical program state machine (PSM)
Tosca [53]	Control-dominated systems	OCCAM	CSP
Chinook [55]	Control-dominated systems	Verilog	Modal processes

All these methodologies use one specification/description language in their design cycle. Different computation models provided by some of these design methodologies are constructed on the top of specification/description language related to the methodology. Utilization of a restricted computation model has the advantage that a system might be modeled in an unambiguous way and formal verification techniques might be applied. A disadvantage consists of a restriction of the represented application class. For the methodologies that support quite general computation models, verification by simulation is the only viable solution to test the model correctness. Formal verification techniques may be used to verify only a part of the design. In all cases, the process transforming a specification into

implementation is highly related to the methodology. There exists a multitude of methodologies, such as CoWare [4] for example, that support reuse at the component or module level. It should be noted, however, that we are talking about a finer level of reuse here, which is that of functional descriptions used inside a module or component for different design methodologies. These intermediate representations of a system under design are impossible or difficult to reuse as the computation model and the specification/description language choice influence the model description enormously.

Chapter 3. Software engineering technologies

The application of software engineering methodologies to hardware and Hw/Sw design is an active field of research within the hardware design community. In our research, we have used modern software engineering technologies such as XML, XSLT and other software engineering paradigms in order to represent the system model at different abstraction levels with a clear separation from the design's environment details. This approach leads to better model reuse and "portability".

3.1 XML technology [9]

Extensible Markup Language, XML [9], – is a markup language designed to provide a standard way to describe content.

Markup languages are characterized by:

- Description of the text structure within the document.
- Separation of content from formatting.

Generalized Markup Languages are HTML (Hypertext Markup Language), SGML (Standard Generalized Murkup Language) and XML.

HTML, contrary to SGML and XML, is technically a markup language that in reality is used as a formatting language. In HTML, content and representation are defined together within the same document.

SGML is a very powerful markup language that is widely used to handle complex, large documents across platforms. SGML's complexity and a limited set of elements for structuring documents in HTML format created the need for XML creation.

XML is a subset of SGML. It provides many of SGML's complex features but in a more manageable form. XML uses element tags to mark up content according to a set of rules created by the document's developer called the Document Type Definition (DTD).

XML provides the ability to

- define the new elements and attributes,
- nest document structures within other document structures,
- check the validity of document structure.

With XML, the layout is separate from the content. The mechanism of style sheets is used to drive the layout in order to display the content across different applications. XSL, Extensible Stylecheet Language, is a language that implements the style sheets mechanism for XML. XSL consists of two parts: one is composed of some formatting features and the other of XSLT. XSLT (XSL transformations) describes syntax for transforming a document from XML format to another. In the next subsection, we will give a more detailed description of the role of XSLT in XML treatment.

3.2 XML processing

Like many formatting languages, XML requires parsers and processors in order to adequately convert the incorporated content. Parsers currently used for XML usually take the form of a code library written in programming languages. The parser verifies the syntax of the DTD and XML document and then the processor provides access to the content and structure of the document.

There are three different approaches to accessing an XML document in a program:

- The Document Object Model (DOM),
- The Simple Application Programming Interface (API) for XML (SAX),
- The Extensible Stylesheet Language Transformations (XSLT) approach.

In the DOM approach, the entire XML document is placed in memory as a hierarchical “tree” and the programmer has the possibility to apply various methods to locate and manipulate the nodes of the tree.

In the SAX approach, a parser analyzes the XML document, identifies each element as it is encountered and calls methods supplied by the programmer as the document is read.

The XSLT approach, as we have discussed in the previous subsection, was initially a composite part of XSL. However, it was quickly discovered that XSLT abilities to reorganize document structure had far greater use outside of XSL. The XSLT have become its own recommendation.

An XSLT document contains the transformation rules that can be applied to the source document tree. The rules are presented as a collection of patterns and templates. When the pattern is matched, its content is used to fill in the XSLT template. The resulting tree contains filled templates with the information gathered from the source tree. A transformation process using XSLT means independence of source tree organization from the resulting one structure. The XSLT approach is widely used in XML processing. It provides a "declarative" style of programming that is different from the procedural programming of DOM and SAX. The characteristics of three program processing approaches are demonstrated in Table II.

Table II: Decision matrix for selecting an XML processing approach [9]

Criterion/Capability	DOM	SAX	XSLT
Document size	Small to medium	Any	Any
Access multiple elements at the same time	Easy	Tricky	Possible
Rearrange elements	Yes	No	Yes
Create a new document	Yes	No	Yes
Modify an existing document	Yes	Tricky but possible	Yes

3.3 XSD technology [20]

The XML Schema (XSD) is a schema definition language expressed in XML which is intended to be used to describe structure and to constrain the content of XML documents [20]. The XML Schema definition language is the current standard schema language for all XML documents and data. XSD improves on the schema

functionality provided by the DTD, which was the original form of schema for XML documents prior to this technology.

A schema specifies the rules that complying XML documents have to follow to be considered valid. Validation of XML documents ensures that external data conforms to the rules defined by the schema, in this manner providing possibilities to exchange information between applications with greater confidence and with less custom programming to test or to confirm document structure or that some data is of a particular type.

3.4 .NET

The Microsoft .NET Framework is a software technology that was intended to simplify application development in the highly distributed environment of the Internet [84]. This objective has to be reached by means of providing multiparadigm support. The .NET Framework includes a large library of precoded solutions to common programming problems termed the .NET Framework class library. It includes programming solutions covering a number of areas. In addition, it also provides functionalities for web application development, XML document manipulation, cryptography, network communication, data access, database connectivity, several numeric algorithms, data structures and user interface. The Common Language Runtime (CLR) acts as an application virtual machine providing capabilities of independency from the specific CPU that will execute the program. The common language runtime manages code at execution time, provides core services such as memory management, thread management, remote execution and enforces strict type safety and other forms of code accuracy. Programming languages on the .NET Framework compile into an intermediate language, the Common Intermediate Language (CIL), which is then compiled into native code in a manner known as just-in-time compilation (JIT). The common language runtime provides built-in support for language interoperability, in other words it allows for the ability of code to interact with code written in different programming languages by means of specifying and enforcing a common type system and by providing metadata.

Language interoperability maximizes code reuse and in this way improves the efficiency of the development process. All .NET programming languages targeting language interoperability follow the rules for defining and using types which is consistent across languages. Metadata defines a uniform mechanism for storing and retrieving information about types thereby providing descriptive information about the context, condition of data or some data characteristics. Furthermore, several elements of the .Net Framework are available as open standards thus creating possibilities for third parties to develop compatible implementations of the framework on other platforms leading to .NET Framework portability.

3.5 Hw/Sw design and software engineering technologies and paradigms

Software engineering technologies and paradigms are actively explored in the Hw/Sw design domain to increase design productivity and to breakdown the growing system complexity by raising the level of abstraction. Several system level languages have been recently proposed to bridge the gap between system modeling and RTL levels, though these languages have some limitations in terms of visual descriptions and ease of use at the system level [74]. In order to resolve this problem, several researchers have proposed to use different software engineering technologies and paradigms in hardware and Hw/Sw design. In this subsection, we will focus on the review of several works that have used XML technology for Hw/Sw design, as we did in ours.

3.5.1 Sesame framework [79]

C. Erbas and al. [79] have proposed a framework named Sesame for the system-level modeling and simulation of embedded system architectures. It primarily focuses on the multimedia application domain to efficiently prune and explore the design space of the target platform architectures. Sesame recognizes two distinct models: an application model, describing the functional behavior of an application, and a platform architecture model that defines architecture resources and captures

their performance constraints. For application modeling, Sesame uses the Kahn process network model of computation. During the execution of the Kahn application model, each process records its computational and communication actions, such as *read*s or *execute*s, thus generating a trace of application events of coarse grain. An architecture model simulates the performance consequences of the communication and computation events by parameterization of each component of the architecture model. In order to cosimulate the application and architecture models, an intermediate mapping layer is used. The mapping layer executes three functions: control of the mapping of Kahn processes onto architecture model components, makes sure that no communication deadlocks occur by providing various strategies for application event scheduling and dynamically transforms application events into low-level architecture events to realize flexible refinement of architecture models. The results of system simulations are performance estimates of the system under study with some statistical information. To be capable of exploring large parts of the design space, Sesame uses analytical methods to identify a small set of promising system architectures for system exploration by simulation. The mapping of an application model onto an architectural model takes into account three objectives: maximum processing time, total power consumption and the cost of the architecture. The mapping is effectuated using multiobjective evolutionary algorithms.

Sesame's model description language, which is used to describe the application model, the architecture model and the mapping which relates two models for cosimulation, is called YML (Y-chart modeling language) and is XML-based. The YML language contains several elements and describes simulation models as directed graphs. Sesame's application simulator reads an YML application description file and executes the application model. The object code of each process is fetched from a shared library; currently C++ processes are supported. The architecture models are implemented in the Pearl discrete event simulation language or in SCPEx, which is a variant of Pearl implemented on top of SystemC.

3.5.2 Colif [79]

Colif is a design representation for modeling on-chip communication at different abstraction levels where component behavior is separated from the communication infrastructure. In this design representation, system specification is presented in a modular manner using the concept of object model and defining its semantics at four abstraction levels of communication refinement. In a modular system specification, three conceptual entities are used: module, port and net. A system in Colif is presented as a set of hierarchical modules interconnected by a communication network which in turn can be composed of hierarchical nets and ports. This syntactical representation is uniform and can be used to describe heterogeneous systems. Each module is defined by its interface consisting of a set of ports and content. The module content can be other module instances or a composition of tasks. The Colif object model has a declarative part and an instances part. The declarations represent objects that define a reusable template. The classes describing Colif objects are polymorphic and their semantics change according to the abstraction level that is considered. The interface between components is implemented using the port concept. These are classified into two categories: internal and external. This allows for providing a mechanism to mix different abstraction levels within the same description. Furthermore, this concept permits the separation of behavior and communication. Different port instances are connected using the net object. The net object can hide very complex behavior. The flexibility of the Colif representation resides in the permission of the hierarchical structure of modules and in the generalized net and port concepts.

The format used for Colif representation is XML. Arbitrary complex object-data models can be created using a special XML dialect called Middle-ML grammar. In the later stages, a special design tool generates a cosimulation model.

3.5.3 IP-XACT [82]

The IP-XACT design-exchange standard was developed by the SPIRIT (Structure for Packaging, Integrating and Re-using IP within Tool-flows) consortium [82], an organization that was formed to resolve the need for an integrated front-end,

multi-vendor system design flow in the semiconductor industry. Several companies signed up for the initiative as the design of complex systems-on-chip was getting significantly harder, and building an IP-reuse solution to this problem required an improved integration of various design flows and better interoperability of reusable IP.

The IP-XACT specification is deliverable for accompanying IP design files written in different popular design languages such as Verilog, VHDL, SystemVerilog, SystemC, etc. It is built on W3C standards to allow capture of metadata that is used across multiple platforms and applications as shown in Figure 10.

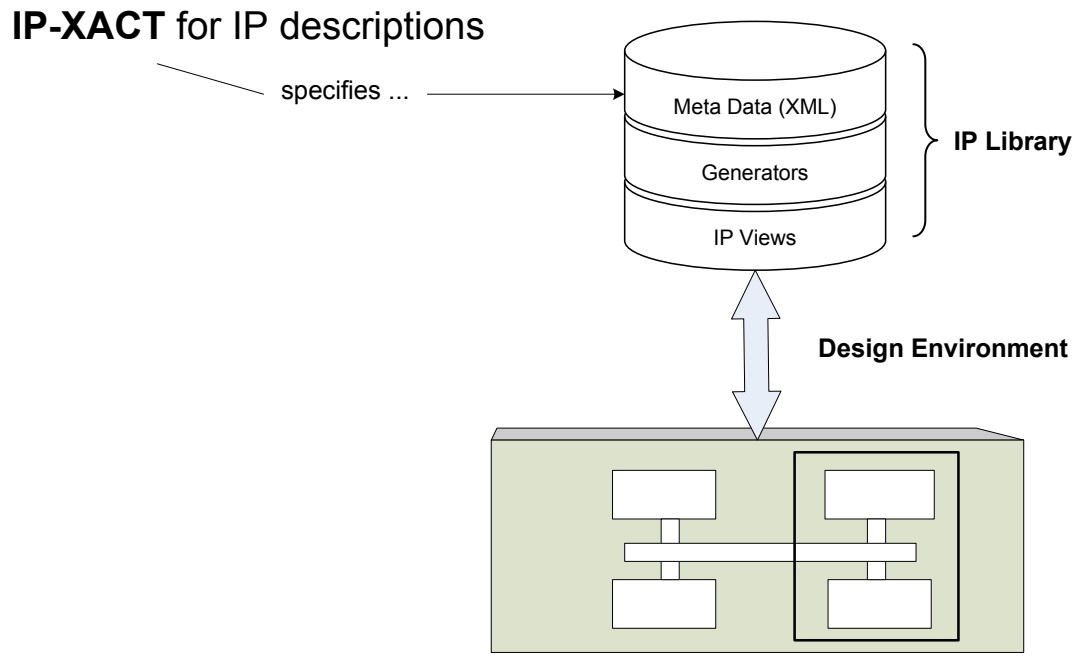

Figure 10: IP-XACT specification [82]

IP-XACT describes soft IP, hard IP, verification IP and software. The specification documents the software and hardware views of a design, the interface of an IP block and the interfaces to standard and custom busses. This allows system design tools to automatically comprehend IP integration requirements. IP-XACT covers two aspects of IP integration. It includes a XML schema that creates a

common way to describe IP and a tool integration API. The API provides a standard method for tools to exchange design data.

3.6 Discussion

In this section, we have overviewed Hw/Sw design methodologies using XML technology in the design cycle. In the Sesame and Colif approaches, XML technology is used in a similar way: an XML-based description language is created for model descriptions. In both cases XML descriptions facilitate internal manipulation of modules. The SPIRIT IP-XACT specification together with its associated tools was created with the aim of increasing IP reuse.

In the following section, we propose a TLM modeling and simulation methodology that uses XML technology to increase the possibility of reuse. It should be noted that reuse is considered at different degrees of granularity that that seen in the SPIRIT approach. In our methodology, the XML document structure has to reflect the computational model chosen for modeling and the .NET interoperability features have to provide a support of several specification/description languages that can be incorporated in a system specification. Similarly to the SPIRIT approach, our methodology uses XSD schema technology to validate XML specifications as well as XSLT transformations to generate the executable system models against “SPIRIT” generators that enable automated design creation and configuration.

Chapter 4. TLM Hw/Sw system specification and modeling methodology

The TLM paradigm permits a more efficient system exploration and an increased simulation speed than traditional RTL design. The possibility of easy and efficient modeling at TLM levels can reduce the development cycle and lead to accurate implementations. These reasons are at the root of TLM modeling becoming very popular and widely adopted in the system level community. When we started our work, there was no accepted methodology for timing specification representation in TLM. The existing TLM methodologies (Section 2.4) were tightly coupled with specification/description languages rendering the integration process of timing specification methodology difficult. They provided reuse at component level and not at "code" level where code segments written in different specification/description languages might be merged and simulated together. Recently, works [89] realized in the context of the MARTE [87] and AADL [88] projects have appeared providing methods of temporal analysis following the TML 2 standard [81]. MARTE, a UML profile for Modeling and Analysis of Real-Time and Embedded Systems, and AADL, an Architecture Analysis and Design Language, are both modeling formalisms that support the analysis of real-time embedded systems. In this work, the authors propose a new methodology for SoC/SoPC applications where the separation of concerns is supported by the MARTE modeling formalism and the description of communications follows the TLM 2 standard. This standard does not include the CP and CP+T levels, but we think that system refinement should be done through multiple abstraction levels, starting from very high abstraction description towards the detailed RTL description. Furthermore, in the MARTE formalism, several models of computation and communication used in embedded applications are absent [89], leading to a limitation of the modeled applications. As the MARTE/AADL approach was unavailable when we tried to introduce the TLM timing specification methodology and any existing TLM methodology had not provided a sufficient separation of concerns, we have decided to define a TLM modeling and simulation methodology in which we can clearly separate several modeling aspects such as

timing, behavior, structure, specification/description languages, simulator details, etc. To achieve the orthogonalization of modeling concepts, the methodology presented below uses software engineering technologies and paradigms such as XML and .Net.

The next section will detail the TLM Hw/Sw system specification and modeling methodology.

4.1 Abstract model

The first abstraction level in Transaction Level Modeling is the Communicating processes level, where the system is modeled as a network of parallel communicating processes exchanging complex, high level data structures. At the CP level, we have proposed to use an abstract model for system specification based on the Kahn process network [8]. In this model, processes communicate through unbounded FIFOs and it should be mentioned that the Kahn process network model matches the TLM CP level description. However, we propose to extend the communication mechanism of the Kahn process network model by adding event based inter-process communication and shared memory communication. With this generalization of the Kahn process network computation model, we lose some of its formal properties, though we have extended a class of applications that was explored in our methodology. The pure Kahn process network computation model is a particular case of a generalized model where event-based inter-process communication and shared memory communication mechanism are absent.

In the proposed abstract model, system functionality has to be distributed between parallel processes (Figure 11). The initial parallelism granularity has to be defined by the designer. Each process performs sequential computations locally. The computation actions can be interleaved with communication actions. Furthermore, the extraction of different level of concurrency is limited to the number of independent actions that can be defined by a designer. The designer does not get support from the methodology in identifying these actions.

The communication scheme in the proposed model allows not only data-dependent, but also control-dependent synchronization. The data-dependent synchronization is modeled using FIFOs channels with unbounded capacity like in the Kahn process network model (rectangles in the Figure 11). The control-dependent synchronization is implemented by means of events (dashed lines in the Figure 11).

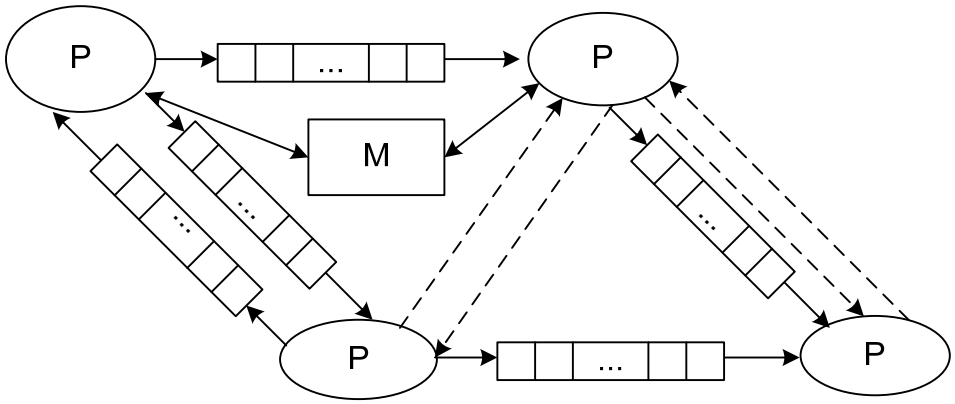

Figure 11: CP Abstract Model

A transaction in our model refers to the exchange of data or an event occurring between two processes of a modeled system. By generalizing the Kahn process network model, we achieve the possibility of enhancing expressiveness for data dominated as well as control dominated systems.

At the CP level, we construct an untimed system model which includes the correct ordering of events. The partitioning of functionality between the communicating nodes has to be done by the designer.

The same abstract model is used in test bench modeling which differs from system modeling by the introduction of an initial timing annotation of processes in order to provide more realistic scenarios for simulation.

4.2 XML Abstract model representation

In order to express the CP abstract model, we have chosen the XML technology [9]. Extensible Markup Language (XML) provides a way to describe structured data using a set of tags. Since we can define an unlimited set of XML tags,

descriptions in this language are very flexible. Furthermore, XML is platform-independent and is a widely adopted standard, therefore making the retrieving of information from XML documents possible in a variety of environments.

We have defined several XML tags to describe the CP Abstract Model. A process' description and its communication scheme are placed in an XML specification. Process functionality is described using one of several programming languages supported by .NET thereby making possible the reuse of functional specifications written in different programming languages. This feature is supported by the .NET language interoperability capabilities.

System specification consists of a functional description of the system and a set of design constraints. In order to unify the functional and non-functional aspects of the system under development, the proposed methodology presents the system as an XML tree of nodes containing functional and non-functional specifications. The initial system specification at the first XML tree level contains the following nodes: *Environment Model*, *System Model*, *Simulation Parameters*, *Shared Code*, *Shared Data and Constraints*. These are depicted in Figure 12 below.

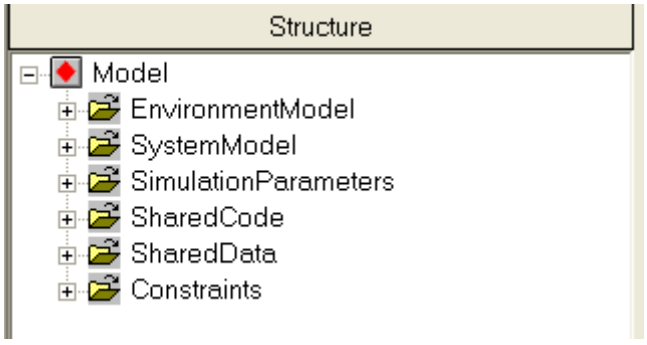

Figure 12: Graphic representation of the XML system specification

The structure of the *Environment Model* node is identical to that of the *System Model* node. The only difference between these nodes is the timing annotation of processes in the *Environment Model* node at the untimed CP level. This allows for generation of valid test-vectors for simulation.

The *System Model* node's structure is presented in the Figure 13 and consists from one or several *Process* nodes and a *Process Network* node.

Figure 13: Graphic representation of the System Model node structure

The *Process* nodes describe parts of system functionality which are/can be performed in parallel. As processes can communicate using either FIFOs, shared memory or through events, the corresponding information has to be presented in the process description shown in Figure 14.

In order to specify the system at the CP level, the designer has to fill in the corresponding tags for each process in the XML specification indicating the chosen transaction mechanism (FIFO, events or shared memory).

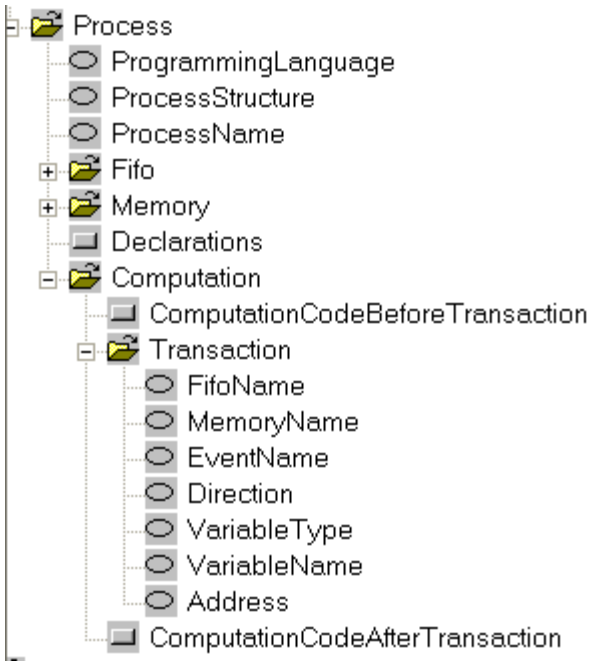

Figure 14: Process node structure

Figure 15 illustrates the use of XML tags for process description. The attribute `ProgrammingLanguage` defines the specification language used for expressing process functionality in the *Computation* node. Each process can be

simple or hierarchical (contains other processes); this is indicated using the *ProcessStructure* attribute.

```
<Process ProgrammingLanguage="" ProcessStructure="simple"
ProcessName="">
<Fifo FifoName="" />
<Memory MemoryType="" MemoryName="" />
<Declarations />
<Computation>
<ComputationCodeBeforeTransaction />
<Transaction FifoName="" MemoryName="" EventName="" Direction=""
VariableType="" VariableName="" Address="" />
<ComputationCodeAfterTransaction />
</Computation>
</Process>
```

Figure 15: XML tags for process description

By default, the process structure is simple. The *ProcessName* tag contains a name of a defining process. As each process can communicate through FIFO, shared memory or by means of events, the corresponding instance names for FIFO and shared memory have to be indicated in the *FIFO* and *Memory* nodes. Computations can be interleaved with communication actions in processes; for this reason there may be one or several *FIFO, Memory* nodes. The *Declarations* node contains declarations of local variables. Finally, each process has one or several *Computation* elements consisting of *Transaction* and *ComputationBefore(After)TransactionCode* nodes.

4.3 Verification of the model structure

Using XML in system description, we have tried to separate multiple modeling elements related to the modeling environment, the specification languages and the simulator. Another advantage of using XML in system description is the possibility of validating the fact that the XML document contains the desired data and structure by means of the XML Schema technology [20]. The validation of model structure at each abstraction level will guarantee the appearance of not many errors during complex system design.

In the presented Hw/Sw simulation and modeling methodology, the XML specification structure at CP level reflects the CP abstract model. The XML Schema technology permits to verify whether the system description conforms to the abstract model and if the XML nodes contain the data of a particular type and a given structure. According to the particular TLM level, the XML system specification structure contains its associated elements and the corresponding XML schemas are used in order to assure the correctness of the automatically transformed code. Furthermore, this creates a possibility of reuse of different TLM models.

In Figure 16 the representation of the XML schema defining a structure for the *process* node is shown. When a designer specifies the *process* node, its structure along with some constraints on the data are validated according to the presented schema.

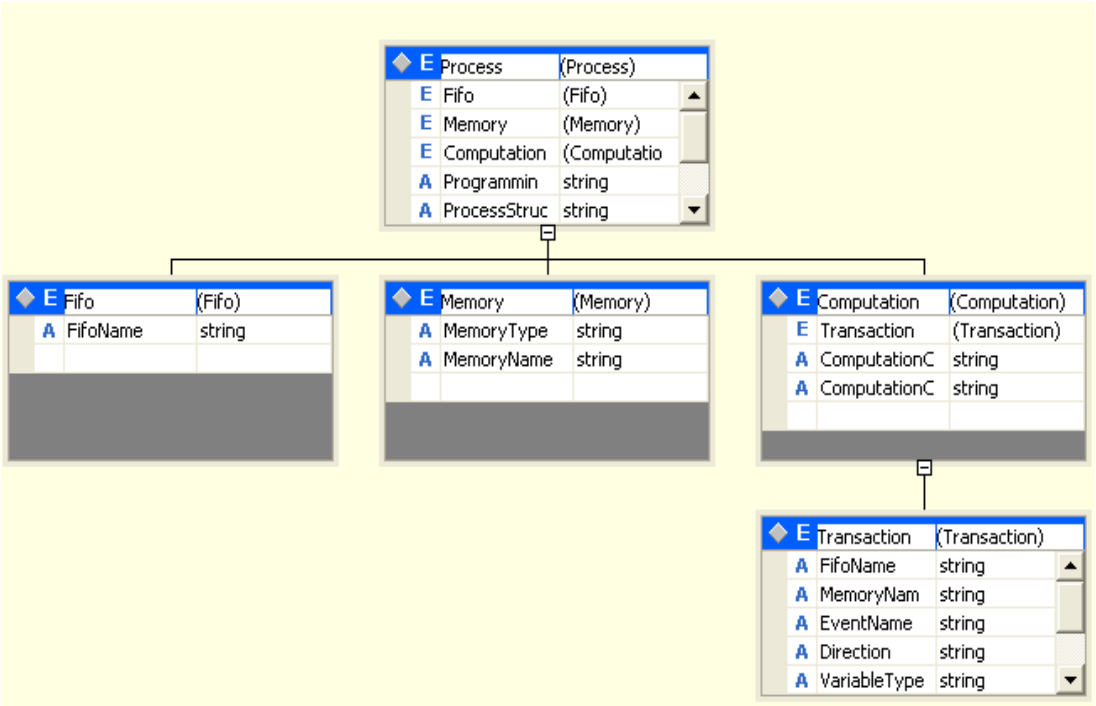

Figure 16: Graphic representation of XML schema for *process* description

4.4 Simulation model generation

The CP XML system specification is a non executable one. In order to simulate the system model, this specification is automatically transformed into a simulation model using XSLT transformations corresponding to each abstraction level.

EXtensible Stylesheet Language Transformation, XSLT [15], is used to transform the content of a source XML document into another document having a different format or structure. We have used XSLT to implement the transformation of the system representation from one computational model to another. Figure 17 demonstrates a fragment of *process* node transformation into C# code. We can see that if the data of the FifoName attribute is not empty, we have to analyze the transaction `Direction` (read or write) and retrieve the necessary information to construct a C# statement.

```
<xsl:if test="Transaction/@FifoName[.!='']">
<xsl:choose>
<xsl:when test="Transaction/@Direction[.='write']">
<xsl:value-of select="Transaction/@FifoName"/>.Write(
               <xsl:value-of select="Transaction/@VariableName"/>);
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="Transaction/@FifoName"/>.Read(
         ref <xsl:value-of select="Transaction/@VariableType"/> O);
</xsl:otherwise>
</xsl:choose>
</xsl:if>
```

Figure 17: Fragment of *process* node transformation

The simulation model generation flow is presented in Figure 18. The XML language allows the definition of an unlimited number of tags. This feature is a language advantage though, in the same way it can also be a cause of difficulties. For each defined tag, we must provide an implementation according to the semantics of the chosen computation model.

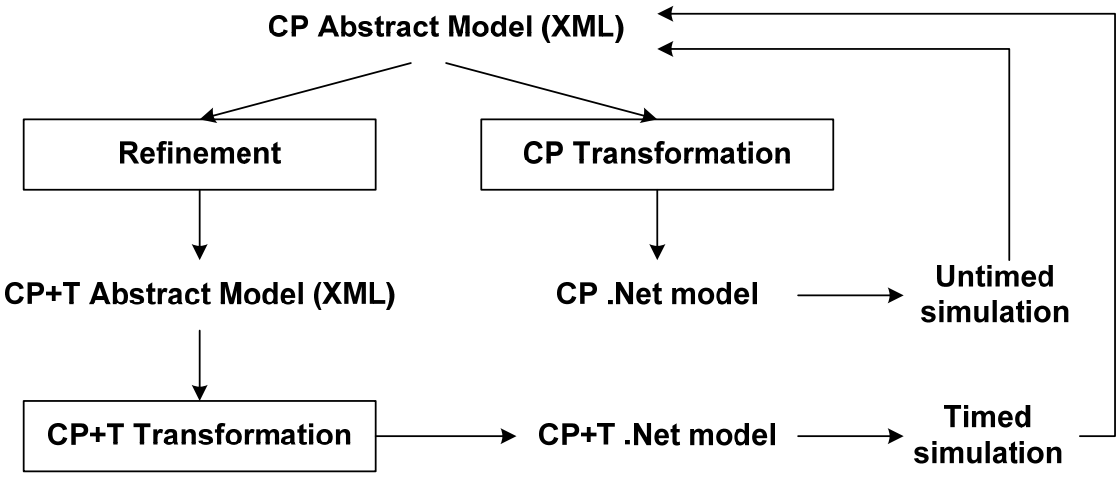

Figure 18: Simulation model generation flow

We have defined a limited set of about sixty tags, structurally organized according to the CP Abstract Model. Process functionality is described using one of several programming languages supported by .NET thereby making possible the reuse of functional specifications written in different programming languages. This feature is supported by the .NET language interoperability capabilities. The transformation process of the XML non-executable model, with data specified in different programming languages, into a .NET simulation model is presented in Figure 19. In simulation models, all transaction implementation details are generated automatically.

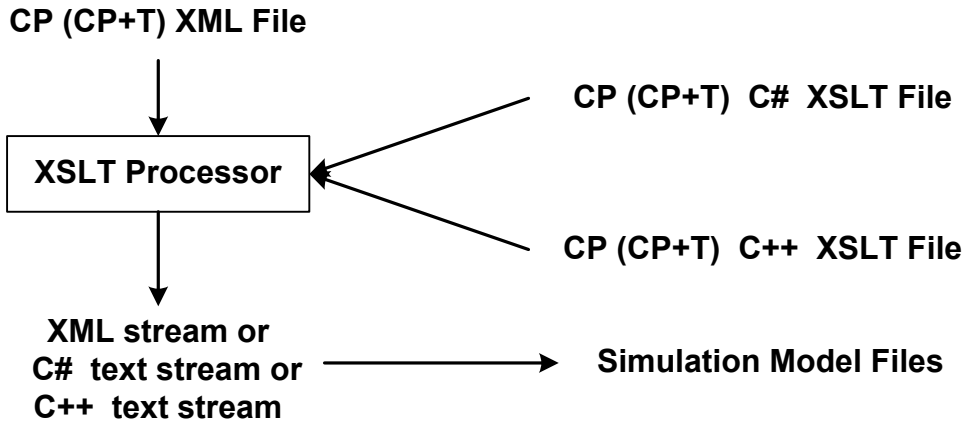

Figure 19: TLM level Transformation

4.5 Abstract model and TLM

System behavior at the CP level is seen as a network of parallel processes communicating through unbounded FIFOs, shared memory or by events. We have chosen the .NET Framework as a development environment as it provides multiple classes for XML parsing, validation, and transformation. This support guarantees easy XML representation and efficient system description transformations into simulation models at different TLM abstraction levels. In .NET, we implement parallel process activities using multithreaded programming. As processes at high TLM levels exchange complex data structures, the existence of Collections Classes that group similarly typed objects is very useful. In these Collections, memory management is handled automatically and the capacity of a collection is expanded as needed. In the modeling of data dependent synchronization, we have used a Queue Class to implement an unbounded FIFO. The control-dependent synchronization used in high level TLM is modeled using the .NET event mechanism.

The XML language allows for the definition of an unlimited set of tags to structure and encapsulate the data. Difficulties appear in using XSLT as the data have to be extracted and executable code must be generated by manipulating this data according to a certain computational model. For example, in the currently proposed abstract model expressing the CP TLM paradigm, we do not support dynamic process instantiations.

By using XML, different types of constraints can be expressed in the proposed abstract model. Firstly, we will concentrate on the temporal type of constraints associated with different architecture components commonly used in Hw/Sw systems. To define the architecture constraints, we will use a component library where with each component we will associate timing information in the form of a set of temporal constraints. During the transformation of a non-executable specification into an executable one, the timing information associated with the architecture component will be incorporated in the simulation model. Our simulation is performed in several steps. The first step corresponds to system representation at the CP level. Following this first step, we will continue the simulation using the

architecture component's temporal constraints and we will verify them. The second step corresponds to system representation at CP+T level. For each simulation, increasingly more precise timing information concerning communication and computation is added in a path. In this way we can add to the system description the implementation details. This process will lead to the system model passing through all TLM abstraction levels.

When the architecture and explicit timing constraints are not specified in the model, we suppose that the maximum performance has to be found and that a number of architecture configurations has to be explored in order to find a better solution for a given system behavior. When a timing violation is found, the designer will have to change the architecture constraints if they have been defined, or "reengineer" the distribution of system functionality between the processes.

According to the type of architectural constraints defined in the application model, the system model is transformed in one of five TLM use-models presented in the section 2.4.2. In the case of heavily architecture constrained systems, a system model is a UM1. For this type of models, the main design activities are concentrated at the PV+T TLM level. Use-models UM2-UM5 [7] that may be received as a result of architecture constraints relaxation will require design space exploration based on performance analysis. A design flow determined by our application model is presented in Figure 20.

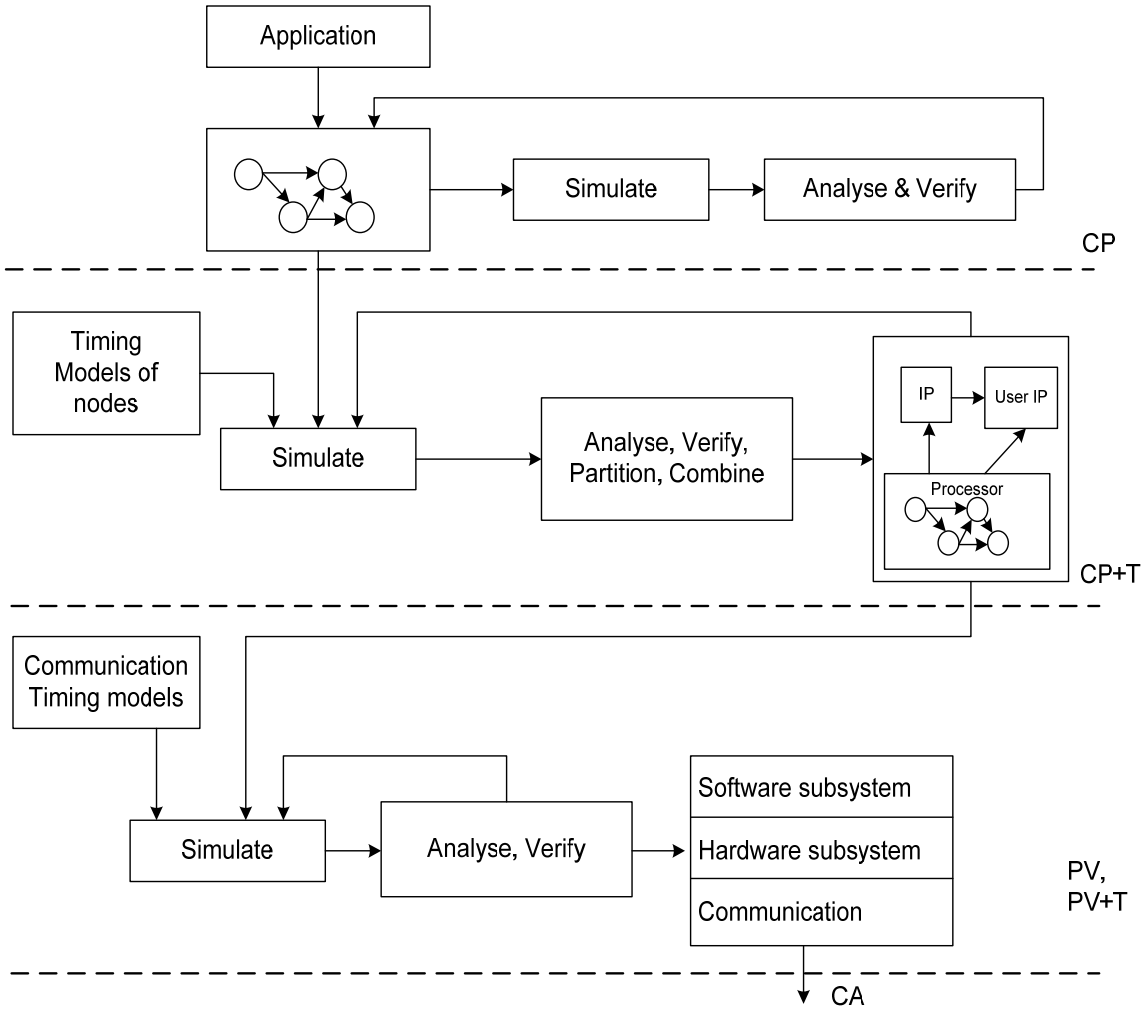

Figure 20: Design flow

4.6 Experimentations

4.6.1 System Description

In this subsection, we present a TLM high level model of an audio-video server system. The initial system specification [13] is as follows: a server system acting as a video player, allowing users to read or write video or audio sequences. These sequences are of variable duration (from one minute to several hours) and the sequence rate may range from 0 to 15 Mb/s. The server has to satisfy a maximum 1000 simultaneous users.

4.6.2 Abstract Model of audio-video server system

In order to explore the audio-video server system at the CP level, we have implemented the test bench or environment and the server CP models (Figure 21).

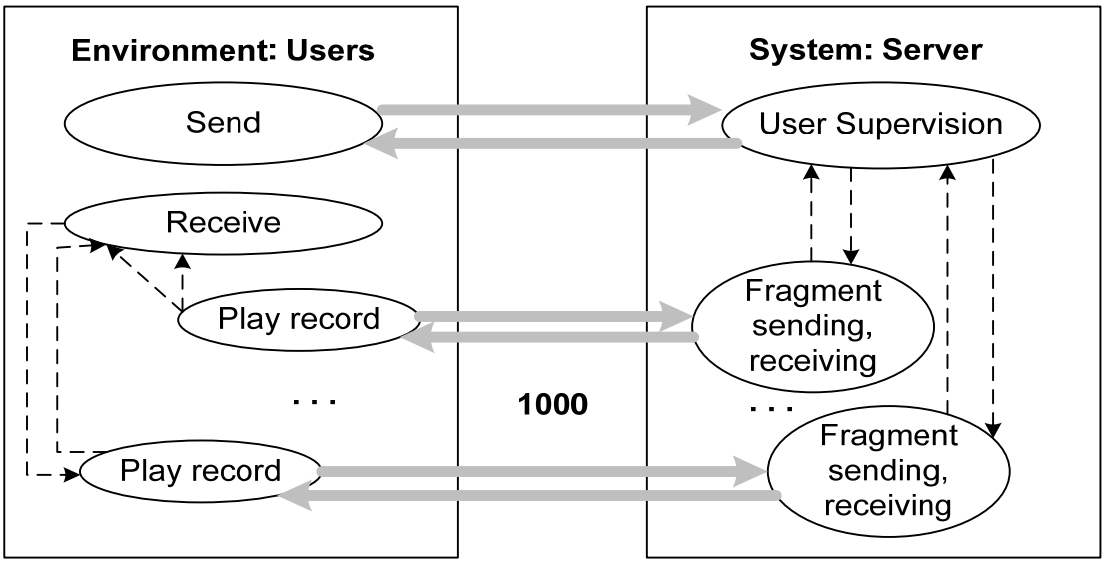

Figure 21: CP Abstract Models of Test Bench and Server System

An environment model represents a set of users and consists of one *Send*, one *Receive* and 1000 *PlayRecord* processes. At each moment, *PlayRecord* processes can be suspended or reactivated with new data for transmission. The *Send* process is responsible of generating different user commands and sending them to the server. The *Receive* process interprets server responses and communicates the necessary information to the corresponding *PlayRecord* process. As this information must have an immediate effect, the *Receive* process communicates with *PlayRecord* processes using events. All processes in the Environment Model are annotated with timing information and, as a consequence, they have some latency during simulation. The environment model's goal is to provide valid test-vectors to the system model.

The Server model contains one *UsersSupervision* and 1000 *FragmentSendingReceiving* processes. *The UsersSupervision* process interprets the received commands, manages access to the sequence bank, allocates and frees the server resources and initiates transmissions. All Server-Environment

communications at the CP level are through FIFOs. In contrast to the Environment Model, the Server system at CP level is completely un-timed with only deals with logical event ordering. During a refinement step, we have to ameliorate the system description by adding the temporal and functional details.

4.6.3 XML server system specification and CP server simulation model

Initial functional descriptions for each process in the audio-video server system and environment model have been coded in C#. These descriptions, as well as the indication of the communication mechanism used for transactions between communicating processes have been placed in an XML specification. A fragment of the server system specification is shown in Figure 22.

```
<Process ProgrammingLanguage="C#" ProcessStructure="simple" Type="Send">
  <Fifo Name="CmdSend" />
  <Declarations>
    ArrayList SimultaneousUsers=new ArrayList();
    . . .
   </Declarations>
   <Computation>
    <ComputationCode>...</ComputationCode>
   <Transaction FifoName="CmdSend" Direction="write"
VariableType="ArrayList"
                  VariableName="SimultaneousUsers" Delay="delay"
EventName="">
   </Transaction>
```

Figure 22: Fragment of XML server system specification

The XSLT transformation supporting the CP abstract model, based on .NET features, is applied to the XML specification to create an executable CP level system model. The Transformation process for C# code generation is presented in Figure 23.

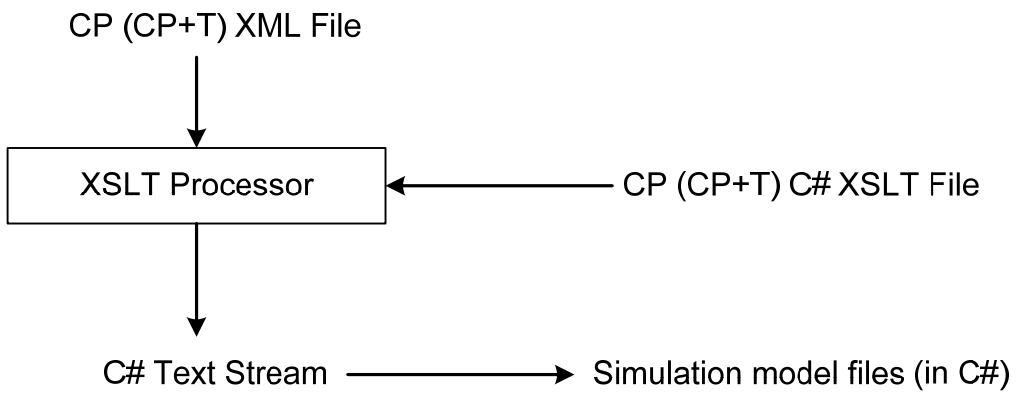

Figure 23 : CP C# Transformation process

C# XSLT transformation extracts the necessary data from the CP XML description, completes it and generates C# simulation model files. Figure 24 shows the result of applying C# XSLT transformation to the XML specification from Figure 22.

```
ArrayList SimultaneousUsers = new ArrayList(); . . .
while(true){
. . .
CmdSend.Write(SimultaneousUsers);
Thread.Sleep(delay);
} . . .
```

Figure 24 : C# code fragment after XSLT transformation

We have simulated the server system in order to verify whether the system specification was captured properly and also to validate the system functionality partitioning. The possibility of using XSLT transformations in automatic code generation facilitates the exploration process. All implementation details of communication mechanisms corresponding to the CP abstraction level are generated automatically providing in this way an easy method of high level model creation. The separation of the system functionality into parallel tasks takes into account some architectural concerns. Multiple high level models can be created and simulated in order to decide the final system functionality distribution between processes and the transaction synchronization types.

During our exploration of different high level server models, we have shown that we cannot only use FIFOs to model the server communication mechanism. In

other words, we need the events to describe the reactive server behavior and the shared memory mechanism for the communication of the *FragmentSendingReceiving* processes. The designed CP model is a starting point for the refinement flow through TLM abstractions.

At the CP+T level, the *delay* tag is added to the transaction node to designate explicit temporal constraints in the specification. In our server model we consider only temporal constraints. Other constraint types can be specified by adding code with corresponding semantics in the XSLT transformation. At the CP level, any HW architecture elements are present. At the CP+T level, some explorations may be done in order to map one or more processes implementing computations into HW modules. For these sorts of explorations we need a library of temporal descriptions of common HW modules.

4.7 Conclusion

In this chapter, we have defined a TLM framework for system refinement and analysis. The goal of the introduction of a new TLM Hw/Sw modeling and simulation methodology was to provide an approach with separation of concerns, where the timing aspect could be used not only in simulation exploration but also in analytical timing verification. We have presented a design flow of this methodology with the identification of design activities at each abstraction level. Automated support of the presented methodology is implemented for the high CP, and partially CP+T, abstraction levels. The example of an audio-video server model is shown to demonstrate in which way the modeling and refinement process may proceed.

Chapter 5. Timing specification in TLM

Several levels in TLM are characterized by a timing annotation of system behavior descriptions. In currently existing TLM based design methodologies, the amount of timing information required at each TLM level is defined by the TLM APIs (Application Program Interface) which is a set of interfaces describing the different communication schemes [58]. Following this, the corresponding TLM simulation models are used in the design space exploration phase leading to final system implementation. In our TLM methodology, we propose a separation of concerns, where the timing aspect is separated from functionality. Different explorations and reduction of the design space can be obtained without any functional specification. After this step, functional and timing aspects can be merged for further refinements. Furthermore, several decisions can not only be done without simulation, but also formally proven.

In the next subsection, we will introduce the notion of timing specifications and timing constraints. Further, we will consider in which way temporal details will be incorporated in the initial abstract model in order to express the system at different TLM abstraction levels.

5.1 Expressing timing

The temporal behavior of systems is frequently described using event-based representations where events are characterized by their occurrence times with timing constraints. A timing model based on min-max-linear constraints was used by Gahlinger [28], Vanbekbergen et al. [61], McMillan and Dill [29] and other researchers to solve the problem of the timing verification of interface timing specifications. This timing model is based on timing diagram analysis and was used at a low level of modeling.

A timing diagram represents an external view of circuit behavior. More precisely, it demonstrates the constraints on the timing of critical signal transitions by showing signal waveforms on the connections between the environment and the

circuitry [28]. Because of their simplicity and expressiveness, timing diagrams are often used to specify the device behavior. A timing specification based on timing diagram analysis includes timing characteristics which represent relationships between the times of events. An example of a timing diagram is given in Figure 25

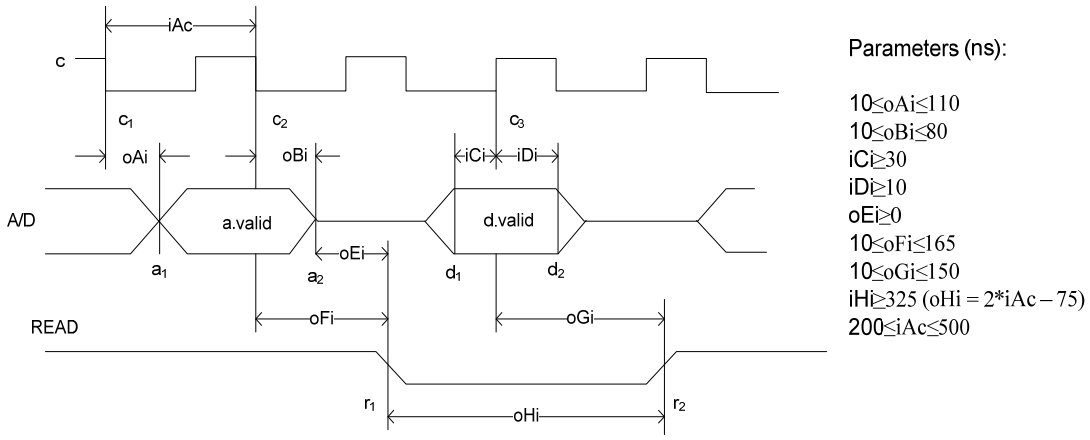

Figure 25: Read-cycle timing diagrams for the 8086 CPU [28]

Graphically, the timing diagram and its characteristics can be presented by an event graph whose nodes define events represented the changes in the values of signals whereas the weighted directed edges show the delay constraints between events. The weights assigned to every arc denote a minimum and a maximum time of event firing relative to the firing time of its predecessor. It is assumed that the time it takes to fire a transition is between zero and infinity and there is only one source event with a firing time of zero.

Definition 1 [32]

An *event graph EG* can be associated with each timing diagram: *EG = (E, C)* where the set of vertices *E* corresponds to the set of events and the set of directed edges *C* corresponds to a set of constraints $C = \{c_{ij} = (e_i, e_j), [l_{ij}, u_{ij}] \mid e_i, e_j \in E\}$.

The event graph associated with the timing diagram from Figure 25 is shown in Figure 26.

Definition 2 [32]

To each event e_i we assign an *occurrence time* of e_i denoted by $t(e_i)$ such that $l_{ij} \le t(e_j) - t(e_i) \le u_{ij}$ for all $c_{ij} \in C$.

In Figure 26, event c_1 is a source. For each arc the first weight number denotes a lower bound and the second, an upper bound on the delay constraint between occurrence times of events.

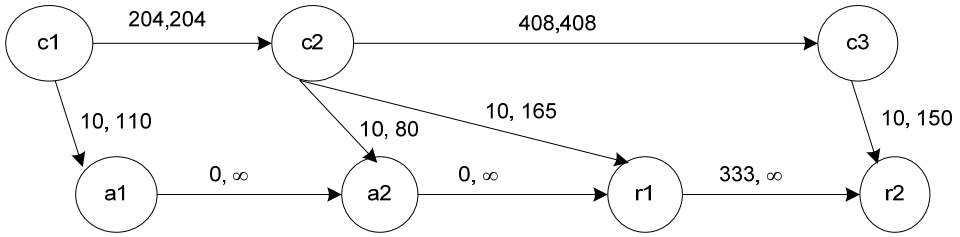

Figure 26: Partial graphic representation of the timing diagram from Figure 25

A definition and an interpretation of four constraint types were given by Vanbekbergen et al. [61]. Inequalities (1) define linear constraints where the occurrence time of the event e_j has to satisfy simultaneously all delay constraints connected it to its predecessors. In the second constraint type, the max type (2), in contrast to linear constraints, only one upper bound, the latest, has to be satisfied. In the case of min type (3), only one lower bound, the earliest one, has to be satisfied among all lower bounds. Finally, in the fourth constraint type, the latest upper bound and the earliest lower bound have to be satisfied. For this constraint type, researchers did not find any practical use.

$$\max_i \left(t(e_i) + l_{ij}\right) \le t(e_j) \le \min_i \left(t(e_i) + u_{ij}\right) \qquad (1)$$

$$\max_i \left(t(e_i) + l_{ij}\right) \le t(e_j) \le \max_i \left(t(e_i) + u_{ij}\right) \qquad (2)$$

$$\min_i \left(t(e_i) + l_{ij}\right) \le t(e_j) \le \min_i \left(t(e_i) + u_{ij}\right) \qquad (3)$$

$$\min_i \left(t(e_i) + l_{ij}\right) \le t(e_j) \le \max_i \left(t(e_i) + u_{ij}\right) \qquad (4)$$

The interpretation of the four constraint types is given in Figure 27. The black dots indicate the firing times of the events e_1, e_2, e_3. Suppose that they are 2, 4, 10 time units correspondingly. We want to find a range of timing assignments for the event e_j, having the following delay constraints: $e_1 \rightarrow e_j = [9, 12]$; $e_2 \rightarrow e_j = [5, 11]$; $e_3 \rightarrow e_j = [2, 7]$. The clear boxes indicate the time interval in which e_j can fire according to each constraint considered separately. The shaded boxes demonstrate the firing interval of event e_j according to each constraint type. As it can be observed, each one of these intervals is different depending on the constraint type.

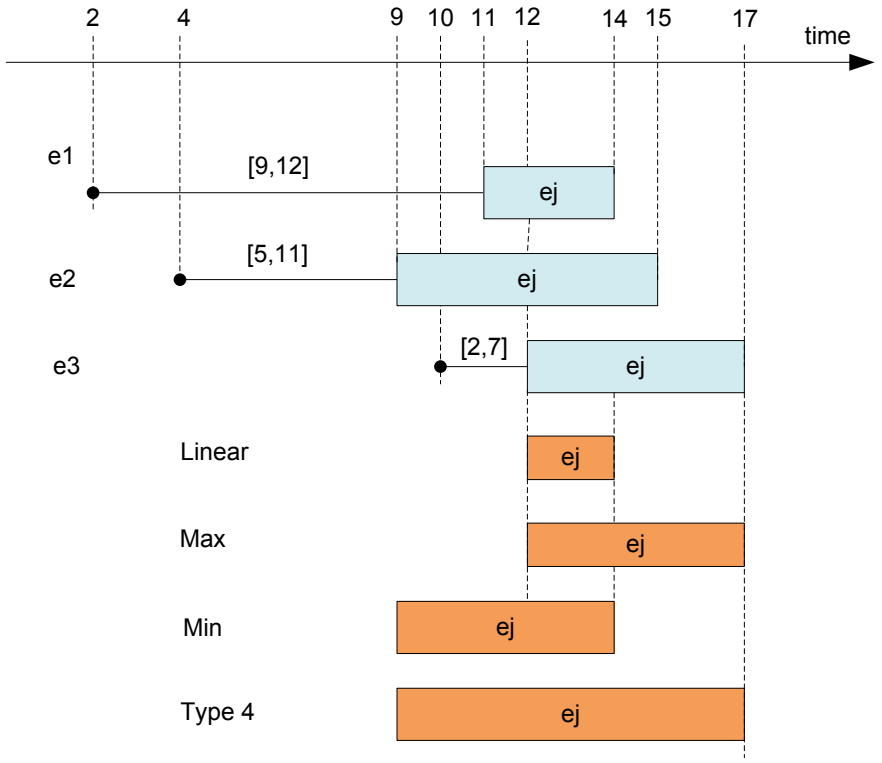

Figure 27: Interpretation of the different types of constraints [61]

One of the problems of timing verification is formulated as finding the maximum achievable separation s_{ij} between each pair of events e_i and e_j under a system of one or several types of timing constraints:

$$s_{ij} = \max_i (t(e_j) - t(e_i))$$

For example, consider the subgraph with linear constraints from Figure 28. Constraints $a \rightarrow c$ and $b \rightarrow c$ are linear; the occurrence time of the event *c* has to satisfy the following two constraints:

$$t(a) + 40 \leq t(c) \leq t(a) + 60$$
$$t(b) + 30 \leq t(c) \leq t(b) + 80$$

From these two inequalities we have:

$$-40 \leq t(b) - t(a) \leq 30$$

Thus, $[-s_{ba}, s_{ab}]$, the maximum separation time between *a* and *b*, is [-40, 30].

In event graphs we will use a circle to denote an event with linear constraints, a square to represent a max event and a pentagon to represent a min event.

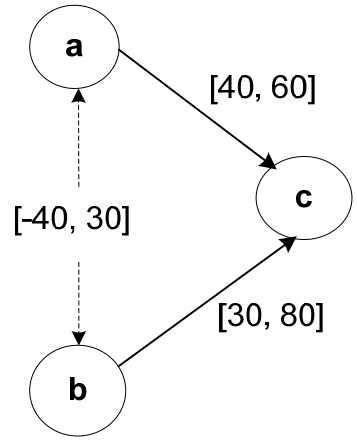

Figure 28: Subgraph with linear constraints

The maximum separation calculation determines if the timing specification is consistent, if at least a feasible timing assignment exists and also gives us the possibility to verify if all the timing requirements are satisfied for all timing assignments.

The complexity for deriving the maximum separations of all nodes from a single node varies with the types of constraints which are allowed (Table III). Some solutions are graph-based algorithms [25], [29], [38] whereas others use a reformulation of constraint specifications into a mathematical optimization problem

followed by use of existing solvers to resolve the min-max-linear constraint problem [62].

Table III: Complexity of the maximum separation problem [25]

Constraint type	Complexity	Proposed by
Linear only	O(EC)	Shortest-path algorithms
Max only	O(C)	McMillian & Dill [29], Vanbekbergen et al. [61]
Max + linear	$O(E^2 \log E + EC)$ conjecture	Ti-Yen Yen et al. [26]
Max + linear	$O(E^5)$ conjecture	Walkup & Borriello [38]
Min + Max	NP- complete	McMillian & Dill [29]
Min + Max + Linear	NP- complete	T. M. Burks, K. A. Sakallah [62]

5.2 Timing Analysis

In the following subsections, we will consider timing specifications with different constraint types and the corresponding graph-based algorithms to resolve the problem of timing verification. We consider the graph-based interpretation of timing specifications as being better understood intuitively than a reformulation of the timing specification into a mathematical optimization problem. Furthermore, graph-based algorithms are quite simple when compared to complex solvers and can be easily implemented.

5.2.1 Linear constraint systems

Several design problems require the solution of a set of linear inequalities of the type:

$$t(e_j) - t(e_i) \le w_{ij},\ i, j = 0, 1, \ldots, n\text{-}1.$$

A delay constraint $t(e_i) + l_{ij} \le t(e_j) \le t(e_i) + u_{ij}$, can be split into two inequalities $t(e_j) \le t(e_i) + u_{ij}$ and $t(e_i) \le t(e_j) - l_{ij}$ which can be presented by forward and backward edges correspondingly (Figure 29).

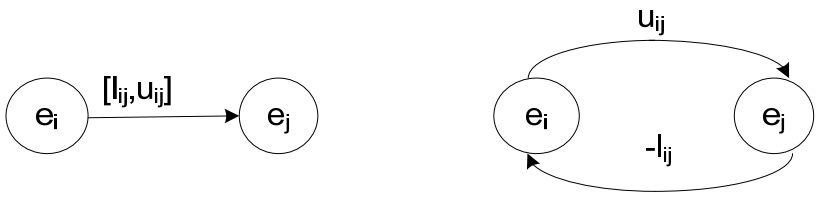

Figure 29: Graphic representation of delay constraint

A set of linear inequalities can be presented as a directed graph with *n* vertices and *m* edges, each one weighted by w_{ij}. Such a graph is called the *constraint graph* and its properties are related to the satisfiability of a set of linear inequalities. More precisely, the set of linear inequalities presented above, is satisfiable if and only if the corresponding shortest path problem is consistent [60].

For linear constraint systems, we need to find, for each pair of nodes, a greatest lower bound and a least upper bound: $\max_i\left(t(e_i)+l_{ij}\right)\le t(e_j)\le \min_i\left(t(e_i)+u_{ij}\right)$. We can do this by finding the shortest paths in the corresponding graph. Considering two nodes in the graph e_i and e_j, in order to get the least upper bound we have to find the shortest path from the e_i node to e_j (s_{ij}) and to get the greatest lower bound, we need to find the shortest path from e_j to e_i ($-s_{ji}$). As a lower bound in the delay constraint is presented in the graph by a negative number, to find the shortest paths we use the Bellman-Ford algorithm [60].

Consider a graph G= (V, E), where for V, a set of vertices and E, a set of edges, the algorithm finds a vector $d(v)$ for all the shortest-path lengths from a source $s \in V$ to all $v \in V$.

Algorithm 1: Bellman-Ford algorithm

```
d(s) ← 0
for each v ∈ V \ {s}
do d(v) ← ∞
end for
for i ← 1 to |V| − 1 do
    for each edge (u,v) ∈ E do
        if d(v) > d(u) + w(u,v) then
```

Initialisation

Relaxation step

```
                    d(v) ← d(u) + w(u,v)
                    π(v) ← u
               end if
          end for
     end for
     for each edge (u,v) ∈ E do
          if d(v) > d(u) + w(u,v) then
               report that a negative-weight cycle exists
          end if
     end for
```

5.2.2 Max constraint systems

For the timing specification of systems with max-only constraints we have to find the greatest lower bound and the greatest upper bound:

$$\max_i\left(t(e_i)+l_{ij}\right)\le t(e_j)\le\max_i\left(t(e_i)+u_{ij}\right)$$

Our algorithm for finding the maximum separation time between any two events in a graph with only max constraints is based on the ideas presented in [61].For each pair of nodes we have to find a greatest lower bound and a greatest upper bound. A greatest lower bound can be found by minimizing the total delay for all l-bound paths of the nodes and a greatest upper bound for the events e_i and e_j can be obtained by means of maximizing the separations s_{ki} for predecessor e_k of e_j. Then, in order to compute the maximum separation time between the two events e_i and e_j we have to consider two cases:

1. There is a path from e_j to e_i ending in the source node. In this case all delays have to be set to their lower bounds.
2. There is no path from e_j to e_i ending in the source node. For this case we have to maximize $t(e_j)$ - $t(e_i)$ by maximizing $t(e_k)$ - $t(e_i)$ for each predecessor e_k of e_j.

Algorithm 2: Maximum separation in max-only constraint systems

```
Step 1:   if i=j then
               return 0
          end if
```

```
          if max separation already is calculated then
               return previously calculated value
          end if
Step 2:   if path between e_i and e_j exists then
               Max = ∞;
               for each vertex k, predecessor of vertex j
               do
                    newMax ← MaxSep(e_i,e_k)-l_kj {l_kj: lower
                                   bound of the arc k→j}
                    if newMax ≤ Max then
                         Max=newMax;
                    end if
               end for
          else
            Max ← -∞;
            for each vertex k, predecessor of vertex i do
               newMax ← MaxSep(e_k,e_j) + u_ki { u_ki: upper
                                   bound of the arc k→i}
                    if Max ≤ newMax  then
                         Max ← newMax
                    end if
            end for
          end if
          return Max
```

The complexity of the algorithm computing the separations s_{ij} between all pairs of events for the graph with number of events (vertices) n and number of edges e, is $O(n \cdot e) \leq O(n^3)$.

We applied the algorithm presented above to the graph shown in Figure 30[61].

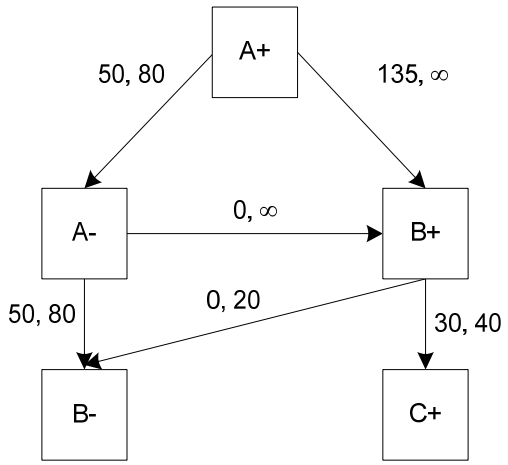

Figure 30: Max only constraint timing model

The results are given in Table IV and correspond to what was obtained in [61].

Table IV: Maximum separation time for the timing specification of Figure 30

	A+	**B+**	**A-**	**C+**	**B-**
A+	0	∞	80	∞	∞
B+	-135	0	-55	40	25
A-	-50	∞	0	∞	∞
C+	-165	-30	-85	0	-5
B-	-135	0	-55	40	0

5.2.3 Max-Linear Systems

Max-linear systems are the most discussed in literature. The max-linear temporal model is widely used in the description of interfaces. Several algorithms were proposed to solve the timing verification problem for such systems. McMillan and Dill [29] proposed a graph-based algorithm with an exponential worst case running time. An interesting algorithm for max-linear systems was proposed by T. Y. Yen et al. [26]. According to their experimental results [30], [31] this algorithm is quite efficient. However, the exact complexity has not been given. The algorithm uses the "iterative tightening from below" approach and is based on two steps.

The first step consists of the generation of a special intermediate graph, termed the compulsory graph, containing the arcs representing bounds that must be satisfied (compulsory bounds). For max events, these arcs represent the lower bounds whereas for min events they denote upper bounds. Also, for linear constraints, upper and lower bound arcs are compulsory arcs. From this graph, the smallest separation values that satisfy the compulsory bounds are obtained such that they may serve as the initial estimation for the tightening process.

Definition 3 [27]

Given an event graph *G=(V, E)* and a source node *s*, the corresponding *compulsory graph* $G_c=(V, E_c)$ is a weighted directed graph, where E_c contains the following edges. For each linear or max constraint $c_{ij} \in E, e_{ij} \in E_c$ and has weight $weight_c\lfloor e_{ij} \rfloor = c_{ij}.lower$. For each linear constraint $c_{ij} \in E, e_{ji} \in E_c$ and has weight $weight_c\lfloor e_{ji} \rfloor = -c_{ij}.upper$. For each node $i, e_{si} \in E_c$ and has weight *–MAXINT*, where

MAXINT is an arbitrary number larger than any sum of the absolute values of the finite bounds, but obeying the arithmetic rules of finite numbers.

A compulsory graph for the graph presented in Figure 31 (a) is shown in Figure 31 (b).

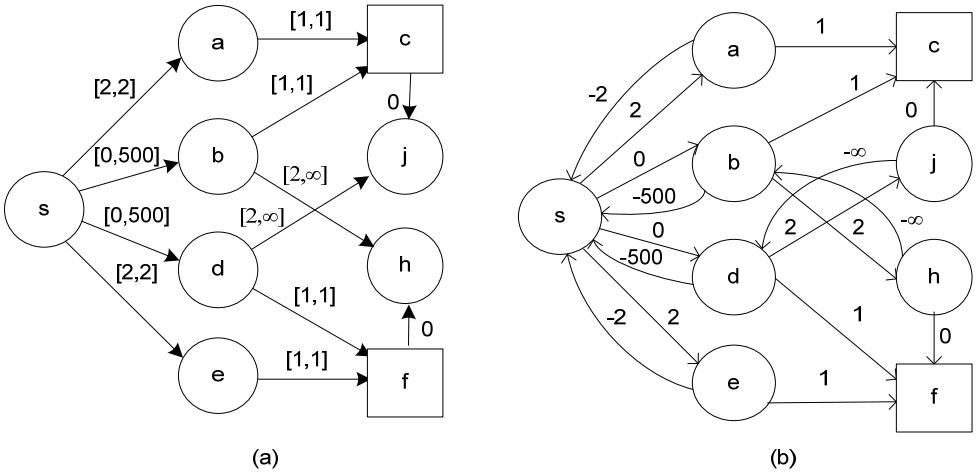

Figure 31: An event graph (a) and the corresponding compulsory graph (b)

The second step of the algorithm consists of constructing another intermediate graph, a slack graph, defining the values by which the max (min in the case of min-linear systems) constraint arcs can be tightened. Following this, the upper bounds for max constraints and lower bounds for min constraints are reintroduced in the compulsory graph and the initial separations are iteratively relaxed according to the slacks.

Definition 4 [27]

Given an event graph *G=(V, E)*, a source node *s*, and a separation value *sepa*[*i*] for each node $i \in V$, the *slack graph* $G_s = (V, E_s)$ is a weighted directed graph where E_s is defined as follows: for each constraint $c_{xy} \in E$, construct two edges e_{xy} and e_{yx}, and define

$$e_{xy}.bound = c_{xy}.upper$$
$$e_{yx}.bound = -c_{xy}.lower$$

For each $e_{ij}.bound$, when $sepa[i] \neq \infty$ or $sepa[j] \neq \infty$, add edge e_{ij} to E_s with edge weight $weight_s[e_{ij}] = e_{ij}.bound - (sepa[j] - sepa[i])$ if the weight is nonnegative.

When $sepa[i] = sepa[j] = \infty$ add edge e_{ij} with $weight_s[e_{ij}] = 0$. Mark $e_{ij} \in E_s$ as *max-optional* if c_{ij} is a max constraint; otherwise it is *compulsory*. If a node *u* is a max event and no max-optional edge enters *u*, add a compulsory edge e_{su} with weight zero.

In Figure 32 (a) the event graph is shown with the calculated initial time separations (labels of the nodes). Figure 32 (b) gives the slack graph after the first iteration. The value inside each node represents its maximal slack.

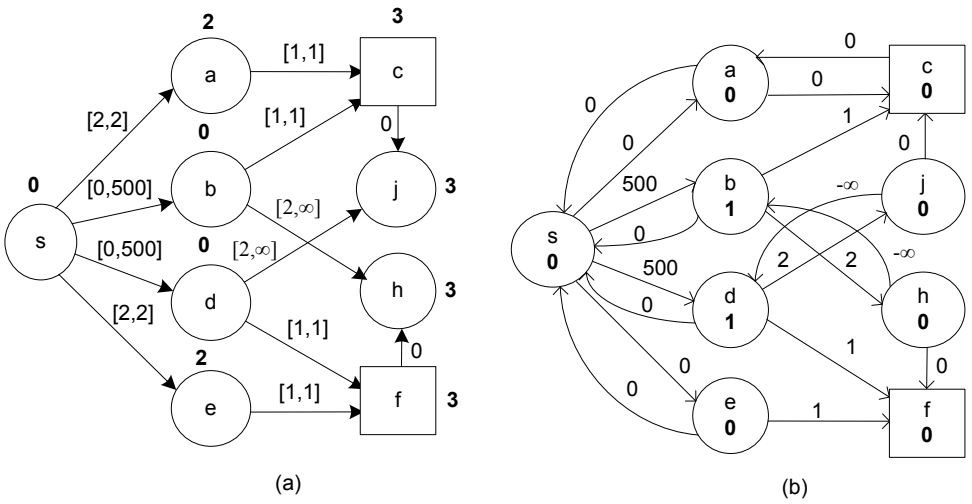

Figure 32: An event graph (a) and the corresponding slack graph in the first iteration (b)

The pseudo code for two procedures [26], namely the computation of shortest slack and the maximum separation calculation, is given below.

Algorithm 3: Shortest slack calculation

```
Shortest slack is calculated from a source node s for
each node in a slack graph Gs
Step 1: Initialization
for each node ei do
     Put node ei in Queue Q
     Initialize safe slack estimate d[i]←∞
     Initialize temporal value for slack m[i]←0
     n[i]←number of max-constraints
end for
d[s]←0
Step 2:
while Queue is not empty do
```

```
     Find node e_i in the Queue with a minimum d[i]
     for each edge outgoing from this node in the slack
          graph Gs do
          relax the edges: t←d[i]+weight(i→j)
               if node e_j is a max event then
                    n[j]--
                    m[j]←min(m[j],t)
                    if n[j]=0 and d[j]>m[j] then
                              d[j]←m[j]
                    end if
               else
                    d[j]←max(d[j],t)
               end if
               Remove e_i from the Queue;
     end for
end while
```

At the end of the execution of shortest slack procedure, *d*[i] is a shortest slack estimate from a source node *s* for each node e_i. This procedure is used for the computation of the maximum achievable separations for a constraint graph.

Algorithm 4: The MaxSeparation algorithm

The maximum separations are found from a source node *s* for an event graph *G* with max and linear constraints

```
Step 1: Construct the compulsory constraint graph Gc

Step 2: Calculate longest paths in the graph Gc from a
source node to each other node
if a positive cycle exists then
     return inconsistent
end if

Step 3: Initialization of the initial separations
for each node e_i do
     Set separation from s to e_i=the weight of the
     longest path from s to e_i
end for

Step 4:   Iterative relaxation
repeat
     Construct the slack graph Gs
     Calculate the shortest slack from a source node s to
     each node using the previous procedure
     for each node e_i do
          if the shortest slack is ∞ then
```

```
                set separation from s to e_i ∞
            else if the shortest slack > 0 then
                    Increase separation by the shortest
                     slack
            end if
        end for
until the shortest slacks do not change
if all constraints in graph G are satisfied then
        return problem is consistent
else
        return problem is inconsistent
end if
```

The authors conjecture that the complexity of this algorithm is $O(VE + V^2\log(V))$.

5.2.4 Min-Max Constraint Systems

McMillan and Dill proved that the problem with max and min constraints is NP-complete [29]. To prove NP-completeness of the min/max constraint problem McMillan and Dill used the reduction from 3-SAT.

Theorem 1 [29]
3-SAT is reducible to the min/max problem.

Proof

Let φ be a 3-CNF with n variables $a, b, c, \ldots$; $p_a, p_b, p_c, \ldots n_a, n_b, n_c, \ldots$ corresponding to the positive and negative literals respectively. Formula φ has m clauses. We demonstrate the reduction from 3-SAT to the min/max problem that converts formulas to graphs. Structures within the graph are designed to mimic the behavior of variables and clauses. Create graph G of a *min/max* problem as follows:

Let s be a source event and $p_a, p_b, p_c, \ldots n_a, n_b, n_c, \ldots$ - events in constraint graph G subject to the constraints:

$$t_{p_v} = t_s + \partial_{sp_v}, \text{ where } 0 \le \partial_{sp_v} \le 1$$

$$t_{n_v} = t_s + \partial_{sn_v}, \text{ where } 0 \le \partial_{sn_v} \le 1$$

The time of each event p_v, n_v, ranges between 0 and 1 relative to the source event s. Graphically, this is expressed by the arcs going from the source event to each event p_v, n_v (Figure 33). For each pair of events p_v, n_v we construct a min event m_v:

$$t_{m_v} = \min(t_{p_v}, t_{n_v}).$$

There is also a max event q that is the latest of the m_v events:

$$t_q = \max_{j \in \{a,b,c,\ldots\}} (t_{m_j})$$

For each clause we construct a max event f_i which is the latest of the corresponding events p_v, n_v forming the clause. For example, for the clause $p_a + p_b + p_c$:

$$t_{f_i} = \max(t_{p_a}, t_{p_b}, t_{p_c})$$

Finally, we construct a min event r that is the earliest of the f_i events:

$$t_r = \min_{1 \le i \le m} (t_{f_i})$$

We have to find the maximum separation s_{qr} in the constructed constrained graph.

To prove that this reduction works, we have to demonstrate that the maximum separation s_{qr} equals 1 if and only if the formula φ is satisfiable.

1. Suppose the formula φ has a satisfying assignment. In that satisfying assignment, at least one literal is true in every clause. For either assignment of variables $p_v = 1$, $n_v = 0$ and $p_v = 0$, $n_v = 1$, it follows that the execution time of the min events m_v coincides with the earlier of p_v or n_v and is equal in all cases to 0. A max event q coincides with the latest of the m_v events and, as the m_v events fire at 0, thus $t_q = 0$. The firing time of max events f_1, f_2 will be always 1 since at least one delay from the events representing clause variables p_v, n_v must be 1, leading to the defining of the execution time of the event r being 1. Thus, $s_{qr} = \max(t_r - t_q) = 1$.

2. Suppose now that the maximum achievable separation $s_{qr} = \max(t_r - t_q) = 1$. This means that $t_r = 1$ and $t_q = 0$. As $t_r = 1$ and *r* is a min event, both events f_1 and f_2 have to fire at time 1 and thus $t_{f_1} = 1, t_{f_2} = 1$. Since $t_q = 0$, at least one delay from each pair p_v, n_v must be 0. Thus, if $p_v = 1$ and $n_v = 0$, this assignment to the variables satisfies φ because according to the firing time $t_{f_1} = 1, t_{f_2} = 1$, each triple of events representing a clause in the graph contains a literal that is assigned TRUE. Therefore we have a satisfying assignment of the formula φ.

Figure 33 shows an example of the constraint graph corresponding to the 3-SAT formula $\varphi(a,b,c,d) = (a + b + c)(\bar{a} + \bar{b} + d)$.

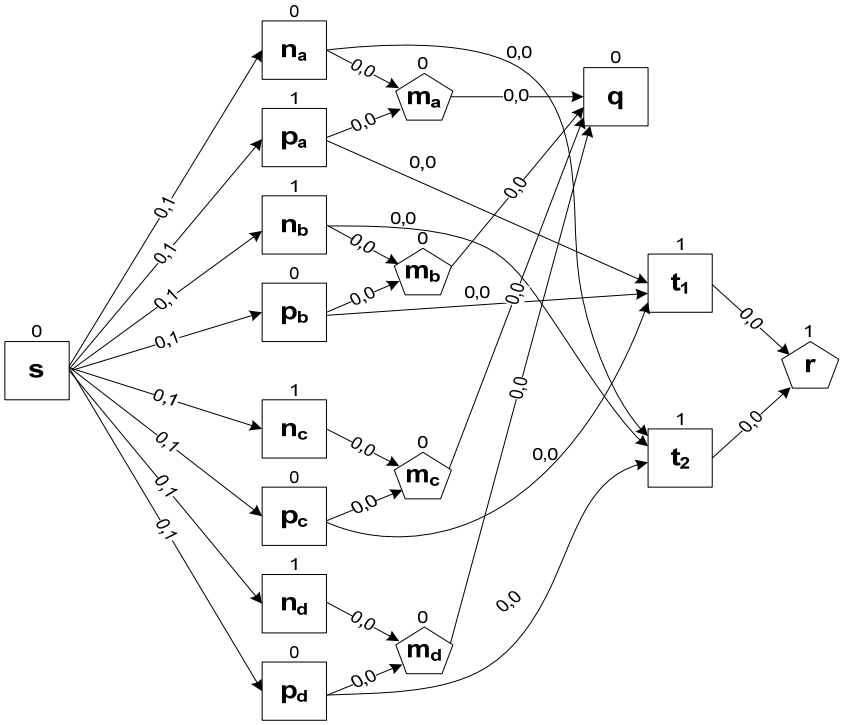

Figure 33: Reduction from 3-SAT to min/max problem

In the graph above, pentagons represent min events and squares represent max events. The numbers depicted over the events indicate a solution of the timing constraints for the satisfiable assignment of the formula φ.

5.2.5 Min-Max-Linear Constraint Systems

The general min-max-linear constraint problem is also NP-hard. This was demonstrated in [62] using a reduction from 0-1 integer programming. This fact justifies a branch and bound approach for resolving such problems.

McMillan and Dill [29] proposed an algorithm where min constraints are eliminated by assuming that one of the min constraints is less than the others. An instance of the max-linear constraint problem is generated by the recursive elimination of all min constraints. The solution is the maximum of the separations for all max-linear sub problems.

T. Y. Yen et al. [26] proposed an algorithm that handles all three constraint types modifying their max-linear algorithm (Algorithm 4 `The MaxSeparation algorithm`). In this algorithm, the min constraints are recursively eliminated as well, as in the McMillan and Dill algorithm, each time generating a max-linear subproblem. If some min constraint has been satisfied by other constraints, it has to be chosen; as for each min event, only the earliest constraint has to be found. Furthermore, when a new sub problem is generated, the algorithm continues with the current status and not from the beginning in order to optimize the calculation process.

5.2.6 Assume-Commit Constraint Systems

Assume-commit constraints were introduced to reflect the input/output nature of events [32, 33, 70]. The maximum separation time computation guarantees that the given system of constraints is consistent, i.e., that it has at least one solution, even though this solution may not be realizable if we take into account the input/output nature of events. The input events are those that cannot be controlled by the system and the timing for these events has to be satisfied for each value in the bounded interval [33]. On the contrary, the output events are under the control of the system and can be constrained in the given interval if needed. Distinguishing the input/output nature of events led to the definition of two constraint types: commit constraints and assume constraints.

Definition 5 [32]

Consider the events e_i and e_j with constraint $C_{ij} = (e_i, e_j, [l_{ij}, u_{ij}])$, where $l_{ij} \le t(e_j) - t(e_i) \le u_{ij}$. C_{ij} is a *commit constraint* if e_j is an output event; otherwise it is an *assume constraint*.

A heuristic method based on the local consistency property was used for solving interface timing specifications with commit-assume constraints [32] and is based on the following reasoning.

Definition 6 [32]

An event (node) is said to be a *convergence event* (*node*) if it has more than one parent.

Definition 7 [32]

Let z be a convergent node of event graph $EG=(E, C)$, and $P(z)$ a set of its parents in EG. Let $EG' = (E', C')$, where $E' = E \setminus \{z\}$, and $C' = C \setminus \{(e_i, z, [l, u]) \in C\}$. The node z is *locally consistent* if $\forall e_1, e_2 \in P(z)$ the maximum separation time s_{12} of e_1, e_2 (respectively s_{21} of e_2, e_1) computed over *EG'* is less than or equal to the maximum separation time s_{12} (s_{21}) computed over the subgraph containing only the set of nodes $\{e_1, e_2, z\}$.

Definition 8 [32]

An event graph, $EG=(E, C)$, is *locally consistent* if each convergent node z is locally consistent.

The local consistency property means that the maximum separation time between each pair of convergence node parents computed over the graph without this node and its corresponding constraints is less than or equal to the maximum separation time computed over the graph formed from the convergence node and its parents.

The idea behind the local consistency property is that the satisfaction of the local consistency condition guarantees the existence of at least one realizable relative schedule for each locally consistent node and, correspondingly, for the locally consistent graph.

In the event graph of Figure 34 (without dashed arcs), nodes i_k (k = 1 to 3) are input events and nodes o_s (s = 1 to 5) are output events.

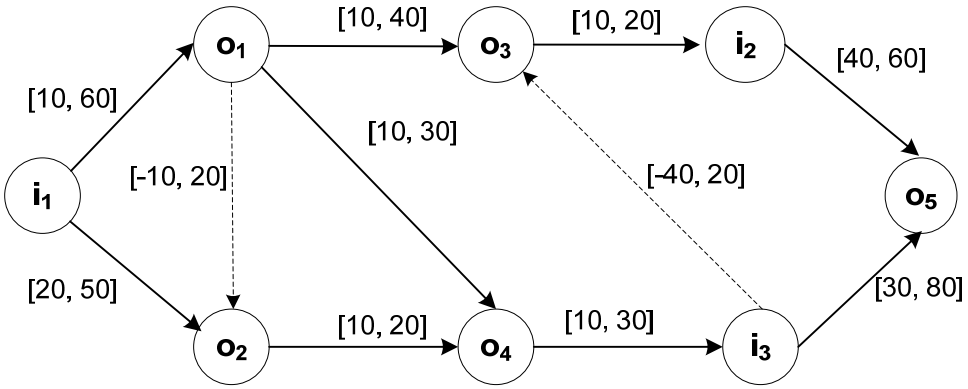

Figure 34: Event graph

This graph is not locally consistent. Two convergent nodes, o_4 and o_5 , do not verify the local consistency property. The algorithm looks for a locally consistent graph. In the case when the initial event graph does not verify the local consistency property, the method determines which commit constraint can be modified or added without altering the given assume constraints.

For example, for node o_5, the maximum separation time between the nodes i_2 and i_3 computed over the graph without this node and its corresponding constraints (Figure 35, (a)) is $-40 \le t(i_3) - t(i_2) \le 40$.

The maximum separation time computed over the graph formed from o_5, i_2 and i_3 (Figure 35, (b)) is $-40 \le t(i_3) - t(i_2) \le 30$.

In the example presented, in order to make the graph locally consistent, two new commit constraints were added (dashed arcs in the Figure 34). The constraint $i_3 \to o_3$ assures the implicit assume constraint between the nodes i_2 and i_3 makes the node o_5 locally consistent. The constraint $o_1 \to o_2$ has been added explicitly to render

o_4 locally consistent as node o_2 is an output event and is considered as being under system control. Each locally consistent node can be removed from scheduling it can always can be scheduled using the As Late As Possible (ALAP) relative schedule. For node o_5: $t\ (o_5) = \min(t(i_2) + 60, t(i_3) + 80)$.

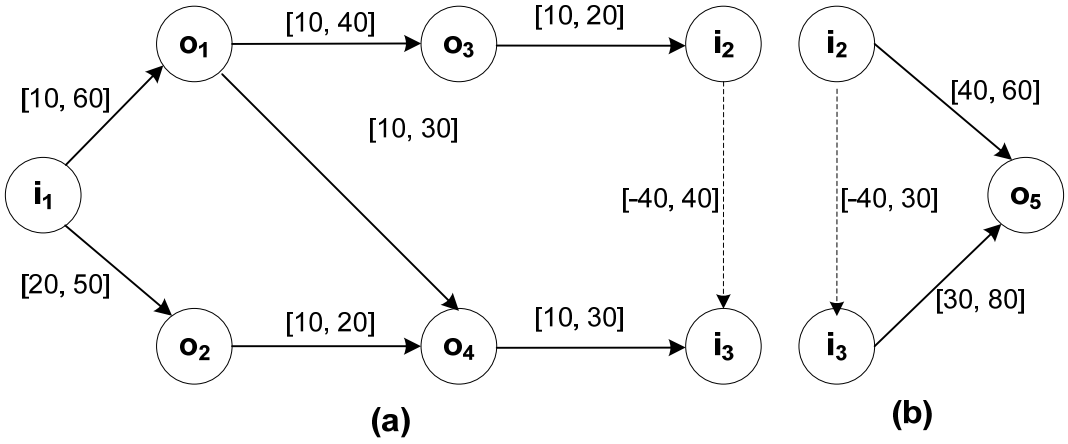

Figure 35: Subgraphs used in the determination of local consistency property for the node o_5.

Algorithm 5: Algorithm for finding a locally consistent event graph

```
Step 1: tighten the event graph

Step 2: sort the list of convergence nodes in a reverse
topological order

Step 3:
for each convergence node do
     Determine its parents
     for each pair of its parents do
          if not locally consistent then
            if possible to add a commit constraint
            then add it
            else find a commit constraint among the
                 upstream ancestors
            end if
            update the list of convergence nodes
          end if
     end for
end for
```

As mentioned earlier, the timing for input events has to be satisfied for each value in the bounded interval, i.e., having multiple assume constraints, we need to satisfy each value from each bounded interval. This means that the assume constraint interpretation corresponds to the type 4 constraints interpretation from Figure 27. In [32], the assume constraint type was considered as a special type of linear constraint though an efficient algorithm to resolve the timing verification problem with linear-assume constraints has not been given. An equivalent set of min-max-linear constraints for the type 4 constraints shown in Figure 36 was proposed in [27].

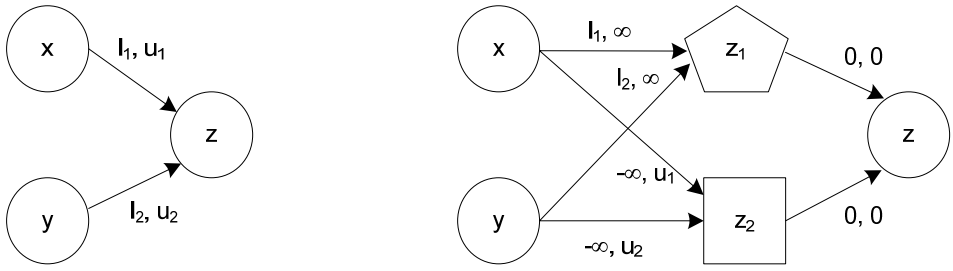

Figure 36: Transformation of type 4 constraints into min-max-linear constraints

The existence of equivalent min-max-linear expression for type 4 constraints and the discovered correspondence between type 4 and assume constraints explain the non-existence of a polynomial time algorithm due to the NP complexity of the general min-max-linear problem [62]. The mathematical interpretation of assume constraints completes the mathematical representation of all constraint types discussed in literature.

5.2.7 Discussion

In previous subsections we have seen graph-based algorithms for timing specification verification. Another approach used to calculate the maximum achievable separations for a temporal constraint specification is based on its reformulation into a mathematical optimization problem. Among the algorithms

using this method, we want to mention the works of T. M. Burks and K. A. Sakallah [62] and Y. Cheng and D.Z. Zheng [30], [31].

T. M. Burks and K. A. Sakallah [62] proposed two methods to solve the min-max-linear constraint problem: a branch-and-bound algorithm in mathematical programming and a transformation method based on the linearization of min-max inequalities into a standard mixed integer linear programming formulation. In both cases, existing solvers were used to cope with reformulated timing specifications. Y. Cheng and D.Z. Zheng [30], [31] mathematically reformulated the Yen et al. [26] algorithm using min-max functions theory.

5.3 Min-max constraint linearization algorithm

In order to cope with a general min-max-linear constraint problem, we propose a new graph-based algorithm. In our approach, we use the standard linearization procedure of min-max constraints as in the work of T. M. Burks and K. A. Sakallah [62], though we interpret it as a graph theory problem thereby obtaining as a result a set of graphs with only linear constraints to which we apply the shortest path algorithm to calculate the maximum separation time for all event pairs.

5.3.1 Min-max constraint linearization

The min value z of two integer numbers x and y, $z = min\{x, y\}$, or the max value $z = max\{x, y\}$ can be found by resolving a set of linear inequalities (Figure 37), where α_1, α_2 are binary variables and M is a suitably large positive constant.

$z = \min(x, y)$	$z = \max(x, y)$
$z \le x$	$z \ge x$
$z \le y$	$z \ge y$
$z \ge x - M\alpha_1$	$z \le x + M\alpha_2$
$z \ge y - M(1-\alpha_1)$	$z \le y + M(1-\alpha_2)$
$M \ge \lvert x - y \rvert$	$M \ge \lvert x - y \rvert$

Figure 37: Min-Max linearization inequalities

Consider the subgraph shown in Figure 38 containing a max event t_k having two predecessors t_i and t_j, and two max constraints: $c_{ik} = (t_i, t_k, [l_i, u_i])$, $c_{jk} = (t_j, t_k, [l_j, u_j])$.

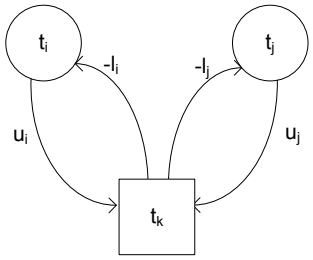

Figure 38: Subgraph with the max constraints

The firing time of the event t_k determined by two max-constraints is:

$$\max(t_i + l_i, t_j + l_j) \le t_k \le \max(t_i + u_i, t_j + u_j)$$

The left part of the inequality is the same as for the linear solution, i.e., we have to satisfy all lower bounds simultaneously. This means that we can leave the lower bounds without any transformation in the graph. To find a greatest upper bound in order to resolve the right part of the inequality shown above, we rewrite it using the linearization procedure presented in Figure 37 and obtain a set of linear inequalities shown in Figure 39.

$$\begin{aligned} & t_k \le z = \max(t_i + u_i, t_j + u_j) \\ & z \ge t_i + u_i \\ & z \ge t_j + u_j \\ & z \le t_i + u_i + M\alpha \\ & z \le t_j + u_j + M(1-\alpha) \end{aligned}$$

Figure 39: Transformed max constraint

We can observe that all inequalities have only two unknown variables for a given constant M and a given value of the binary variable α (Figure 39). A solution of such a linear system of inequalities can be found by a shortest path algorithm applied to a corresponding graph. We can transform the min constraints to a set of linear

inequalities in a similar way as for the max constraints. The firing time of the event t_k determined by two min constraints is:

$$\min(t_i + l_i, t_j + l_j) \le t_k \le \min(t_i + u_i, t_j + u_j)$$

We transform the left part of the above expression into a set of linear inequalities as shown in Figure 40.

$$\begin{aligned}
&\min(t_i + l_i, t_j + l_j) = z \le t_k \\
&z \le t_i + l_i \\
&z \le t_j + l_j \\
&z \ge t_i + l_i - M\alpha \\
&z \ge t_j + l_j - M(1-\alpha)
\end{aligned}$$

Figure 40: Transformed min constraint

The graph interpretation of the transformed min and max constraints of Figure 39 and Figure 40 is given in Figure 41.

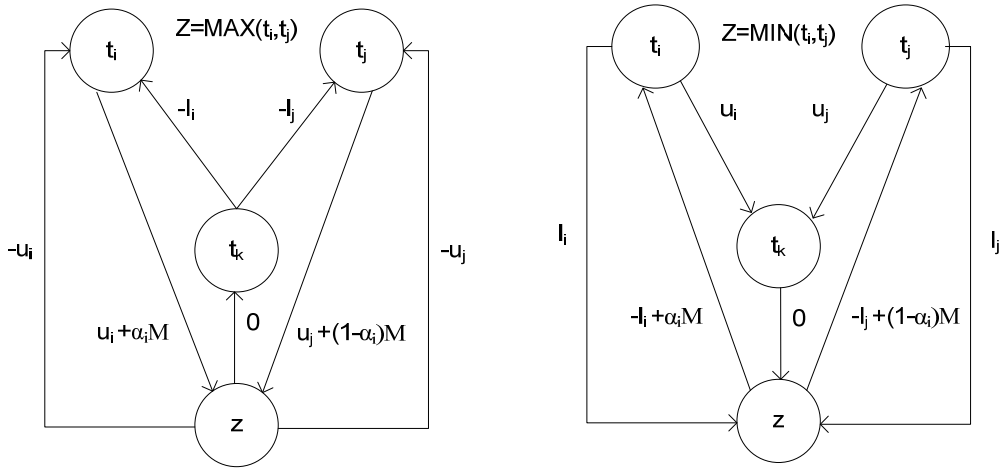

Figure 41: Graph representation of the transformed max and min constraints

According to the last inequality of Figure 37, the constant M in the linearization procedure of Figure 39 has to verify $M \ge \left|t_i + u_i - t_j - u_j\right|$ for max constraints and $M \ge \left|t_i + l_i - t_j - l_j\right|$ for min constraints. We have also the following system of inequalities:

$$\left|t_i + u_i - t_j - u_j\right| \le \left|t_i - t_j\right| + \left|u_i + u_j\right| \le \left|s_{ji}\right| + \left|u_i\right| + \left|u_j\right|$$

$$\left|t_i + l_i - t_j - l_j\right| \le \left|t_i - t_j\right| + \left|l_i + l_j\right| \le \left|s_{ji}\right| + \left|u_i\right| + \left|u_j\right| \text{ since } l_i \le u_i \text{ and } l_j \le u_j$$

The constant M has to be larger than separation between the nodes t_j and t_i: for max-only constraint systems the maximum separation between each pair of nodes t_j and t_i, as we have seen, is $\max_m(t_m + l_m) \le t_k \le \max_m(t_m + u_m)$, where m denotes all parents of the node t_k. In other words, a maximum separation calculated in the graph where all constraints are considered as being max along upper bounds will be greater than the corresponding maximum separation for the general constraint system. An efficient polynomial time algorithm for systems with only max constraints [61] was used to obtain the separation between all nodes t_j and t_i. This value was then employed to obtain an approximate upper bound for M. Thus, for min and max constraints we take the constant $M = \left|s_{ji}\right|_{\max only} + u_i + u_j \quad (u_i, u_j \ge 0)$ in the case of finite bounds. If one or more values in the constant M's expression are equal ∞, we will consider that M is equal ∞ and we represent it by an arbitrary number larger than any sum of the absolute values of the finite bounds.

A max and min function of more than two variables can be replaced by a composition of two-variable min, max functions. Therefore, from now on, we will consider two variable min-max constraints.

Algorithm 6: Min-max constraint linearization algorithm

```
Step 1: Calculate the maximum separations considering all
constraints being max type

Step 2: Calculate the number of min-max events and form
binary vector of parameters αi

Step 3: Construct an intermediate graph where each min
and max node are transformed into "linear" one

Step 4:
repeat
     for each min or max node do
          Calculate the constant value M
```

```
            Adjust the weights on the corresponding edges
            according to the binary vector value
      end for
      Calculate the maximum separations using Bellman-Ford
      algorithm
      if linear solution exists then
            for all i,j do
                  s_ij←max(previous(s_ij),current(s_ij))
            end for
      end if
until all binary combinations of α_i represented by the
binary vector have been explored
```

Example

In the above presented algorithm, we illustrate an example taken from [26].

In Figure 42 we have a graph with two max events, *c* and *f*. There are four max constraints, $a \rightarrow c$, $b \rightarrow c$, $d \rightarrow f$ and $e \rightarrow f$.

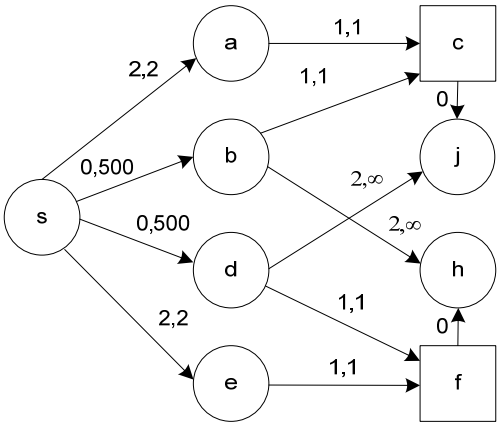

Figure 42: Initial graph with max constraints

We transformed the two max events in the initial graph in the manner presented by Figure 41 thus obtaining the "parameterized" graph presented in Figure 43. The parameters in the graph are the binary values of variables α_i. The arcs whose weight has to be changed according to the binary values α_i are represented by the dashed lines in Figure 43. For each binary value we have to find the linear solution, if it exists, and take the maximum values of the separations obtained during calculations.

The complexity of the presented algorithm is exponential in terms of min-max events. If *m* is a number of min-max events, each with two constraints, we have

to explore, in the worst case, 2^m subproblems having only linear constraints. Thus, the algorithm's complexity is $2^m n^3$ where n is the number of graph nodes and m is the number of min-max events.

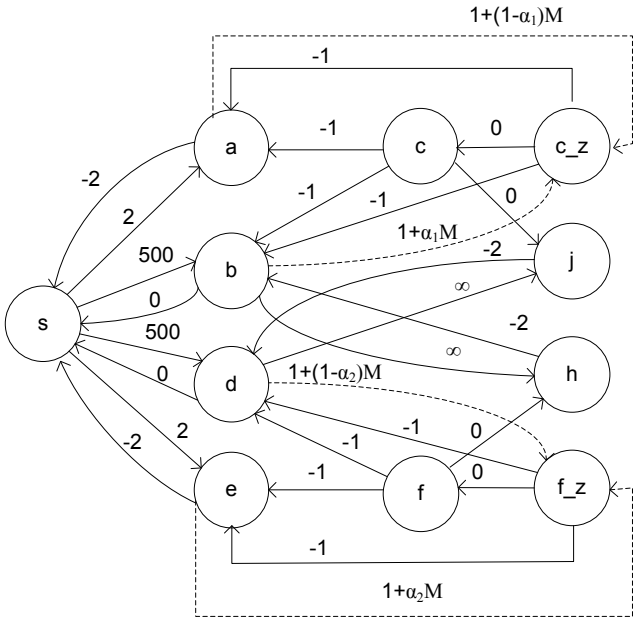

Figure 43: Graph with transformed max constraints

5.3.2 Algorithm Optimization

In order to reduce the number of linear solution explorations, we propose to employ a procedure used in T.Y.Yen et al.'s algorithm [26] such that we may eliminate unsatisfied upper bounds of max constraints or unsatisfied lower bounds of min constraints, as, for both constraint types, only one bound has to be satisfied, namely the earliest one for min constraints and the latest one for max constraints.

We have presented T. Y. Yen et al.'s algorithm [26] in Section 5.2.3. The general idea of this algorithm consists of two steps:

1. Satisfaction of all compulsory bounds and obtaining the smallest separation values that satisfy the compulsory bounds. For max events all lower bounds have to be satisfied, for min, all upper bounds and for linear constraints, both.

2. Construction of the slack graph with values defining an amount by which the bound value can be tightened and with iterative relaxation of the separations according to the slacks.

In our algorithm, in order to eliminate the unsatisfied bounds, we construct a compulsory graph and calculate the smallest separations. Following this step, we construct the slack graph. If the initial slack values are negative, the corresponding bounds cannot be tightened and, in the context of our algorithm, we do not need to explore the corresponding binary combinations for α_i.

For the example shown in Figure 42, we have obtained negative slacks for the max constraints $b \rightarrow c$ and $d \rightarrow f$. This means that constraints $b \rightarrow c\_z$ and $d \rightarrow f\_z$ (Figure 43) cannot be tightened and the values of $\alpha_1 = 1$ and $\alpha_2 = 0$ have not been explored, thus leading to the verification of only one parameter combination: $\alpha_1 = 0$ and $\alpha_2 = 1$ instead of four binary combinations. The linear solution corresponding to the assignments $\alpha_1 = 0$ and $\alpha_2 = 1$ in the parameterized graph (Figure 43) gives us the solution to the initial problem.

With this kind of optimization, the worst case complexity stays the same, as, sometimes in the graph, all bounds for min-max constraints can give some amount by which the bounds can be tightened. We thus have to find the one that provides the largest or smallest value for the max or min constraint correspondingly, therefore exploring all binary assignments.

5.3.3 Experimentations

We have applied the presented algorithm with optimization to the graph of timing specification shown in Figure 42. We have obtained results exploring only one linear solution. For comparison, Yen et al.'s algorithm requires two iterations and McMillan and Dill's almost 500 in order to get the results [26].

The second example that we have used is the Intel 8086 ROM read cycle from [28]. The exploring system, shown in Figure 44, contains a clock generator, an address decoder and an address latch. The clock generator emits a clock signal with a period of 204 ns. The latch holds the address and has a delay of [0, 12] ns. The

address decoder has a delay of [0, 30] ns and ensures that only the selected PROM outputs data into the bus at any given time. The designer's problem is to verify whether or not the 2716 PROM is fast enough to work with an 8086 having a clock period of 204 ns. In timing terms, this means that we have to verify the following timing requirements:

$$d_1 \rightarrow d_2 = [0, \infty] \qquad A_1 \rightarrow A_2 = [0, \infty]$$
$$c_3 \rightarrow d_2 = [10, \infty] \qquad a_2 \rightarrow d_1 = [0, \infty]$$
$$R_1 \rightarrow R_2 = [0, \infty] \qquad d_1 \rightarrow c_3 = [30, \infty]$$

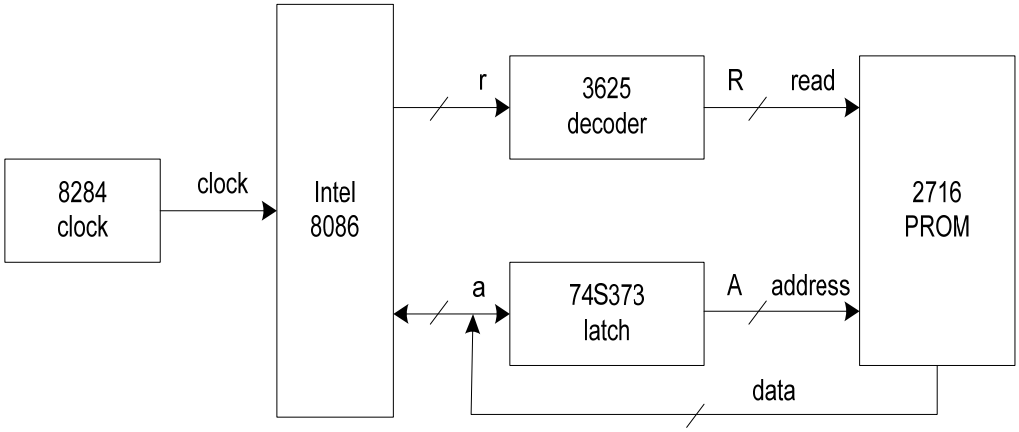

Figure 44: Intel 8086 ROM read cycle

The timing specification of this example consists of 13 events among which there are one min event, one max event and the rest are linear events, as shown in Figure 45. Events c_1, c_2, c_3 are clock transitions; a_1 and a_2 are, respectively, the beginning and the end of a valid address on the data/address bus. The events A_1 and A_2 are the beginning and end of a valid address at the address latch outputs. The events r_1/R_1 and r_2/R_2 are, respectively, the beginning and the end of the read signal of the address decoder output. A min event, d_2, denotes the end of valid data on the data/address bus. This event has to occur as soon as either the address or the read signal is removed. A max event, d_1, is the start of valid data on the data/address bus which depends on the later of the two input signals.

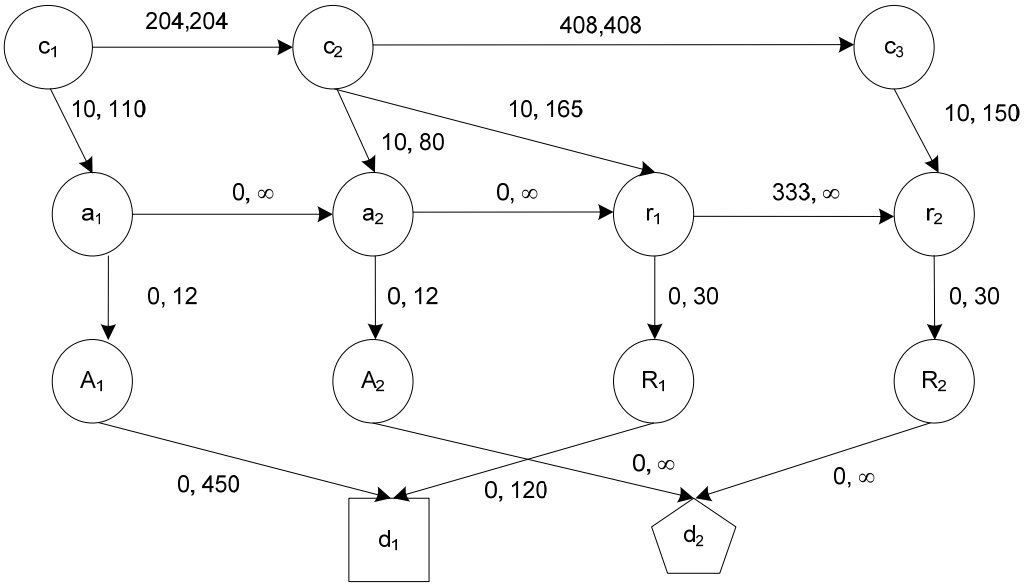

Figure 45: Timing specification of the Intel 8086 ROM read cycle

In order to verify the timing requirements we have applied the algorithm to the system timing specification (Figure 45) and calculated actual timing separations (Table V).

Table V: Timing separations for the Intel 8086 ROM read cycle

	c1	**c2**	**c3**	**a1**	**a2**	**r1**	**r2**	**A1**	**A2**	**R1**	**R2**	**d1**	**d2**
c1	0	204	612	110	284	369	762	122	296	399	792	572	∞
c2	-204	0	408	-94	80	165	558	-82	92	195	588	368	∞
c3	-612	-408	0	-502	-328	-243	150	-490	-316	-213	180	-40	∞
a1	-10	194	602	0	274	359	752	12	286	389	782	509	∞
a2	-214	-10	398	-104	0	155	548	-92	12	185	578	358	∞
r1	-214	-10	398	-104	0	0	548	-92	12	30	578	358	∞
r2	-622	-418	-10	-512	-338	-333	0	-500	-326	-303	30	-50	∞
A1	-10	194	602	0	274	359	752	0	286	389	782	509	∞
A2	-214	-10	398	-104	0	155	548	-92	0	185	578	358	∞
R1	-214	-10	398	-104	0	0	548	-92	12	0	578	358	∞
R2	-622	-418	-10	-512	-338	-333	0	-500	-326	-303	0	-50	∞
d1	-214	-10	398	-104	0	0	548	-92	12	0	578	0	∞
d2	-214	-10	398	-104	0	155	548	-92	12	185	578	358	0

The results are the same as presented in [28]. As we can see several timing requirements are violated. A comparison of the actual separations computed by the algorithm and the required separations is given in the Table VI.

Table VI: Required and computed separations for the Intel 8086 ROM read cycle

Constraints	Required time intervals	Actual timing separations, $\lfloor -s_{ji}, s_{ij} \rfloor$
$d_1 \rightarrow d_2$	[0,∞]	[-358,∞]
$c_3 \rightarrow d_2$	[10, ∞]	[-398,∞]
$R_1 \rightarrow R_2$	[0,∞]	[303,578]
$A_1 \rightarrow A_2$	[0,∞]	[92,286]
$a_2 \rightarrow d_1$	[0,∞]	[0,358]
$d_1 \rightarrow c_3$	[30,∞]	[40,398]

In our algorithm, for timing specification with one max and one min event, we have to explore four linear solutions. By applying the above discussed optimization technique we have reduced the number of linear solution explorations to two.

In order to estimate the average complexity of the proposed algorithm, we have performed several numerical experiments. As there is not enough real data for the experiments, we have decided, like other researchers [30, 31], to automatically generate the timing specification graphs, which should be acyclic. In order to respect this condition, we have used an initial acyclic graph having a solution (Figure 45) for the generation of larger graphs. We have generated graphs in three manners, resulting in obtaining graphs of one of three types, as presented in Figure 46.

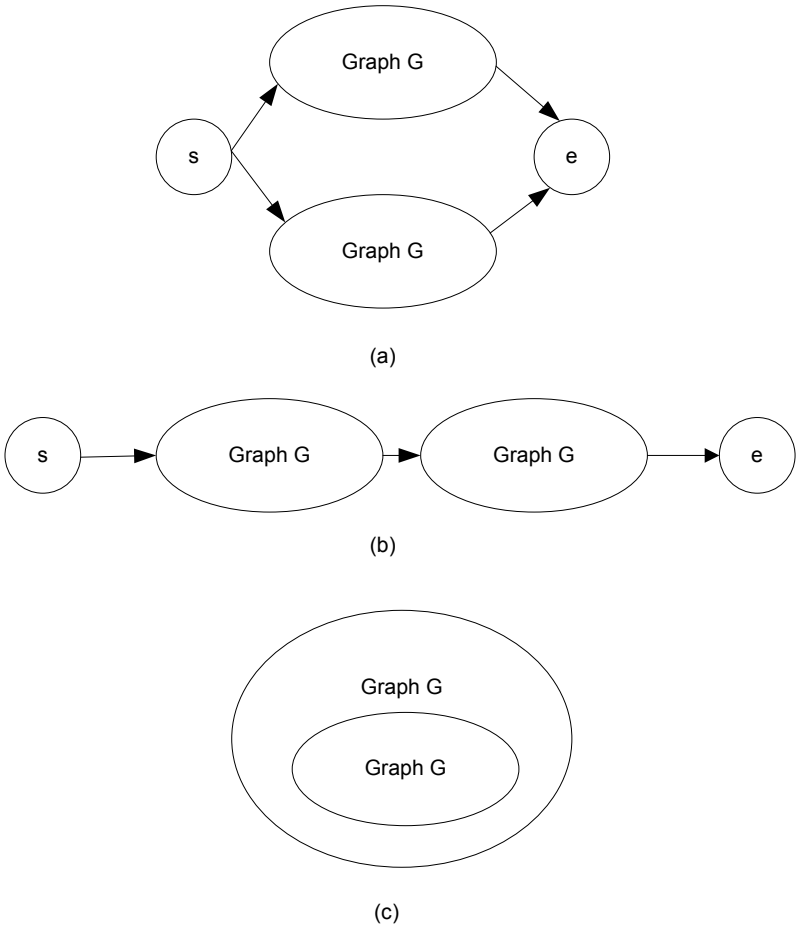

Figure 46: Structures of the generated graphs

In Figure 46 (a), the initial graph is merged with other graphs of identical structure in a parallel manner; in (b) the merging is done in a sequential manner and in (c) one of the initial graph nodes is replaced by a graph with the same structure. In all cases, the values of arc weights are randomly generated. In our experiments, we wanted to compare the linearization algorithm's efficiency with that of T.Y. Yen et al's [26]. The authors conjecture that the complexity of their algorithm is O(VE $+V^2\log(V)$) when it is trying to resolve a max-linear problem. On the other hand, for min-max-linear problem the algorithm has to be modified and its complexity is exponential in the worst case. Our algorithm treats all three types of constraints in optimized and non optimized versions. However, to compare our algorithm with the most efficient version of the T.Y. Yen et al's algorithm, we have decided to use in our experimentations graphs with only linear and max constraints.

We have limited the number of nodes to approximately 150 as it is difficult to generate a large graph with a non cyclic structure having a solution. The hardware platform for the experiments was a computer with an Intel Pentium 1.5 GHz CPU and 512 Mb of RAM. The results were measured in number of ticks, which is the smallest unit available to measure time and is equal to 100 nanoseconds. We have converted the tick measures to milliseconds. The results are given in Table VII, which lists the number of constraints representing the number of arcs in the graphs, the total number of nodes in the graphs and the number of max nodes among them. The next three columns contain the costs, in milliseconds, for resolving max-linear problems of different size and complexity for three algorithms: T. Y. Yen et al.'s, the Linearization algorithm in the non optimized version and the Linearization algorithm in the optimized version. As the complexity of the linearization algorithm is exponential in terms of the number of min/max nodes, more precisely $O(2^m n^3)$, where m is the number of min/max nodes and n the total number of nodes in a graph, for certain experiments we do not have the numerical data. For example, for experiment number 4 (b) with 21 max nodes, we have to explore 2^{21} linear solutions, leading to very long execution delays. In such cases, we present the timing for the optimized version of the algorithm.

Table VII: Results of experiments

Graph			**CPU time**		
Constraints		**Nodes/Max nodes**	**Algorithm of T.Y. Yen et al., in ms**	**Linearization Algorithm, in ms**	**Linearization Algorithm with optimization, in ms**
1.	17	13/2	30	0 (< 100 ns)	0 (< 100 ns)
2.	(a) 40	30/7	50	400	0 (< 100 ns)
	(b)38	27/7	40	360	0 (< 100 ns)
	(c)38	27/7	40	400	0 (< 100 ns)
3.	(a)84	58/17	360	1698091	881
	(b)76	53/16	250	721056	370
	(c)78	53/15	280	354840	360
4.	(a)168	114/12 114/13	3054 3094	290758 595606	40 40
	(b)158	110/21 110/13	2393 2513	- 277218	1261 60

	(c)160	107/21 107/16	2343 2413	- 2297864	1241 80
5.	(a)228	152/17	7180	18615658	70
	(b)218	148/18	6109	17256613	140
	(c)220	145/20	5848	-	140

The data presented in the table show that the optimization of the linearization algorithm significantly reduces the cost incurred by its non optimized version. Furthermore, in most cases, the optimized version of our algorithm is better than the max-linear version of T. Y. Yen et al.'s. In [30, 31], the experimental results for T. Y. Yen et al.'s algorithm are presented and mathematically reformulated using min-max functions. It is quite difficult to compare these results to our data due to the different nature of representations and as a consequence of the experimental parameters. However, for the experiment in [31] with parameters corresponding to 170 constraints and 10 min/max nodes, the execution time is around 50 milliseconds whereas, in our case, the execution time is between 40 to 80 milliseconds depending on graph structure. Thus, the complexity of both algorithms is comparable.

The examples presented demonstrate that regardless of the exponential worst case complexity, the proposed optimized linearization algorithm is quite efficient.

5.4 Timing in TLM

In Transaction Level Modeling, the first level where some quantity of timing information is added to the system description is the CP+T level. At this level we have to extract, from the system specification information and perhaps from the CP simulation results, the first global temporal system model. All architectural constraints can be presented by their corresponding temporal models and can be verified analytically using timing verification algorithms and then again verified with refining functional details by simulation. The Hardware/Software partitioning process, if needed, can be guided by criteria of temporal constraints realizability. At the PV and PV+T levels, the communication structure can be explored and verified using timing analysis.

In this subsection, we demonstrate the way in which the presented timing expression methodology can be applied at different TLM abstraction levels.

5.4.1 Timing modeling at CP+T level

We will demonstrate an application of the proposed timing expression methodology in the exploration of the CP+T TLM models in the design of an audio-video server system first described in Section 4.6. For exploration at the CP+T level, we use a CP audio-video server system model. From the information collected at the CP level, we have manually created the temporal server model shown in Figure 47. As shown in the aforementioned figure, events appearing on one server channel are presented with its temporal relationships. This CP+T temporal model of the audio-video server system is explored and has to be refined. As we can see, the server's temporal model contains several conditional events such as user commands. At the CP level these commands are modeled using probabilistic generation. Furthermore, some parameters are defined by their variation intervals, leading to the specification of timing intervals in a parameterized manner. Thus, the server's behavior is quite complex and the corresponding temporal graph includes cyclic treatment, parameterized borders of the delay intervals and several conditional events describing nondeterministic behavior. These concepts are not supported by the previously presented timing expression methodology, which is based on timing constraint analysis. For this case, we have tried to combine the simulation and the temporal verification. In order to manage the exploration of the systems with conditional events and repetitive behavior, we can use dynamic verification that combines the simulation and analytical methods. We can subdivide the graph with cyclic behavior into acyclic subgraphs and explore several of them by analytical methods in the parameterized form. Having the verified subgraphs with timing intervals presented in parametric form, we can, during the simulation, dynamically extract the corresponding subgraphs, substitute the simulation values and verify the system requirements. In this manner we can verify and adjust temporal constraints in the temporal model and add the behavior details in the simulation model.

For the audio-video server model, we have extracted several parameterized acyclic subgraphs, though all of them gave us trivial solutions. This fact signifies that each acyclic subgraph in the temporal audio-video server model is not detailed enough to provide information for temporal analysis. In addition, this also means that the whole CP+T temporal graph is very complex for the proposed temporal expression methodology.

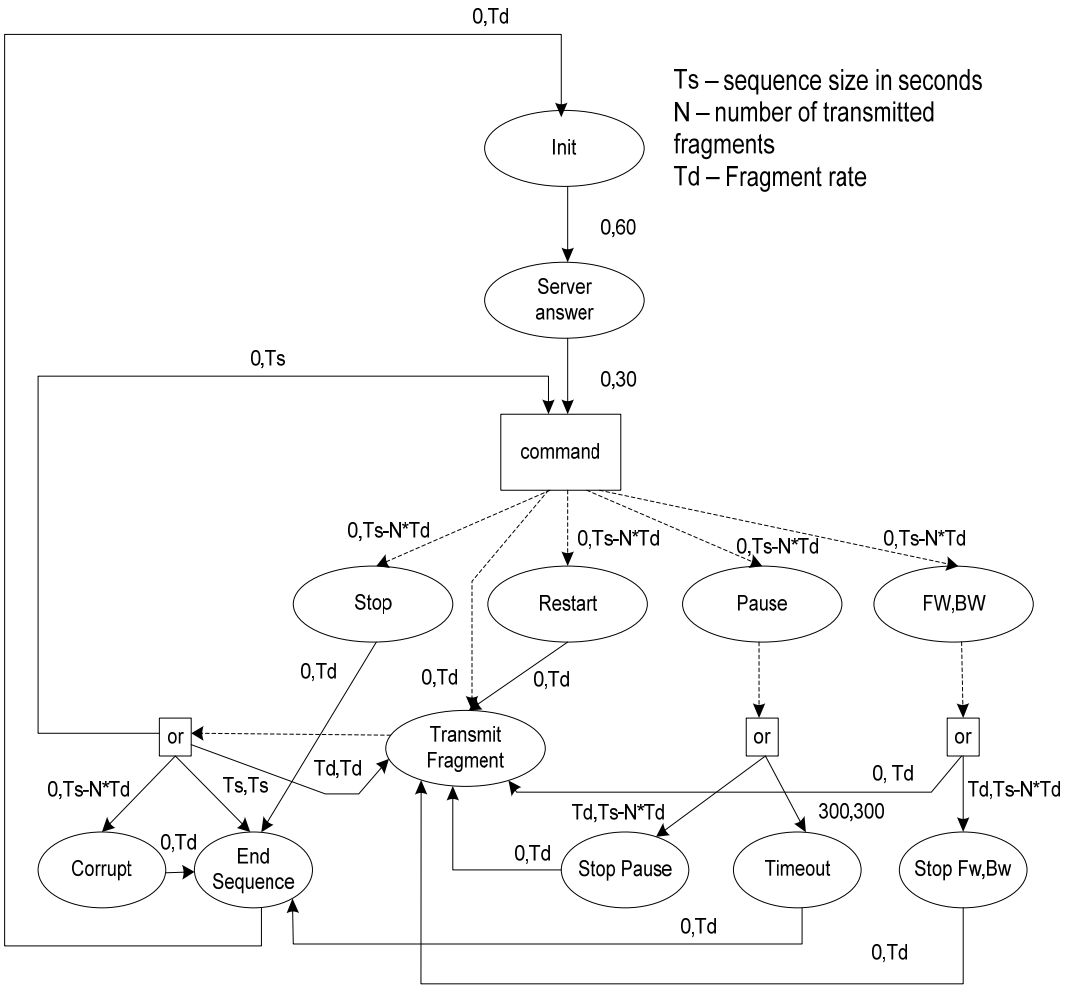

Figure 47: Temporal model of the audio-video server system

Our experimentations have demonstrated that the timing constraint analysis is probably not very appropriate for exploration of high level system descriptions and for applications with nondeterministic and repetitive behavior. Indeed, for such systems, the simulation method will be sufficient in providing acceptable simulation

speed as there are very little amounts of details regarding system description at the CP level.

5.4.2 Communication Exploration at PV and PV+T Levels

The TLM paradigm assumes development of system computation structure separately from development of communication structure in the design flow. In this subsection, we present a method which uses the min-max-linear temporal model to explore and refine the system communication structure in the TLM design flow. Detailed communication structure exploration and refinement is done at the PV and PV+T abstraction levels. The presented method is an extension and generalization of a heuristic algorithm based on the local consistency property for assume-commit constraints systems [32] discussed in Section 5.2.6.

Structurally, modern systems can be seen as one or several processing components which communicate with each other or/and the environment. The idea of distinguishing the event's nature leads to the identification of events that can be controlled by some component or that can be grouped according to certain functionality for future implementation as a unique component in the system. A "subdivision" of the global temporal system specification into regions of events belonging to some structural or logical unit gives us the possibility to verify the temporal interactions between the communicating elements and eventually propose a set of rules thereby defining a protocol of communication assuring the timing consistency of the system specification. This verification mechanism can be applied at different abstraction levels providing, in this way, the exploration and refinement of the system communication infrastructure. In the case of platform-based modeling, this method can help to adjust several configuration parameters.

Consider the event graph of Figure 48 that represents the temporal specification of two communicating components. Suppose that in this graph the events c_{1i} (i=1 to 5) are those of communicating component 1 that can be controlled by it. Correspondingly, c_{2j} (j = 1, 2) are the events controlled by communicating component 2. The constraints represented by the arcs finishing at these events are commit constraints.

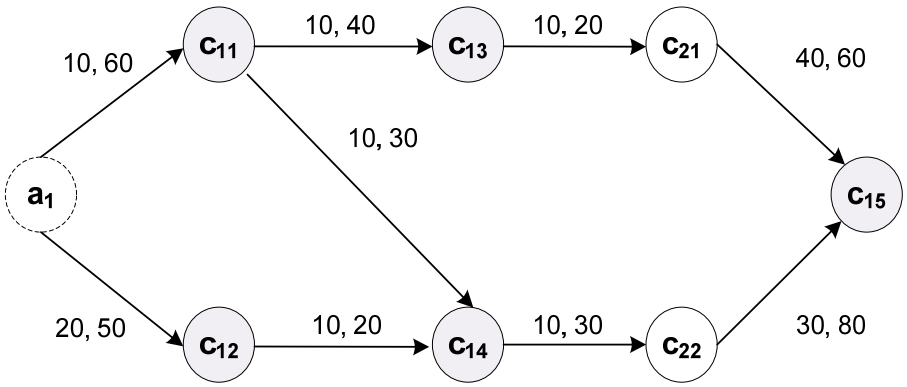

Figure 48: Temporal specification of two communicating components

There is only one event that cannot be controlled in Figure 48, an event source labeled a_1. Events that cannot be controlled in the system timing specification are the "true" input events which form assume constraints. In the temporal system specification, if a certain component cannot control an event associated with it, the nature of this event must be defined as being "in". Determining this nature has to be done judiciously because each communicating component can have several uncontrolled events. In this case, assume constraints can be generated with environment events that are generally uncontrollable.

Applying the local consistency verification algorithm (Section 5.2.6) to the graph in Figure 48, and considering all constraints to be linear, we have obtained a solution which is presented in Figure 49. In the initial event graph, nodes c_{14} and c_{15} are not locally consistent. To change this, we can examine the parents of these nodes and, since they are events belonging to the same logical partition, i.e. they are controlled by the same communicating component, we can add the corresponding commit constraints. Thus, we have generated timing relationships between two communicating devices which can be thought of as a protocol of communication that has to be respected to assure temporally correct system functionality.

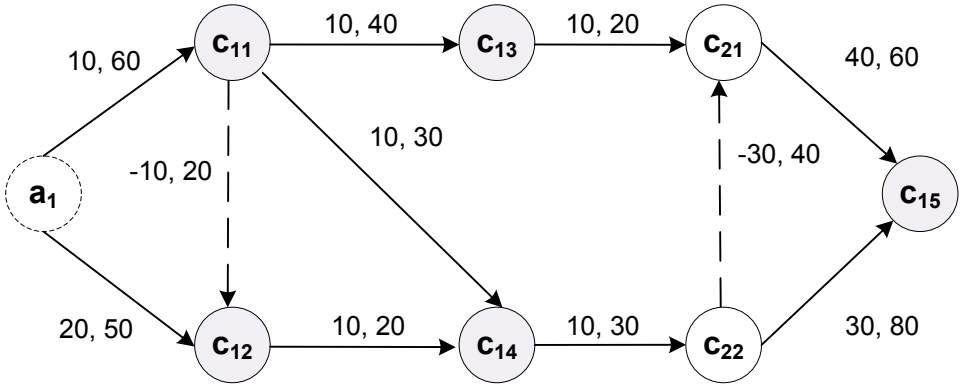

Figure 49: Solution for the temporal specification of Figure 48

In case of presence of multiple uncontrolled events, the assume constraints generated by them have to be satisfied for all values of bounded intervals. To satisfy the assume constraints, the algorithm looks for a candidate for a new commit constraint among the events belonging to the same logical unit.

Let us now consider the fact that component 2 generates only one event, c_{21} (Figure 50).

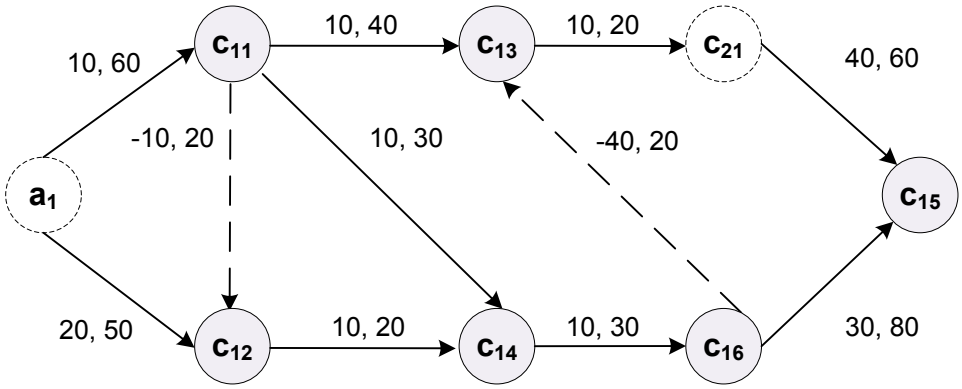

Figure 50: Changed temporal specification of Figure 48

In trying to make node c_{15} locally consistent, we should enforce the implicit assume constraint between c_{21} and c_{16} to be in [-30, 40] by means of inserting the corresponding commit constraint in the appropriate logical unit. The new commit constraint is [-40, 20], inserted between events c_{16} and c_{13} which are under the control of communicating component 1. A pseudo code of the algorithm for finding a realizable timing specification of multiple communicating components is presented below.

Algorithm 7: Algorithm for finding a realizable timing specification of multiple communicating components

Step 1: Annotate in the timing specification graph events that can be controlled by the same communicating component and "true" input events

```
Step 2:
for each convergence node do
     for each pair of its parents do
          verify the local consistency property
          if the condition does not hold then
               if the pair of convergence node parents
                  belong to the same component
               then
                  add or adjust the commit constraint
               else
                  look for a necessary commit constraint
                 in the corresponding component
               end if
          else
               signal timing inconsistency
          end if
     end for
end for
```

5.4.2.1 Experimentations

In this section, we apply the above presented concepts to the timing of bus arbitration [28] example.

Buses are basic blocks of complex digital systems and often cause some difficult timing problems. Considering the timing specification of the multi-master system configuration on an Intel Multibus, we derive a realizable bus arbitration protocol.

The aforementioned multi-master system configuration involves three masters: *A*, *B*, *C*. Each master has a distinct priority. Its resolution is handled by the parallel priority resolution scheme implemented by means of encoder/decoder logic on Master *A*. In this example, the bus arbitration concerns Masters *B* and *C*, this has the lowest priority.

Consider the following situation. Master *C* is transferring data and *B* requests a bus data transfer. The priority resolution logic at *A* asserts and removes the corresponding signals. *C* is allowed to complete the transfer. Thereafter, Master *C* surrenders the bus and *B* prepares for and begins the transfer. The relevant signals and events involved in Multibus arbitration with parallel priority resolution are presented in Figure 51.

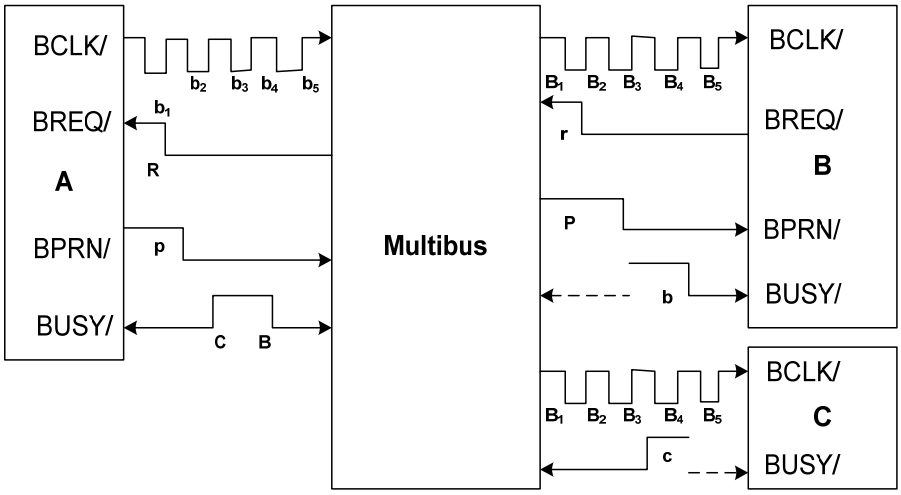

Figure 51: Events and signals involved in arbitration [28]

The bus arbitration timing specification for a minimum bus clock of 100 ns (the maximum bus transfer rate) derived from Intel's data sheets is presented in [28] and is given in Figure 52. In this temporal specification, we can distinguish four groups of events. "True" input events are clock events represented in Figure 52 by dashed lined white circles. The second group of events contains those controlled by the bus: *R*, *P*, *c*, *b*. The third group contains only one event, *r*, which is the request event sent by Master *B* to access the bus for data transfer. Finally, events *p*, *C*, *B* are controlled by the priority resolution logic and are forming the fourth event group. Dashed arcs represent the constraints involved in the arbitration process. Now we can explore the temporal interactions between communicating devices and, if needed, constrain the temporal specifications inside each group of events and between groups thereby generating a realizable bus arbitration protocol. In this example, all temporal constraints are linear.

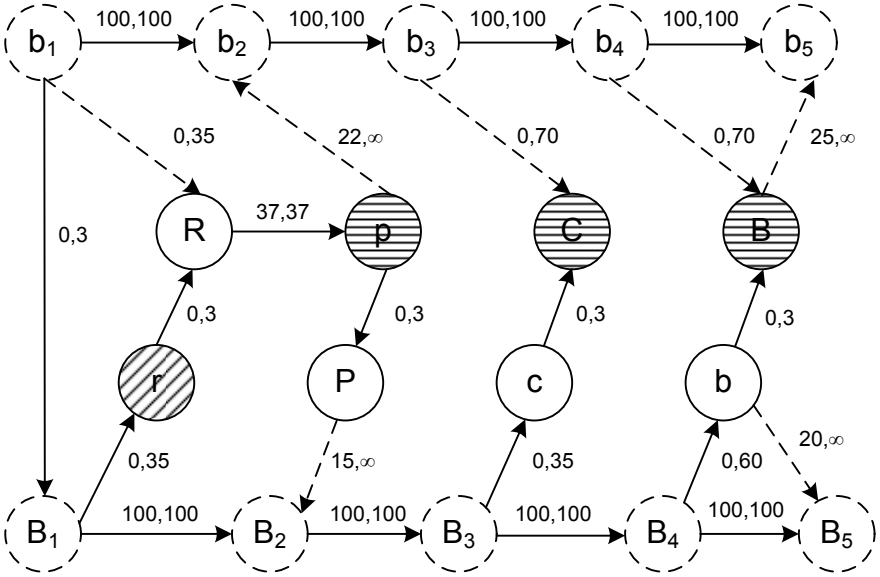

Figure 52: Timing specification of the bus arbitration

We have applied to the graph of Figure 52 the local consistency property method. The results are given in Figure 53.

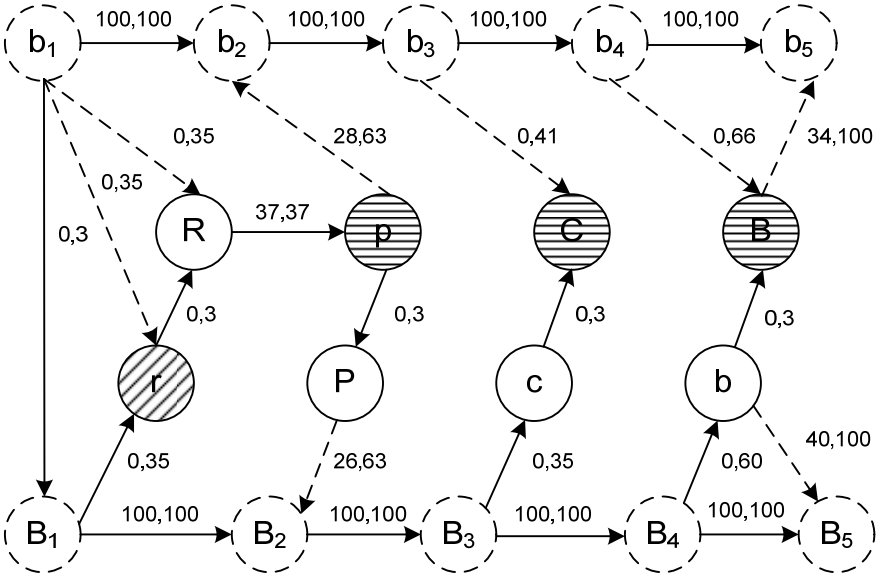

Figure 53: Realizable bus arbitration timing specification

The timing depicted on the dashed arcs represents a realizable bus arbitration protocol on the Intel Multibus system. As we can see, in order to satisfy a priority resolution delay of 37 ns (constraint $R \rightarrow p$), we have to add a constraint to the clock-request delay ($b_1 \rightarrow r$). If we want to avoid this, faster priority resolution logic has to

be chosen. In [28], the timing for the constraint $R \rightarrow p$ was calculated manually using the specification of worst-case bounds on the corresponding signals and the constraint violation was determined; another timing violation was detected using the simulation method. In both cases, timing violations were discovered, but no solution was given. In applying our method, we derived a protocol for communicating components that can satisfy a priority resolution delay of 37 ns thus reducing the temporal bounds of the events that can be controlled by the system.

5.4.3 Conclusion

In this chapter, we have presented an analytical timing model and the way in which it may be used in transaction modeling. This allowed for enhancing the TLM verification simulation method with the analytical one. The presented timing model is based on temporal constraint analysis for the *min-max-linear* and *commit/assume* constraint types. A new timing verification algorithm was proposed as well as an optimization technique to improve the algorithm efficiency. This algorithm was used in combination with the TLM simulation method to explore the CP+T model of the server application, though, for this kind of application, the presented timing model was quite restricted to the exploration of the functional specification at a high modeling level. In contrast, for late phase CP+T level architecture explorations involving HW modules and exploration of Hw/Sw partitioning, this timing verification model is very suitable as it matches temporal hardware module behavior. We have estimated the average verification algorithm performance and showed that it is acceptable. Furthermore, a second algorithm was proposed for PV+T communication structure explorations and its successful application was demonstrated through the bus arbitration example as the temporal inconsistency was not only discovered but the solution was also proposed. Finally, we can predict that the most important impact of the proposed timing verification methods on system exploration performance will be at the late phase of the PV+T level, where many HW components are involved and the simulation speed drastically decreases.

Chapter 6. Conclusions and future work

6.1 Conclusions

In this research, we have presented a timing expression methodology in Transaction Level Modeling, providing a method to accelerate the overall design process. This acceleration can be achieved by means of a combination of two paradigms: a simulation TLM approach on one hand and an analytical timing expression methodology to specify timing between different transactions on the other. Two verification methods (simulation and analytical verification algorithms) provide an excellent solution to the verification and design space exploration of large designs. Indeed, while several parts can be verified and explored analytically, others require simulations and some need both methods. The proposed timing expression methodology is based on previous works and is enhanced with several important theoretical aspects. We have completed the mathematical representation of all temporal constraint types discussed in literature by means of discovering the correspondence between min-max-linear and assume constraints semantics. The assume constraint type can be modeled as min and max constraints. It corresponds to the fourth constraint type that was considered as being without any practical usage. This fact explained the non-existence of a polynomial time algorithm for linear-assume constraints systems as being due to the NP complexity of the general min-max-linear problem. We have implemented several existing timing verification algorithms.

A new general algorithm based on the linearization of min-max inequalities was proposed as well as an optimization technique that significantly improves its efficiency. This method has been used in the timing analysis of all four constraint type systems. Thus, the method can be applied to the exploration of a wider class of applications than previously presented ones.

The timing expression based on temporal constraint analysis is completely independent from languages used for system design, providing, in this manner, a possibility of reuse.

In this research, we presented a framework supporting the Transactional Level Modeling paradigm as one of the promising solutions to cope with the growing system complexity and the time to market pressure. The proposed framework provides a general approach in system modeling that separates various application modeling aspects from system specification: computational models, timing, specification/description languages, different functional and non functional parameters. To achieve this separation, we have used software engineering technologies and paradigms. Using software engineering technologies in TLM expression eases the creation of high level transaction models and provides a method to productively explore some architectural concerns at a high abstraction level. Among the software engineering technologies, we have chosen the .NET Framework, XML/XSLT and XSD technologies.

The .NET Framework has been chosen because it provides multiple predefined modeling constructs that allow easy creation of high level transaction models. Some of these elements are: multithreaded programming to model parallel activities, an events mechanism and interfaces and different kinds of abstract data types in order to model transactions The XML technology permitted to powerfully express a system specification with unifying functional and non-functional aspects such as cost, size, power dissipation, bandwidth, etc. in a single model framework. In this way, these constraints can be modeled in the earliest specification phase so that a designer can deal with them right from the beginning.

The XSLT technology used in the transformation of specification into a simulation model provides an efficient mechanism for design space exploration at different abstraction levels. The XSLT transformation contains simulator and specification language details, i.e. computation models related to the system under design. XML/XSLT and the .NET language interoperability support render TLM high level models independent from description and programming languages, in this way leading to high level functional specification reuse. This will allow reuse at different design stages, easier design exploration and exchange of specifications between different users having different HDLs and implementation environments.

Using the XSD technology, we have implemented the validation of the intermediate models' structures at each abstraction level in order to assure correctness of the modeling. As the correctness of the design tools is a necessity in modern conditions, the possibility to guide the designer in the designing process through multiple abstraction levels respecting several modeling rules guarantees error reduction in system design. We have implemented a graphic user interface to isolate the designer from the cumbersome and error prone syntax of the CP and CP+T abstraction levels.

We have integrated the min-max-linear-assume constraint timing specification methodology in the TLM design flow in order to represent and explore timing descriptions at different abstraction levels. We have extended the existing methods to perform the communication structure exploration and refinement at the PV+T level. This method can be used in different system design methodologies. The exploration of communicating components of the temporal model has been done using the local consistency property method which was generalized to handle min-max events. The proposed communication structure exploration methodology can be used in automatic protocol generation, in determining temporal specification inconsistencies and in adjusting some parameters in case of platform-based design methodologies. All these features lead to the reduction in the time needed for the exploration of the communication design space.

6.2 Future work

The presented analytical approach does not support verification of systems with cyclic non deterministic behavior. Thus, the elaboration of analytical methods to manage these kinds of behavior as well as parameterized temporal interval boundaries in the timing specification methodology is an important issue.

To estimate the average performance of the min-max constraint linearization algorithm, we have used random generated graphs. In the future, we need to collect numerical data of algorithm execution time for different real applications to confirm the obtained average case complexity.

In order to demonstrate the full strength of the presented analytical approach in the system verification step, it is necessary to create a library of temporal descriptions of hardware architecture components. In this case, many design decisions might be done efficiently without simulation. These decisions concern Hw/Sw partitioning, the mapping of system functionality to architecture components and the definition of the system communication infrastructure.

Further development of tools for the TLM framework is needed. The automated support for the PV and PV+T levels has to be implemented and a link with the SystemC TLM standard, as well as with other Hw/Sw design methodologies has to be established. An important contribution to easing the design will be an implementation of a semi-automated procedure of construction of the initial temporal specification from an untimed high level functional model. This model might be used for further exploration and refinement along with the simulation model providing, in this way, a powerful verification method combining analytical and simulation techniques.

Bibliography

1. S.Abdi, J. Peng, R. Doemer, D. Shin, A. Gerstlauer, A. Gluhak, L. Cai, Q. Xie, H. Yu, P. Zhang, D. Gajski, *System-On-Chip Environment (SCE): Tutorial*, tech. report CECS-TR-02-28, Center for Embedded Computer Systems, Univ. of California, Irvine, CA, 2002.
2. K. Keutzer, S. Malik, R. Newton, J. Rabaey, A. Sangiovanni-Vincentelli, "System Level Design: Orthogonalization of Concerns and Platform-Based Design", *IEEE Trans. On CAD*, vol. 19, no.12, Dec. 2000, pp.1523-1543.
3. VCC home page (www.cadence.com/products/vcc.html).
4. CoWare home page (www.coware.com).
5. W. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, L. Gauthier, M. Diaz-Nava, "Multiprocessor SoC Platforms: a Component-Based Design Approach", *IEEE Trans. On Design and Test*, vol. 19, no. 6, Nov-Dec. 2002, pp. 52-63.
6. L. Cai and D. Gajski, "Transaction Level Modeling: An Overview", *Proc. 1st Int'l Conf. Hrdware/Software codesign and syst. Synthesis* (CODES+ISSS'03), Newport Beach, California, 2003, pp.19-24.
7. A. Donlin, "Transaction Level Modeling: Flows and Use Models", *Proc. 2nd Int'l Conf. Hrdware/Software codesign and syst. Synthesis* (CODES+ISSS'04), Stockholm, Sweden, 2004, pp. 75-80.
8. G. Kahn, "The semantics of a simple language for parallel programming", *Proc. IFIP Congress 74*, North-Holland Publishing Co., 1974, pp. 471-475.
9. W3C Recommendation, XML 1.0 Specification, 3d ed., 2004, http://www.w3.org/XML/.
10. N. Pitts, *XML Black Book*, 2nd edition, Coriolis Technology Press, Scottsdale, Arizona, 2001.
11. A. Ceponkus, F. Hoodbhoy, *Applied XML: A Toolkit for Programmers*, Wiley Computer Publishing, Toronto, ON, 1999.
12. R. Powell, R. Weeks, *C# and the .NET Framework, The C++ Perspective*, Sams Publishing, Indianapolis, Indiana, USA, 2002.
13. C.T.I.Comité, *Codesign Conception conjointe logiciel – matériel*, Eyrolles, 1998 (in French).
14. P. Eles, K. Kuchcinski and Z. Peng, *System Synthesis with VHDL*, Kluwer Academic Publishers, 1998.
15. W3C Recommandation, XSL Transformations (XSLT) Version 1.0, 1999, http://www.w3.org/TR/xslt.
16. J. Lapalme, E. M. Aboulhamid, G. Nicolescu, L. Charest, F. Boyer, J-P David, G Bois, ".NET Framework - A Solution for the Next Generation Tools for System-Level Design", *Proc. of the Automation and Test in Europe*

Conference and Exhibition (DATE 2004), Paris, France, 2004, vol.1 pp. 732-733.

17. F.Balarin, E. Sentovich, M. Chiodo, P. Giusto, H. Hsieh, B. Tabbara, A. Jurecska, L. Lavagno, C. Passerone, K. Suzuki, and A. Sangiovanni-Vincentelli, *Hardware-Software Co-design of Embedded Systems – The POLIS approach*, Kluwer Academic Publishers, Norwell, MA, 1997.
18. J. Buck, Soonhoi Ha, Edward A. Lee, and David G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems", *Int. Journal of Computer Simulation,* special issue on "Simulation Software Development", vol. 4, Apr. 1994, pp. 155-182.
19. SystemC 2.1 library, http://www.systemc.org/.
20. W3C XML Schema Recommandation, http://www.w3.org/TR/xmlschema-0/, 2004.
21. D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, S. Zhao, *SpecC: Specification Language and Methodology*, Kluwer Academic Publishers, Jan. 2000.
22. T. Grotker, S. Liao, G. Martin, S. Swan, *System Design with SystemC*, Kluwer Academic Publishers, 2002.
23. S. Pasricha, "Transaction Level Modeling of SoC with SystemC 2.0", Synopsys User Group Conference (SNUG 2002), Bangalore, India, 2002.
24. SystemC 2.0 functional specification, http://www.systemc.org/.
25. R. Niemann, *Hardware/software Co-design for Data Flow Dominated Embedded systems*, Kluwer Academic Publishers, 1998.
26. T.Y. Yen, A. Ishii, A. Casavant, W. Wolf, "Efficient Algorithms for Interface Timing Verification", *Proc. of the Conf. on European design automation* (DATE 94), IEEE CS Press, Los Alamitos, CA, USA, 1994, pp.34-39.
27. T.Y Yen, A. Ishii, A. Casavant, W. Wolf, "Efficient Algorithms for Interface Timing Verification", *Formal Methods in System Design*, vol. 12, Kluwer Academic Publishers, 1998, pp. 241-265.
28. A. J. Gahlinger, *Coherence and Satisfiability of Waveform Timing Specifications*, doctoral dissertation, Dept. Computer Sciences, Univ. of Waterloo, Ontario, Canada, 1990.
29. K. L. McMillan, D. L. Dill, "Algorithms for Interface Timing Verification", *Proc. IEEE Int'l Conf. on Computer Design on VLSI in Computer&Processors*, IEEE CS, Washington, DC, USA, 1992, pp.48-51.
30. Y. Cheng, D.Z. Zheng, "Min-Max Inequalities and the Timing Verification Problem with Max and Linear Constraints", *Discrete Event Dynamic Systems: Theory and Applications*, vol. 15, Netherlands, 2005, pp. 119-143.
31. Y. Cheng, D.Z. Zheng, "An Algorithm for Timing Verification of Systems Constrained by Min-max Inequalities", *Discrete Event Dynamic Systems*, published online, Springer Science+Buisiness Media, LLC 2006.
32. A. El-Aboudi, E. M. Aboulhamid, "An algorithm for the verification of timing diagrams realizability", *Proc. IEEE Int'l Symposium on Circuits and Systems* (ISCAS'99), vol.1, May-June 1999, pp. 314-317.

33. A. El-Aboudi, E. M. Aboulhamid, E. Cerny, "Verification of Synchronous Realizability of Interfaces from Timing Diagram Specifications", *Proc. 10th Int'l Conf. on Microelectronics* (ICM'98), Dec. 1998, pp.103-106.
34. W. Wolf, J. Staunstrup, *Hardware/Software Co-design: Principles and Practice*, Kluwer Academic Publishers, Norwell, MA, 1997, pp.307-357.
35. *ISO Std. IS 8807, LOTOS a formal description technique based on the temporal ordering of observational behavior*, ISO, February 1989.
36. *CCITT Recommendation Z.100, Specification and Description Language* (SDL), ITU, Geneva, 1989.
37. SpecC Language Reference Manual, Version 2.0, December 12, 2002, www.specc.org.
38. E.A. Walkup, G. Borriello, "Interface timing verification with application to synthesis", *Proc. of the 31st annual conference on Design automation*, ACM Press, New York, NY, USA, 1994, pp.106-112.
39. R. K. Gupta, G. De Micheli, "Hardware-software Co-synthesis for Digital Systems", *IEEE Design and Test of Computers*, vol. 10, no. 3, Sept. 1993, pp.29-41.
40. C.A.R. Hoare, "Communicating Sequential Processes", *Communications of the ACM*, vol. 21, no. 8, August 1978, pp. 666-677.
41. A. Jerraya, K. O'Brien, "SOLAR: An Intermediate Format for System-Level Modeling", *Codesign: Computer Aided Software/Hardware Engineering",* IEEE Press, Piscataway, NJ, 1994, pp.145-175.
42. D. Pellerin, D. Taylor, *VHDL Made Easy!*, Pretice Hall, 1997.
43. S. Palnitkar, *Verilog HDL*, 2nd. Ed., Prentice Hall, 2003.
44. A.A. Jerraya, M. Romdhani, PH. Le Marrec, F. Hessel, P. Coste, C. Vaderrama, G. F. Marchioro, J.M. Daveau, N. – E. Zergainoh, "Multilanguage speification for system design and codesign", chapter in *System Level Synthesis*, NATO ASI 1998, Kluwer Academic Publishers, 1999.
45. D. Harel, "StateCharts: A visual formalism for complex systems", *Science of Computer Programming*, vol. 8, no.3, June 1987, pp.231-274.
46. C. A. R. Hoare, "Communicating Sequential Processes", http://www.usingcsp.com/ (current Jan. 2005).
47. L. A. Cortés, P. Eles, Z. Peng, *A Survey on Hardware/Software Codesign Representation Models*, SAVE project report S-58183, Dept. of Computer and Information Science, Linkoping Univ., Linkoping, Sweden, June 1999.
48. G. Berry, *The Foundations of Esterel, Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press, 1998.
49. E. A. Lee, D. G. Messerchmitt, "Synchronous Data Flow", *IEEE Transactions on Computers.*, vol. C-36, no. 1, Jan.1987, pp.24-35.
50. C. G. Cassandras, *Discrete Event Systems: Modeling and Performance Analysis*, Aksen Associates, Boston, MA, 1993.
51. M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni- Vicentelli, *A Formal Specification Model for Hardware/Software Codesign*, tech. report USB/ERL M93/48, Dept. EECS, Univ. of California, Berkeley, June 1993.

52. R. K. Gupta, C. N. Coelho Jr., and G. DeMicheli, "Synthesis and simulation of digital systems containing interacting hardware and software components", *Proc. of the 29th Design Automation Conf.* (DAC'92), IEEE CS Press, Anaheim, CA, 1992, pp. 225-230.
53. A. Balboni, W. Fornaciari, D. Sciuto, "Partitioning and exploration strategies in the tosca co-design flow", *Proc. of the 4th Int'l Workshop on Hardware/Software Co-Design,* IEEE CS Press, Washington, DC, USA, 1996, pp. 62-69.
54. D. D. Gajski, F. Vahid, S. Narayan, J. Gong, "System-Level Exploration with SpecSyn", *Proc. of the 35th Design Automation Conf.* (DAC 98), San Francisco, California, 1998, pp. 812-817.
55. The Chinook project, http://www.cs.washington.edu/research/projects/lis/chinook/www/.
56. E.A. de Kock, G. Essink, W.J.M. Smits, P. van der Wolf, J. – Y. Brunel, W.M. Kruijtzer, P. Lieverse, K. A. Vissers, "YAPI: Application Modeling for Signal Processing Systems", *Proc. 37th Design automation Conf.* (DAC'00), Los Angeles, CA, 2000, pp.402-405.
57. P. Lieverse, P. Van Der Wolf, E. Deprettere, K. Vissers, "A methodology for Architecture Exploration of Heterogeneous Signal Processing Systems", *Jurnal of VLSI Signal Processing*, vol. 29, no. 3, Nov. 2001, pp. 197-207.
58. A. Colgan, P. Hardee "Advancing Transaction Level Modeling: Linking the OSCI and OCP-IP Worlds at Transaction Level", http://www.opensystems-publishing.com/whitepapers.
59. W. Klingauf, M. Burton, R. Gьnzel, U. Golze, "Why We Need Standards for Transaction-Level Modeling", http://www.soccentral.com/.
60. G. De Micheli, *Synthesis and Optimization of Digital Circuits*, Mc Graw-Hill, 1994.
61. P. Vanbekbergen, G. Goossens, "Specification and Analysis of Timing Constraints in Signal Transition Graph", *Proc. of the 3rd European Conference on Design Automation* (DAC'92), 1992, pp.32-36.
62. T. M. Burks, K. A. Sakallah, " Min-Max Linear Programming and the Timing Analysis of Digital Circuits", *Proc. of the International Conference on Computer-aided design*, Santa Clara, California, United States, 1993, pp.152-155.
63. A. Tsikhanovich, E. M. Aboulhamid, G. Bois, "A Methodology for Hw/Sw Specification and Simulation at Multiple Levels of Abstraction", *Proc. of the 5th International Workshop on System-on Chip for Real-Time Applications (IWSOC)*, Banff, Canada, 2005, pp. 24-29.
64. A. Tsikhanovich, F. Rousseau, E. M. Aboulhamid, G. Bois "Transaction Level Modeling in Hardware/Software System Design using .Net Framework", *Proc. of the Canadian Conference on Electrical and Computer Engineering (CCECE)*, Ottawa, Canada, 2006, pp.140-143.
65. A. Tsikhanovich, G. Bois, chapter 8, "Timing specification in transaction level models", *System Level Design with .Net Technology*, E. M. Aboulhamid, F. Rousseau (ed.), CRC Press, USA, 2010, pp. 203-239.

66. A. Tsikhanovich, E. M. Aboulhamid, G. Bois "Timing Specification in Transaction Level Modeling of Hardware/Software Systems", *Proc. of the 50th Midwest Symposium on Circuits and Systems (MWSCAS*), Montréal, Canada, 2007, pp. 249-252.
67. A. Tsikhanovich, E. M. Aboulhamid, G. Bois "Communication Structure Refinement using Temporal Constraints Analysis", *Proc. of the 14th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Marrakech, Maroc, 2007, pp. 1284-1287.
68. A. Tsikhanovich, E. M. Aboulhamid, G. Bois "Temporal Constraints Analysis for Timing Verification of Systems", *Proc. of the 20th IEEE International Conference on Microelectronics (ICM*), Sharjah, UAE, 2008.
69. M. Sipser, *Introduction to the Theory of Computation*, PWS Publishing company, 1997.
70. E. Cerny, K. Khordoc, "Semantics and Verification of Action Diagrams with Linear Timing Constraints", *Transactions on Design Automation of Electronic Systems*, vol. 3, no. 1, January 1998, pp.21-50.
71. Embedded Systems Roadmap 2002, *Vision on technology for the future of PROGRESS*, ed. by L. D. J. Eggermont, 2002, Technology Foundation (STW), The Netherlands.
72. International Technology Roadmap for Semiconductors, Design Report, www.itrs.org, 2007.
73. R. Goering, "Ten 2008 trends in system and chip design", 2008, www.scdsource.com
74. W. Raslan, A. Sameh, "Accelerating High-Level SysML and SystemC SoC Designs", Design and Reuse Industry articles, www.design-reuse.com/articles/17562/high-level-sysml-systemc-soc-designs/html.
75. M. Abdurohman, K. Kuspriyanto, S. Sutikno, A. Sasongko, "Transaction Level Modeling for Early Verification on Embedded System Design", *Proc. of 8th IEEE Int. Conf. on Computer and Information Science (ICIS)*, Shanghai, Chine, 2009, pp. 277-282.
76. S. Mellor, M. Balcer, *Executable UML: A Foundation for Model-Driven Architecture*, Addison-Wesley, 2003.
77. A. S. Basu, M. Lajolo, M. Prevostini, chapter "A Methodology for Bridging the Gap between UML and Codesign", *UML for SoC Design,* Kluwer/Springer, 2005.
78. W. H. Tan, P.S. Thiagarajan, W. F. Wong, Y. Zhu, S. K. Pilakkat, "Synthesizable SystemC Code from UML Models", *Proc. of the Int. Workshop on UML for SoC Design (DAC)*, 2004, pp.23 - 27.
79. C. Erbas, A. D. Pimental, M. Thompson, S. Polstra, "A Framework for System-Level Modeling and Simulation of Embedded Systems Architectures", *EURASIP Journal on Embedded Systems*, vol. 2007, June 2007, pp. 12-24.
80. W. O. Cesario, G. Nicolescu, L. Gauthier, D. Lyonnard, A. A. Jerraya, "Colif: A Design Representation for Application-Specific Multiprocessor SOCs", *IEEE Design and Test of Computers*, vol.18, no. 5, September-October 2001, pp. 8-20.

81. OSCI TLM-2.0 User manual, June 2008, http://www.systemc.org/ .
82. The SPIRIT Consortium, www.spiritconsortium.org.
83. R. A. Wyke, A. Watt, *XML Schema Essentials*, Wiley computer Publishing, 2002.
84. .NET Framework, http://www.microsoft.com/net.
85. SPACE project, http://www.spacecodesign.grm.polymtl.ca.
86. Popularity of .NET is Grinding JAVA's Market Share, Finds Info-Tech Research Group", http://www.reuters.com, 2007.
87. MARTE project, http://www.omgmarte.org/
88. AADL project, http://www.aadl.info/
89. A. Koudri, D. Aulagnier, D. Vojtisek, P. Soulard, "Using MARTE in a Co-Design Methodology", *in Modeling and Analysis of Real-Time and Embedded Systems with the MARTE UML profile workshop co-located with DATE'08*, Munich, Germany, 2008, pp. 431-437.