A timed communication behaviour model for distributed systems

Yanwen Chen

2014 届研究生博士学位论文 学校代号: 10269

学号: 52101500002

華東師範大學

分布式系统的
时间化通信行为模型

院系:	软件学院
专业名称:	计算机应用技术
研究方向:	软件工程理论
博士研究生:	陈艳文
法国学校:	尼斯大学
项目:	中法联合培养，973物联网专项
指导老师:	陈仪香， Eric Madelaine

Université Nice Sophia Antipolis

二零一四年十月

DISSERTATION FOR DOCTOR DEGREE, 2014

School Code: 10269
Student Number: 52101500002

EAST CHINA NORMAL UNIVERSITY
&
UNIVERSITY DE NICE SOPHIA ANTIPOLIS

A Timed Communication Behaviour Model for Distributed Systems

Department:	Software Engineering Institute
Major:	Computer Application Technology
Subject:	Software Theories
Author:	Yanwen Chen
Project:	Joint Project, 973 CPS Project
Tutor:	Professor Yixiang Chen, Eric Madelaine

2014. 10

华东师范大学学位论文原创性声明

郑重声明： 本人呈交的学位论文《分布式系统的时间化通信行为模型》是在华东师范大学攻读硕士/博士（请勾选）学位期间，在导师的指导下进行的研究 工作及取得的研究成果。除文中已经注明引用的内容外，本论文不包含其他个人已经发表或撰写过的研究成果。对本文的研究 做出重要贡献的个人和集体，均已在文中做了明确说明并表示谢意。

作者签名:________ 日期: 年 月 日

华东师范大学学位论文著作权使用声明

《分布式系统的时间化通信行为模型》系本人在华东师范大学攻读学位期间在导师指导下完成的硕士/博士（请勾选）学位论文，本论文 的研究成果归华东师范大学所有。本人同意华东师范大学根据相关规定保留和使用此学位论文，并向主管部门 和相关机构如国家图书馆，中信所和"知网"送交学位论文的印刷版和电子版；允许学位论文进入华东师范大学图书馆及 数据库被查阅，借阅；同意学校将学位论文加入全国博士，硕士学位论文共建单位数据库进行检索，将学位论文的标题 和摘要汇编出版，采用影印，缩印或者其他方式合理复制学位论文。

本学位论文属于（请勾选）

（ ）1.经华东师范大学相关部门审查核定的"内部"或"保密"学位论文[1] 于 年 月 日解密， 解密后适用上述授权。

（ ）2.不保密，适用上述授权。

导师签名：________ 本人签名：________

年 月 日

[1]"保密"学位 论文应是已经华东师范大学学位评定委员会办公室或保密委员会审定过的学位论文（需附获批的《华东师范大学 研究生申请学位论文"保密"审批表》方为有效），未经上述部门审定的学位论文均为公开学位论文。此声明栏 不填写的，默认为公开学位论文，均适用上述授权。

博士学位论文答辩委员会成员名单

姓名	职称	单位	备注

摘要

随着网络技术的不断发展，物联网/物理信息融合系统成为目前研究和发展的热点。 一个典型的例子是智能交通系统（ITS)。 通信作为信息交换的媒介，已成为物联网研究的核心问题之一。 在智能交通系统中，车辆可以与服务中心沟通（V2I），告知其他车辆他们的存在以便于车辆的安全监控和安全驾驶; 另外车辆和车辆之间也可以通信（V2V），从而提高交通的安全性，避免恶性交通事故的发生。

该系统通信的实时性研究是非常重要的。分布式的系统通信更强调逻辑时钟。 为此本论文提出一种新型的适用于分布式系统通信的时间化网络通信模型（Timed-pNets）。 该模型包括了刻画时间化动作(Time Action)的逻辑时钟（Logical Clock）、时间化规范(Timed Specification)、 时间参数化标签迁移系统(timed Parameterized Label Transition System,timed-pLTS)等基本构件。 Timed-pNets是树型分层结构模型， 其叶子节点由Timed-pLTS表述，非叶子节点是子网的抽象，用于同步子网之间的通信。

本论文的主要贡献如下：

- 建立了一个分布式具有同步和异步通信的时间化模型Timed-pNets。在引入时间化动作形成动作逻辑时钟基础上， 建立时间化的pLTS系统（Timed-pLTS）。 Timed-pLTS中的标签为逻辑时钟， 用于触发系统从一个状态迁移到另一个状态。 论文把信道设计为Timed-pLTS，具有信息接受和发送两个动作逻辑时钟，用来描述异步通信。 基于Timed-pLTS，论文构造了Timed-pNets模型，它的同步向量用于描述不同节点之间的同步通信。 研究Timed-pNets的相容性（Compatibility）和延迟性（Delay）。

- 提出了时间规范（Timed Specication）的概念。时间规范定义为一组逻辑时钟和这些时钟上的关系， 包括时钟优先关系和时钟同步关系。提出了时钟划分和时钟合并的概念以简化时间规范， 研究了时钟的优先关系和同步关系作用在划分后时钟上的语义。

- 设计了一组算法用于把Timed-pLTS和Timed-pNets转化为时间规范，并提出了一套利用时间规范来建立层次化模型的理论和方法。 这样

人们可以灵活地设计通信系统：既可以先设计叶子节点， 然后组合成Timed-pNets节点这种层层向上的方法构建系统， 也可以先设计一个抽象的Timed-pNets系统， 然后用具体的Timed-pLTS实例化该系统中每个抽象孔的由上至下方式构建系统。

- 以智能交通系统中车辆相互通信为例子，实现如何建立Timed-pNets模型， 以及检查通信的安全性和时间性质。 使用TimeSquare工具完成这些性质的测试，结果表明论文建立的timed-pNets具有通用性和灵活性。

关键词： 分布式系统，物联网， 逻辑时钟， 时间规范， 形式化方法，智能交通系统（ITS）， 同步通信，异步通信

Abstract

With the development of the Internet, CPSs (Cyber Physical Systems) become a hot topic. A typical example are ITSs (Intelligent Transportation Systems), where communication is a critical part. In this kind of systems, vehicles can communicate with the infrastructure (V2I) to inform their existence for safety checking; and vehicles can also communicate between each other to improve the efficiency of traffic and avoid accidents.

The real-time communication in the system is a critical aspect. This thesis presents a novel timed model called timed-pNets for modeling and verifying the timed communication behaviours for distributed systems. Since the nodes in distributed systems have no common physical clock, this brings the challenge of correctly specifying the system time constraints. Timed-pNets build the time model on top of logical clocks such that the time of this model does not rely on a common physical clock.

The main contribution of the thesis are as follows:

- A formalism named Timed-pNets that is based on tree style hierarchical structures. The leaves of the structures are represented by timed Parametrized Label Transition Systems (timed-pLTSs). Non-leaf nodes (called timed-pNets nodes) are synchronisation devices that synchronize the behaviours of subnets (these subnets can be leaves or non-leaf nodes). Moreover, we discuss the compatibility and delay properties of the model.

- Timed specifications, which are at the core of this model and are designed to specify the system behaviours including synchronous and asynchronous communications. They consist of sets of logical clocks and some relations on these clocks. Moreover, we proposed the concept of clock partition and clock union to simplify the timed specifications, and investigate the clock relations on clock partitions.

- Algorithms design for the translation of timed-pLTS and timed-pNets

to timed specifications. Thanks to the timed specification, timed-pNets are able to model systems in a flexible way: from bottom to up, starting with detailed timed-pLTSs and assembling them in a compatible way; or from top to down, constructing timed specifications for abstract timed-pNets, using their holes timed specifications as hypotheses in an assume-guarantee style, and providing later some specific (compatible) implementations for these holes in various contexts.

- A discussion on time bound analysis, safety and latency properties based on the analysis of the relations conflicts between system logical clocks. We take a simple case of car insertion from the area of Intelligent Transportation Systems (ITS) as an example to demonstrate the use of the timed-pNets model. In the end, the TimeSquare tool is used to perform a logical simulation and check the validity of our model.

Key words: Distributed Systems, CPS (Cyber Physical Systems), Logical Clock, Timed Specification, Formal Methods, ITS, Synchronous Communication, Asynchronous Communication

Résumé

Cette thèse présente un nouveau modèle temporisé appelé timed-pNets pour la modélisation et la vérification des comportements des systèmes distribués hétérogènes. Un défi essentiel de ces systèmes est de spécifier correctement les contraintes de temps du système, dans la mesure où les nœuds dans les systèmes distribués n'ont pas l'horloge physique commune. Timed-pNets utilise un modèle de temps basé sur des horloges logiques, de manière à ce que les mesures de temps dans ce modèle ne reposent pas sur une horloge physique commune. Les timed-pNets ont une structure hiérarchique en arbre: les feuilles de cet arbre sont des Systèmes de Transition Étiquetés paramétrés temporisés (timed-LTSs), et les autres nœuds (appelés eux-aussi, par abus, Timed-pNets) sont des dispositifs de synchronisation qui permettent de composer les comportements de leurs sous-réseaux (eux-mêmes des timed-pNets). A chaque nœud d'un timed-pNet peut-être associée une Spécification temporisée, qui consiste en un ensemble d'horloges logiques et de relations sur ces horloges.

Les spécifications temporisées, en tant que le noyau de ce modèle, sont utilisées pour spécifier les comportements du système, y compris les communications synchrones et asynchrones. Grâce à la spécification temporisée, les timed-pNets peuvent modéliser des systèmes de manière flexible: soit de bas en haut, en commençant par des timed-pLTSs détaillés et en les composant de manière compatible; ou de haut en bas, construisant les spécifications temporisées pour des timed-pNets abstraits, en utilisant les spécifications temporisées de leurs arguments (trous) comme des hypothèses du style assume-garantee, et en fournissant plus tard des implémentations spécifiques (compatibles) pour ces trous dans divers contextes. Notre méthodologie permet un cycle de conception, qui part d'une spécification temporisée abstraite, et passe par des étapes de décisions d'architecture et de conception dépendant de l'infrastructure visée, correspondant à un raffinement des horloges logiques, contraint par des décisions d'ordonnance et de placements.

La version finale (entièrement raffinée) sera soumise à des vérifications de propriétés et de contraintes temporelles. Les analyses des limites de temps (relatives aux différentes horloges ou à une horloge de référence), de la sûreté et de la latence sont discutées par l'étude des conflits de relations entre les horloges logiques du système. Nous utilisons un scénario d'insertion de voitures dans les systèmes de transport intelligents (ITS) comme un exemple pour illustrer l'utilisation de notre modèle timed-pNets. Finalement, l'outil TimeSquare est utilisé pour effectuer une simulation logique et vérifier la validité de notre modèle.

Acknowledgement

The work described in this thesis was not and could not have been performed in isolation. It involved the help and support of many, to whom I am largely indebted. There are many people I would like to thank who directly or indirectly helped me achieve the milestone of completing this PhD thesis.

Foremost, I thank my PhD advisers Eric Madelaine and Yixiang Chen, for giving all possible support so that I make a successful research work. Thanks you for your encouragement and guidance throughout my research, for giving me opportunity to come to INRIA, Sophia Antipolis and to perform this work together with an enthusiastic team of researchers. Thanks Eric, for all useful discussions and wonderful words of wisdom. He has taught me most of what I know about formal method and supported me during a long period of research. It has been a wonderful time working with you and I couldn't have asked for more from you as my PhD adviser. This thesis would not be possible without you two!

I would like to thank all teachers, researchers, professors in SCALE team, INRIA and Fost team, ECNU for valuable discussions during round table, EPW and also many other occasions. Interacting with you all has always been a great learning experience. Special thanks to Ludovic HENRIO, Fabrice Huet for valuable suggestions. I also would like extend special thanks to Françoise BAUDE for her kindly support during my difficult period. Many thanks to Robert de Simone, Frédéric MALLET and Julien Deantoni for you suggestions on logical time, CCSL and TimeSquare. And thanks all professors and colleges of FOST team in ECNU, especially thanks to Min ZHANG, Tianmin BU, Jie ZHOU, Yanfang MA, etc. for all your supports. I thank all members of administrative staff and research coordinators for helping in many practical things. Special thanks to Christel KOZINSKI, Changbo WANG, Linjuan YE, Linying WU, etc.for their kindly help. I would like to thank all former and present members of my research group SCALE for inspiring research discussions and presentations. Thanks Sophie Song, for always giving an extra effort in making sure I continue my work without losing focus and passion during critical moments. I also would like to give my thanks to my office mate Alexandra Bardiau, even though you went to another world, your kindness will be always in my heart.

Finally, I would like to thank my family for always being there as an anchor of support and strength through all the trials and tribulations. I would like to thank my parents for all of the support, love, and encouragement they have given me. Their direction and advice has been invaluable, and without them, I could not have achieved nearly so much. I also thank my parents-in-law and sister-in-law for all the support they gave me.

Finally, I would like to thank my husband, Quirino Zagarese, for everything he has given to me. There is no other person I am indebted to on so many levels. Academically, his excellence has profoundly influenced me. I have learned uncountable lessons from his rigorous and careful pursuit of "the truth". He has been my sounding board, my expert reader, and my example to follow. For all of these, I thank him. Personally, his love and support has kept me alive and happy for the past three years. I am very lucky to have found such a wonderful person.

My deepest love to you all!

Table of Contents

Chapter 1 Introduction

1.1 Motivation and Challenges

The world is moving rapidly towards ubiquitous connectivity of smart devices that are interconnected and collaborating, which provides people with a wide range of innovative applications and services. It will further change how and where people associate, gather and share information, and consume media, which may be unimaginable today. The new world creates an unprecedented opportunity to connect not just devices, but peoples, data and processes as well, making networked connections more relevant and valuable.

One typical example is next-generation intelligent transportation systems (ITSs), in which wireless communications are used to exchange information among smart vehicles. These vehicles can communicate with service centers, inform other vehicles of their existence, monitor safety and use the latest road and weather conditions. Communications are needed to support safe driving, curtail traffic congestion and decrease travel delays by improving the way of the overall transportation system and its infrastructure work. The future of automotive safety is not about more airbags or stronger steel. It is about building smarter automobile that can "talk" to each other, so a car knows that another car is about to run a red light and applies brakes to avoid a possible accident. The U.S. Department of Transportation and the National Highway Traffic Safety Administration [91] have approved vehicle-to-vehicle (V2V) communication systems that will pave the way for connected cars to increase safety and reduce accidents. V2V communications can provide the vehicle and driver with 360-degree situational awareness to address additional crash situations. This technology would improve safety by allowing vehicles to "talk" to each other and ultimately avoid many crashes altogether by exchanging basic safety data, such as speed and position, ten times per second. In addition to enhancing safety, these future applications and technologies could help drivers to save fuel and time. Besides, German automakers have launched a pilot program that combines V2V with vehicle-to-infrastructure technology, allowing cars to communicate with each other and with traffic lights.

Not only vehicles, every devices can also connect to each other and communicate to provide better services. These devices include everything from cell phones, coffee makers, washing machines, headphones, lamps, wearable devices and almost anything else you can think of. The connection and communication of these devices bring a huge potential value to our life. For example, when you are on your way to a meeting, your car could have access to your calendar and already know the best route to take. If the traffic is heavy your car might send a text to the other parties to notify them that you will be late. It is also possible that your alarm clock can wake up you at 6 am and then notify your coffee maker to start brewing coffee for you. Also it will happen that your office equipment knows when it is running low on supplies and automatically re-orders more. And the wearable device you used in the workplace could tell you when and where you were most active and productive and share that information with other devices that you used while working. All these applications can help us reduce waste and improve efficiency and energy use. They will help us understand and improve how we work and live.

To realize the systems we expect especially for the efficiency we mentioned, very often it is necessary to consider real-time aspects of communication behaviours: quantitative information about time elapsing has to be handled explicitly. This can be the case to describe a particular behaviour (for instance, a time-out) or to state a complex property (for example, "the alarm has to be activated within at most 10 time units after a problem has occurred").

The real-time aspects for centralized systems such as embedded systems have been discussed for more than a decade. Usually, the communications in centralized systems are simple (synchronized communications) and limited (fixed number of communications are generated in a closed embedded system). Even though some systems include complex communications (asynchronous communication), the response time of the communications can be measured by a global physical clock. Comparing to the centralized systems, the decentralized system in the next generation world will generate large quantities of communications. These communications are created by millions of diverse devices periodically sending observations about certain monitored phenomena or reporting the occurrence of certain abnormal events of interest [88]. Furthermore, distributed smart devices in our future system may have their own clocks and the time measurements of the behaviour of each device are based on the physical clock of the device. The fact that no common physical global clock exists causes the most typical problems of the next generation heterogeneous distributed systems. The time measurement of communication behaviour and deadlock detection are much more difficult to solve in a distributed environment than in a classical centralized environment.

Besides, depending on communications between these distributed smart devices, the distributed systems can be classified as either synchronous or asynchronous. Synchronous communication is direct communication where time is synchronized. This means that all parties involved in the communication are present at the same time and ready to accept input signals. Asynchronous communication is the exchange of messages with a certain time lag between sending and responding. This means that the data in asynchronous communication can be transmitted intermittently. Future systems need the collaboration of synchronous and asynchronous communication. Furthermore, future distributed sensors, actuators, and smart devices with both deterministic and stochastic data traffic require a new paradigm for timed communication behaviour model that goes far beyond traditional methods. The interconnection topology of smart devices is dynamic and the system infrastructure can also be dynamically reconfigured in order to contain system disruptions or optimize system performance. There is a need of novel distributed communication models for dynamic topology control.

When talking about asynchronous communications models of distributed systems, most published research is based on the time-free model [13] [37],

[47]. In these models, the specifications describe what outputs and state transitions should occur in response to inputs, without placing any bounds on the time it takes for these outputs and state transitions to occur. This kind of free-time models are of importance in practice, such as consensus, election, or membership. However, investigating time properties (e.g. if system behaviours can be successfully executed before a certain deadline [78]) in distributed systems become important aspects. So we need a timed asynchronous distributed system model (or, for short, a timed model) where all the behaviours are timed: their specification prescribes not only the outputs and state transitions that should occur in response to inputs, but also the time intervals within which a client can expect these outputs and transitions to occur.

As we know, formal methods provide powerful techniques for specifying and verifying complex distributed systems. Most formal methods strive for simplicity, to allow for efficient analysis. A formal model can be very abstract, capturing precisely those aspects that are to be analysed, or can be very detailed, trying to capture as many of the design aspects as possible. Formalisms to construct mathematical models of systems include process algebra, labelled transitions systems, finite state automata, petri nets, and markov chains. All have their particular views on a system and focus on particular aspects. Design a formal model for the distributed systems and assess the correctness of the design of the system especially taking the time constraints into account is a difficult problem, because distributed systems have complex communication mechanisms and lack of a common physical clock. The mix of synchronous and asynchronous communications, as well as the possible time bound requirement in the distributed systems may lead to incorrect behaviours. This requires us to check the correctness of the formal models in terms of property requirements. If the required properties are satisfied, the result should have a meaningful interpretation for the verification of the actual design. Formal models for modeling time constrained systems include timed automata [4], timed petri net [94], AUTOSAR [55], STeC [39], BIP [14], etc. Each of them has its own special advantages, but, as far as we know, all of them use physical global time variables for time constraints, which does not match our goal of avoiding using a global common time when buidling models.

1.2 Research Approach

Heterogeneous distributed systems, as targeted in this thesis, can be characterized by the fact that the processors are spatially separated and that a common time base does not exist. Distinct processors in such systems communicate with each other by exchanging messages with an unpredictable (but non-zero) transmission delay. Each action in those processors is either a local step of a process, a send action, or a receive action. Since the processors in the systems may neither have synchronized clocks nor common physical time base, the logical order of the actions may not agree with the clock times associated with them. For example, we expect a logical view of the system in which the send action for a given message happens before the receive action for that message. However, if the clocks at the sender and the receiver are sufficiently skewed, a clock-based trace of the events might report that the receive occurred before the send.

One solution of this problem is to run algorithms to keep clocks closely synchronized, within some tolerance. In the Internet world, this is typically done with the Network Time Protocol (NTP). NTP is one of the earliest Internet protocols used and is probably one of the most used protocols today. However, it is much complicated and may cause problems by drastically changing time [70].

A better and simpler approach is to maintain logical clocks at the processors. Time-constrained models for distributed systems should take advantage of the system logical nature. The fact that one action causally affects another makes it possible to determine the practical order among actions. We use the concept of logical time to capture the causal relations of actions, which do not rely on a real time/clock. By this way, we are able to assign time values to actions such that it is possible to infer potential causality between these actions or to exclude causal influence in the sense that a “later” action cannot affect an “earlier” action.

To reflect the fact that the actions in a processor can repetitively occur and their causality relations keep the same, we define a logical clock as a sequence of repetitive occurrences of an action. A logical clock does not “tick” like a real time clock that is equally spaced, but instead keeps track of the order of action occurrences. Furthermore, inspired by the CCSL model [7] (the detail technique background of CCSL is presented in section 1.4), we define clock relations to specify logical time constraints between clocks. In distributed systems, as communication between processors is either synchronous or asynchronous, we choose the basic CCSL clock relations like coincidence and precedence to specify synchronous and asynchronous communications. We propose a novel way of modeling distributed systems by building system logical clocks and clock relations (called timed specification). A timed specification is usually used to specify the behaviour of a processor. Since a clock relation of two clocks is applied on all corresponding action occurrences of them, we can ensure that these action occurrences are assigned consistent logical times according to the relations between clocks. Then we employ time specifications into pNets(parameterized networks of synchronized automata) [13] to build a hierarchical structure of timed specification framework. The timed specification in a higher level is an abstraction of it low level subsystems. In our design model, by analyzing the inherent conflicts that might exist in the timed specifications, we check the logical correctness of the systems.

1.3 Research Contributions

In this thesis, we attempt to build a formal timed model (called timed-pNets) by introducing a set of logical clocks and clock relations into an untimed model called pNets (parameterized networks of synchronized automata) [13]. In this novel model, timed specifications (a set of logical clocks and clock relations) are used to specify the system behaviours, and furthermore, be used to build a hierarchical structure by composing the timed specifications of subsystems. By taking advantage of the timed specifications, system time constraints and properties (e.g. safety, latency properties) can be specified and verified. The main contributions of the thesis are as follows.

We design a novel model that is capable to specify logical time constraints in terms of system behaviours without relying on physical clocks (ref. chapter 3). In this new model, logical clock relations in bottom-level (synchronous) components are derived from the corresponding label transition systems (called timed-pLTS). Usually logical clocks are a priori independent. They become dependent when the instants (or the timed-action occurrences) from different clocks are linked by relationships (e.g. coincidence or precedence). Instead of imposing local dependencies between the instants (or the timed-action occurrences), we impose dependencies directly between clocks. A clock relation specifies many (usually an infinity of) individual time instant relations. As a result of adding clock relations to multiple clocks, these clocks are no longer independent and the instants (or the timed-action occurrences) are partially ordered. This partial ordering of instants characterizes the time specifications (TSs) of an application.

Timed specifications (TSs) are logical characterizations, that can be either provided by the application designer, or computed from the model. The consequence is that the two procedures above can be used arbitrarily in a bottom-up fashion, starting with detailed timed-pLTSs and assembling them in a compatible way; or in a top-down fashion, constructing TSs for abstract timed-pNets, using their holes TSs as hypotheses in an assume-guarantee style, and providing later some specific (compatible) implementations for these holes in various contexts. As our model has a hierarchical structure, the timed specification of an upper layer must be compatible with the timed specifications of its subnets (or subsystems). In order to be able to build a compatible model, we discuss the compatibility of refined implementations and abstract specifications. Moreover, we propose a theorem to generate a compatible structure of timed-pNets (ref. chapter 4).

Since the model does not rely on common physical clocks, the delays in the timed specifications that come from different subnets are uncomparable, which brings the difficulty of building a higher layer structure especially when the delays are taken into account. To solve the issue, we introduce the concept of reference clocks and virtual timestamps into our model so that the delays can be calculated in terms of a reference clock that a user choose (ref. chapter 5). The introducing of a reference clock also helps us to specify delay bounds and latency properties that are important aspects for a timed model. Therefore, this model has the capability of checking not only system's correctness and safety properties, but also the timed properties (e.g. deadline, latency properties).

The fact of using timed specifications in the new model paves the way of utilizing the TimeSquare tool to check system time constraint conflicts. Thanks to the timed specifications, timed-pNets are able to represent the basic behaviours of heterogeneous distributed systems. However, when facing to complex behaviours (e.g. undetermined clock choices), the current timed specifications are not easy to specify them. To simplify the way of encoding the complex situations, we design the concept of clock partition and clock union (ref. chapter 6). The clock partition allows us to flexibly split the timed-action occurrences into groups so that the clock relations can be applied to the groups instead of to every single occurrence. We prove that the relations (precedence and coincidence relations) on partition clocks can be substituted by those relations on a set of filtered clocks, which illustrates the advantages of using partition clocks: simple and easy to understand. Another extension, clock union, provides us a way to compose logical clocks. Usually it is used to specify the branches of transition systems.

In the end, to gain insight into our model, we apply our model on the Intelligent Transportation Systems (ITSs). We choose the TimeSquare [41] tool to do simulation (ref. chapter 7). TimeSquare is a software environment for modelling and analyzing of timed systems. It displays possible time evolutions as waveforms generated in the standard VCD format (more information of TimeSquare are introduced in the next section). Errors can be reported if conflicts exist in timed specifications.

As a conclusion, we contribute to design a formal model that provides a simple and flexible way to model communication behaviours (synchronous and asynchronous) with time constraints without relying on physical clocks. This is the main difference with other current timed models. Moreover, our model is able to check the logical correctness and verify time properties of distributed systems.

1.4 Technical Background

In this section, we introduce the technique background of timed-pNets, including logical clocks, CCSL, TimeSquare and pNets.

1.4.1 Logical Clocks

The logical nature of time is of primary importance when designing or analyzing distributed systems. The concept of logical clocks was first introduced by Leslie Lamport in 1978 to represent the execution of distributed systems [58]. It has then been extended and used in distributed systems to check the communication and causality path correctness [45]. The logical clock timestamps each event with an integer value such that the resulting order of events is consistent with a happened-before relation. Logical time has also been intensively used in synchronous languages [23] [20] for its multiform nature. The multiform nature of logical time consists in the ability to use any repetitive event as a reference for the other ones. It is then possible to express temporal properties between various references. In the synchronous domain it has proved to be adaptable to any level of description, from very flexible causal time descriptions to very precise scheduling descriptions [30].

Based on Lamport's logical clock, two more advanced logical clock (vector clock and matrix clock) have been proposed to capture causality between events of a distributed computation. Vector clock is proposed in order to retain the complete partial order information in a logical clock system. It is represented by an n-dimensional vector. Such clocks have been introduced and used by several authors. Parker et al. used in 1983 a very rudimentary vector clocks system to detect inconsistencies of duplicated data due to partitioning [72]. Liskov and Ladin proposed a vector clock system to define highly available distributed services [63]. The theory associated to these vector clocks has been developed in 1988 independently by Fidge [46] [45], Mattern [67] and Schmuck [76]. Similar clocks systems have also been proposed and used by Strom and Yemini [81] to implement an optimistic recovery mechanism, and by Raynal to prevent drift between logical clocks [73]. Another advanced logical clock called matrix clock is represented by an $n \times n$ matrix. Such a clock system has been proposed in 1984 by Wuu and Bernstein [90] to discard obsolete information of a log system. A similar mechanism has also been used by Lynch and Satin in 1987 for a similar purpose [75].

The aim of the logical time is to be able to timestamp consistently events in order to ensure some properties such as liveness, consistency, fairness, etc. In order to coordinate distributed processes, Jefferson proposed virtual time (or logical time, model time) [54] in 1985 for the causally connected distributed time. The virtual time is implemented with an optimistic time warp mechanism that is able to process messages quickly with independent of the future messages. The aim of using such virtual time is to ensure that the simulation program has the liveness property. The logical time is nothing else than the logical counterpart of the physical time offered by the environment and used in real-time applications [22].

1.4.2 CCSL

Logical time has been proved very useful to model heterogeneous and concurrent systems at various abstraction levels. The Clock Constraint Specification Language (CCSL) [7] uses logical clocks as first-class citizens and supports a set of (logical) time patterns to specify the time behaviours of systems. It is initially specified in an annex of MARTE [92], providing an expressive set of constructs to specify causality (both synchronous and asynchronous) as well as chronological and timing properties of the system models.

CCSL is a declarative language that specifies constraints imposed on the logical clocks of a model. A CCSL clock is defined as a sequence of clock instants (event occurrences). If c is a CCSL clock, for any $k \in \mathbb{N}$, $c[k]$ denotes its k^{th} instant. Below, we describe only the constraints used in this thesis. A comprehensive description of CCSL constructs can be found in [7].

The basic clock relations can be classified in three main categories: 1) coincidence-based constraints, 2) precedence-based constraints, and 3) mixed constraints.

Synchronous constraints rely on the notion of coincidence of clock instants. For example, the clock constraint c_1 $isSubclockOf$ c_2, denoted by $c_1 \subset c_2$, specifies that each instant of c_1 must coincide with an instant of c_2. In logical words this says that c_1 ticks only if c_2 ticks. Another example is coincidence constraint (c_1 *coincides* c_2), denoted by $c_1 = c_2$. It is a special case of subclocking, when there is a bijection between the sets of instants of the two clocks. It states that c_1 ticks if and only if c_2 ticks. Other examples of synchronous constraints are *excludes* (denoted $\sharp$) or *discretizedBy* . The former prevents two clocks from ticking simultaneously. The latter discretizes a dense clock to derive discrete chronometric clocks, mostly from $IdealClk$, a perfect dense chronometric clock, predefined in MARTE Time Library [8], and assumed to follow "physical time" faithfully (without jitter).

Asynchronous constraints are based on instant precedence, which may appear in a strict ($\prec$) or a non-strict ($\preceq$) form. The clock constraint c_1 $isFasterThan$ c_2 (denoted $c_1 \preceq c_2$) specifies that clock c_1 is (non-strictly) faster than clock c_2, that is for all natural number k, the k^{th} instant of c_1 precedes or coincides the k^{th} instant of c_2 ($\forall k \in \mathbb{N}$, $a[k] \preceq b[k]$). The constraint $c_1 \boxed{\prec} c_2$ specifies that clock c_1 is strictly faster than clock c_2, that is for all natural number k, the k^{th} instant of c_1 precedes the k^{th} instant of c_2 ($\forall k \in \mathbb{N}$, $a[k] \boxed{\prec} b[k]$).

Mixed constraints combine coincidence and precedence. For example, The constraint $c_3 = c_1$ *delayedFor n on* c_2 enforces a delayed coincidence, i.e., imposes c_3 to tick synchronously with the n^{th} tick of c_2 following a tick of c_1. It is considered as a mixed constraint since c_1 and c_2 are not assumed to be synchronous.

Moreover, CCSL includes clock expressions that define a set of new clocks from existing ones. A CCSL specification consists of clock declarations and conjunctions of clock relations between clock expressions. All these clock relations and clock expressions constitute the kernel of CCSL.

1.4.3 TimeSquare Tool

TimeSquare [41] is a software environment dedicated to analyze timed systems specified with clock constraints using the CCSL language [7]. It is composed of a set of Eclipse plugins and has been integrated into the Open-EmbeDD platform. It developed with Ganymede Eclipse Modeling Tools: ANTLR for constraint parsing, and JavaBDD for the solver.

TimeSquare has four main functionalities: 1) interactive clock-related specifications, 2) clock constraint checking, 3) generation of a consistent temporal structure, using a Boolean solver, 4) displaying and exploring waveforms, written in the IEEE standard VCD format.

TimeSquare has been designed to be used with the UML tools applying the MARTE profile. In this profile, clocks and clock constraints are associated with various model elements. A wizard is included in TimeSquare. It facilitates clock definitions, clock constraint specifications, model element browsing, and parameter setting. The second functionality checks constraint sanity. The third functionality relies on a constraint solver that yields a satisfying execution trace or issues an error message in case of inconsistency. The traces are given as waveforms written in VCD format. VCD (Value

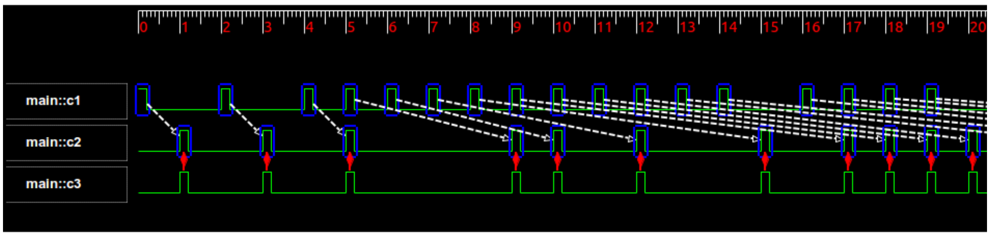

Fig. 1.1: VCD view of an example

Change Dump) is an IEEE standard textual format for dump files used by EDA (Electronic Design Automation) logic simulation tools. The solver intensively uses Binary Decision Diagrams (BDD). Waveforms can be displayed with any VCD viewer. TimeSquare has its own viewer enriched with interactive constraint highlighting and access facilities. For instance, the screen copy in Figure 1.1 shows precedence relations (white oblique dashed arrows) and coincidence relations (red vertical solid lines).

1.4.4 pNets Model

We build our behavioral semantic model by introducing logical clocks into pNets (parameterized networks of synchronized automata) [13]. pNets is an expressive and flexible semantic model for the modeling and verification of (untimed) distributed systems. pNets are networks of processes: they provide a hierarchical structure to organize processes. At the leaves of the structure, they have pLTS (parameterised labelled transition systems) describe in the definition II. Definition II describes the hierarchical composition. pNets are d To encode both families of processes and data value passing communications, parameters are used in pNets as communication arguments. The parameters is a set P of variables. The P is supposed to be defined globally, but it can also be defined locally in each pNet. The usage of parameters enables compact and generic description of parameterized and dynamic topologies. In the following part we recall definitions of pLTS and pNets. We start by giving the notion of parameterized actions that are basic elements for pLTSs. Parameterized Actions have a rich structure, because they take care of value passing in the communication actions, assignment of state variables and process parameters.

Definition I [Parameterized Actions] Let P be a set of parameter names, $\mathcal{L}_{\mathcal{A},\mathcal{P}}$ a term algebra built over P, including at least a distinguished sort A for actions, and a constant action τ. We call $v \in P$ a parameter, and $a \in \mathcal{L}_{\mathcal{A},\mathcal{P}}$ a parameterized action, $\mathcal{B}_{\mathcal{A},\mathcal{P}}$ is the set of boolean expressions (guards) over $\mathcal{L}_{\mathcal{A},\mathcal{P}}$.

The behaviour of a process is modelled as a parameterized labelled transition system (pLTS), in which the variables can be written and read by the actions performed in the transitions. A pLTS can have guards and assignment of variables on transitions. Variables can be manipulated, defined, or accessed inside states, actions, guards, and assignments. Parameters are used both for encoding data in value passing messages and for manipulating indexed families of processes.

Definition II [pLTS] A parameterized LTS is a tuple $< P, S, s_0, L, \to>$ where:

- P is a finite set of parameters, from which we construct the term algebra $\mathcal{L}_{\mathcal{A},\mathcal{P}}$,
- S is a set of states; each state $s \in S$ is associated to a finite indexed set of free variables $fv(s) = \widetilde{x}_{J_s} \subseteq P$,
- $s_0 \in S$ is the initial state,
- L is the set of labels, $\to$ the transition relation $\to \subset S \times L \times S$,
- Labels have the form $l =< \alpha, e_b, \widetilde{x}_{J_{s'}} := \widetilde{e}_{J_{s'}} >$ such that if $s \to s'$, then:
  - α is a parameterized action, expressing a combination of inputs $iv(\alpha) \subseteq P$ (defining new variables) and outputs $oe(\alpha)$ (using action expressions),
  - $e_b \in \mathcal{B}_{\mathcal{A},\mathcal{P}}$ is the optional guard,

- the variables $\widetilde{x}_{J_{s'}}$ are assigned during the transition by the optional expressions $\widetilde{e}_{J_{s'}}$ with the constraints: $fv(oe(\alpha)) \subset iv(\alpha) \cup \widetilde{x}_{J_s}$ and $fv(e_b) \cup fv(\widetilde{e}_{J_{s'}}) \subseteq iv(\alpha) \cup \widetilde{x}_{J_s} \cup \widetilde{x}_{J_{s'}}$.

pNets are constructors for hierarchical behavioural structures: a pNet is formed of other pNets, or pLTSs at the bottom of the hierarchy tree. A composite pNet consists of a set of subnets, each exposing a set of actions. The synchronisation between a global action of the pNet and the actions of the subnets is given by synchronisation vectors [10] with the form $< \alpha_i, \ldots, \alpha_j > \rightarrow \alpha_g$: a synchronisation vector synchronises one or several actions of subnets, and exposes a single resulting global action (α_g). The synchronous vectors are used to synchronise a (potentially infinite) number of processes. A pNet can either compose sub-pNets given explicitly, or be used as an operator accepting other pNets as parameters. Placeholders for the pNets that will be provided later are called holes. Actions synchronised in synchronisation vectors can involve both some sub-pNets that are given in the definition and some other that will be provided later. The holes in pNets can be indexed by a parameter, to represent (potentially unbounded) families of similar arguments. We represent the definition of pNets taken from [13] as follows:

Definition III [pNets] A pNet is a tuple $< P, pA_G, J, \widetilde{p}_J, \widetilde{O}_J, \overrightarrow{V} >$ where: P is a set of parameters, $pA_G \subset \mathcal{L}_{\mathcal{A},\mathcal{P}}$ is its set of (parameterized) external actions, J is a finite set of holes, each hole j being associated with (at most) a parameter $p_j \in P$ and with a sort $O_j \subset \mathcal{L}_{\mathcal{A},\mathcal{P}}$. $\overrightarrow{V} = \{\overrightarrow{v}\}$ is a set of synchronisation vectors of the form: $\overrightarrow{v} =< \alpha_g, \{\alpha_t\}_{i \in I, t \in B_i} >$ such that: $I \subseteq J \wedge B_i \subseteq Dom(p_i) \wedge \alpha_i \in O_i \wedge fv(\alpha_i) \subset P$.

Each hole in the pNet has a parameter p_j , expressing that this “parameterized hole” corresponds to as many actual processes as necessary in a given instantiation of its parameter. In other words, the parameterized holes express parameterized topologies of processes synchronised by a given Net. Each parameterized synchronisation vector in the pNet expresses a synchronisation between some instances ($\{t\}_{t \in B_i}$) of some of the pNet holes ($I \subseteq J$).

The hole parameters being part of the variables of the action algebras can be used in communications and synchronisations between the processes.

The pNets allow to model a large variety of synchronisation mechanisms and have been traditionally used for systems of either synchronously or asynchronously communicating objects, and of distributed components [13]. The flexibility of the synchronisation vectors mechanism naturally provides descriptions of heterogeneous systems, from point-to-point or multipoint synchronisations, to sophisticated asynchronous queuing policies. It is a low-level semantic model, supporting a large variety of parallel operators and communication mechanisms that are flexible enough to address a large set of distributed programming concepts. pNets can be used typically as the target of behaviour semantics for same high level language. For example, [13] gives the semantics of the component based framework in terms of pNets. Parametrization and hierarchy also makes pNet models compact, and close to the program structure, and as a consequence easy to generate in a compositional way [6]. Its parameterized and hierarchical features can build a tree like structure in which each node is pNets and leaves are pLTSs. Each pNets node, which can also be presented as a pLTS, is an upper layer abstract node composed by its subsystems in terms of the communication behaviours among them. The parameterized models have successfully been used for modelling ProActive [35] that is a pure Java implementation of distributed active objects with asynchronous remote method calls and replies. It has been proven that the pNets are suitable as a specification language for the distributed systems, and for the models resulting from static analysis of source code. Moreover, the model enables us to have a finite representation of infinite systems. It naturally encodes the semantics of languages for distributed applications.

All these incomparable advantages attracted us to choose it for modelling distributed systems. However, pNets have no mechanism to describe system time constraints.

1.5 Use Case

In this section, we represent two use cases taken from ITS. One is vehicle-to-infrastructure communication application which intends to avoid vehicles accidents and to increase environmental benefits by wireless exchange of critical data between vehicles and highway infrastructures. Another is a vehicle-to-vehicle communication application that offers the opportunity for significant safety improvements by dynamic wireless exchange of data between nearby vehicles. The two cases will be used from the chapter 3 to the chapter 7 to explain our approach of building semantic behaviour models.

1.5.1 Vehicle-to-Infrustructure Communications

We present a use case called speed controlling system taken from [91]. The speeds of cars are monitored by an infrastructure that collects information from cars and sends brake signals back to cars if they exceed the speed limit. To realize it, the cars in a highway keep on sending signal “I'm here” with their location and speed data. The infrastructure along the highway collects the heartbeat signals and checks the speeds of those cars. If the speed of a car exceeds the speed limit of the highway, the infrastructure will send a “brake” signal to let the car to reduce its speed. The communication protocol is described as follows:

- Cars send heartbeat signals ”I'm here” with parameters ”(location, speed)”;
- A infrastructure collects heartbeat signals from cars;
- The infrastructure sends ”brake” signal to the cars that exceed the speed limit;
- The cars reduce their speed when they get the ”brake” signals.

We require that the cars can receive brake signals and response to the infrastructure before sending the next heartbeat signal. This use case will

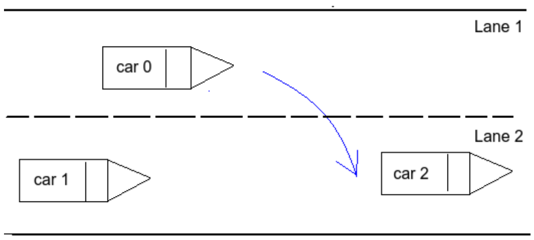

Fig. 1.2: Car Insertion

be simulated in chapter 3.

1.5.2 Vehicle-to-Vehicle Communications

We choose another small scenario on vehicle-to-vehicle communications. It is about an autonomous lane change involving 3 smart cars. These cars are equipped with sensors to detect the physical environments and parameters (e.g. such as the speeds and distances of the cars). And they communicate among each other to coordinate their movements and avoid collisions. Assume three vehicles ($car0$, $car1$ and $car2$) are running on a road as Fig. 7.1.

The scenario of inserting $car0$ between $car1$ and $car2$ may follow the following steps: 0) $car0$ gets a change-lane request (e.g. from a human user); 1) $car0$ sends "notify" requests to $car1$ and $car2$ to get an agreement; 2) $car1$ (resp. $car2$) acknowledges $car0$ "yes" or "no"; 3) $car0$ collects results from $car1$ and $car2$; 4) If both $car1$ and $car2$ answer "yes", $car0$ signals the consensus to $car1$ and $car2$ and then go to step 5, otherwise $car0$ aborts the procedure; 5) $car1$ slows down and/or $car2$ speeds up to leave more space between them for $car0$; 6) $car0$ changes its direction and moves to lane2; 7) $car0$ notifies the end of the procedure with a "finish" signal.

We require that the system has no deadlock or clock relation conflicts. Furthermore, assuming that the network communication delay is less than 10 time units, we require that the latency from sending notifications to finishing collecting all acknowledgements is no more than 30 time units. And the latency of whole procedure from car0 getting change-lane requests to sending "finish" signals is no more than 55 time units.

This use case will be used to explain the timed-pNets model in the chapters 4, 5 and 6. Moreover, the full simulation is represented in chapter 7.

1.6 The Outline of The Thesis

The rest of the thesis has been organized as follows.

- chapter 2 discusses related works and carefully investigates some time models like timed-automata, timed petri Nets, MARTE and AADL that are famous on modelling real-time systems.

- chapter 3 generalizes a novel semantic model by introducing logical clocks and clock relations into pNet so that it has the capability of modeling time constrained distributed systems.

- chapter 4 describes a communication behavioural semantic model called timed-pNets. It is an extension of chapter 3. Timed-pNets build a hierarchical structure of timed specifications by which the system timed constraints can be specified in a more compatible and easier way. Moreover we discuss the compatibility and refinement of timed specifications, as well as the property checking. We demonstrate that timed-pNets are able to model the timed constrained communication behavior for heterogeneous distributed systems that include synchronous and asynchronous communications.

- chapter 5 discusses how to compute the delays and delay bounds in timed-pNets. Moreover, we define the concept of time conflicts and propose a way to detect them.

- chapter 6 discusses advanced extensions of timed-pNets, including clock partition and clock union for simplify the timed specifications.

- chapter 7 represents the full details of car insertion use case to demonstrate how we build and refine a timed-pNets model and check its safety and timed properties.
- chapter 8 concludes our work and represents future works.

Chapter 2 Related Work

If you want to understand today, you have to search yesterday.

Pearl Buck, American female writer

Our idea of avoiding using any common physical clock when modelling distributed systems leads us to investigate logical time and some existing time models. This chapter starts from introducing discrete-event model to understand how system behaviours are specified by taking advantage of events. Then we investigate globally asynchronous locally synchronous (GALS) model including HipHop to understand how synchronous and asynchronous communications are handled. BIP, as a framework for the incremental composition of heterogeneous components, is also investigated. Moreover, we carefully investigate some time models like timed-automata, timed petri net, MARTE and AADL that are famous on modelling real-time systems. Other time related systems like STeC are introduced to see how they specify time and location constraints for actions.

Model-integrated development [53] [56] commonly uses actor-oriented software component models [60] [61]. In such models, software components (called actors) execute concurrently and communicate by sending messages via interconnected ports [71] [29]. Examples that support such designs include Simulink, LabVIEW, SystemC, SysML, UML and pNets.

A well-defined actor-oriented model of computations (MoCs) should always has well-defined semantics. One of the key challenges is to integrate actor-oriented models with practical and realistic notions of time. For example, when modeling distributed behaviours, it is essential to provide multiform models of time. The frameworks that include a semantic notion of time, such as Simulink, assume that time is homogeneous in the sense that it advances uniformly across the entire system. In practical distributed systems, even those as small as systems-on-chip, however, no such homogeneous notion of time is measurable or observable. In a distributed system, even though it uses network time synchronization protocols (such as IEEE 1588 [62]), local notions of time will differ. So when introducing time into the pNets model, we should carefully handle the notions of time. Failing to model such differences of time could cause errors in the design.

Based on the idea of logical time, the related models such as discrete-event models, asynchronous language models and so on have been proposed. Besides, some formal models or frameworks with time constraints have also been proposed to describe timed systems. Here we list and describe these previous works that relate to our work.

2.1 Discrete-event Models

Discrete-event (DE) [36] [59] [93] models are formal system specifications that have analyzable deterministic behaviours. DE models are concurrent compositions of components that interact via events. An event is a time-stamped value, where time is "logical time" or "modeling time" [58]. Correct execution of such models requires respecting the order of time stamps.

Using a global, consistent notion of time, DE components communicate via time-stamped events. DE models have primarily been used in performance modeling and simulation, where time stamps are a modeling property bearing no relationship to real time during execution of the model.

One interesting project that directly confronts the multiform nature of time in distributed systems is the PTIDES (Programming Temporally Integrated Distributed Embedded Systems) project [42] [43]. PTIDES serves as a coordination language for model-based design of distributed real-time systems. PTIDES provides a framework for exploring a family of execution strategies so that it can directly confront the multiform nature of time in distributed systems. DE is usually a simulation technology (e.g. in hardware description languages such as Verilog and VHDL and network modeling languages such as OPNET Modeler1 and Ns-22). When DE models are executed on distributed platforms, the objective is usually to accelerate simulation, not to implement distributed real-time systems [36] [48] [93]. However, PTIDES does not use DE as a simulation technology, but rather an application specification language, which serves as a semantic basis for obtaining determinism in distributed real-time systems. Applications of PTIDES are given as distributed DE models, where for certain events, their modeling time is mapped to physical time. Simulations of it can simultaneously have many time lines, with events that are logically or physically placed on these time lines. PTIDES has DE semantics, but with carefully chosen relations between model time and real time. It provides semantics for the interactions between events to model the communications of distributed systems. Key to making this model effective is to ensure that the constraints that guarantee determinacy in the semantics are preserved at runtime. To accomplish this, a distributed execution strategy is given, which obeys DE semantics without the penalty of totally ordered executions based on time stamps. The execution strategies are divided into two layers: global coordination, and local resource scheduling. When receiving an event from the network, the global coordination layer determines whether the event can be processed immediately or it has to wait for other potentially proceeding events. Once it is sure that the current event can be processed according to DE semantics, it delivers the event over to local resource scheduler, which may use existing real-time scheduling algorithms, such as earliest deadline first (EDF) to prioritize the processing of all pending events. Based on causality analysis of DE models, relevant dependency and relevant orders is defined to enable out-of-order execution without compromising determinism and without requiring backtracking. Since the global, consistent notion of time may lead to a total ordering of execution in a distributed system, which is an unnecessary waste of resources, PTIDES takes this event-driven execution strategy. Unlike many hard real-time distributed systems that depend on domain specific network architectures, PTIDES only requires a reliable packets delivery with a known bounded delay.

The DE models encourage us to take advantage of logical nature of systems and to introduce logical time into models. However, we need more mature communication mechanisms like synchronous and asynchronous communications that are not yet supported in PTIDES. This drives us to investigate heterogeneous communication models.

2.2 Synchronous and Asynchronous Communication Models

Synchronous languages [20] have been effectively applied to design reactive systems. These languages (which include Esterel, SCADE, Lustre, Signal, etc.) provide deterministic concurrent semantics. Synchronous programs can be efficiently and safely implemented. The correctness is ensured by usual verification methods. However, in the domain of distributed systems, asynchronous languages (e.g., SDL [80]) are naturally be used. It brings the needs for programming the system “globally asynchronous locally synchronous (GALS)” [38]. GALS is a model of computation that allows to design computer systems consisting of several synchronous components, among which the communications are asynchronous, e.g., FIFOs. It can be used both in software and hardware. In software, these synchronous components usually are specified as finite state machines (FSMs) and the asynchronous communication between them is modeled with a buffer [40]. The idea of the GALS approach provides a methodology for combining concurrent embedded systems within loosely coupled systems. Several formalisms have been proposed which combine synchronous and asynchronous primitives (e.g., [5]). And the concept of GALS has been used in several models and tools [19] [21] [12].

Another model that can deal with asynchronous communication events is HipHop language [24] [26]. It came out for helping programming rich applications driven by computers, smart phones or tablets. Since they interact with various external services and devices, safe programming of this network of devices requires tight cooperation between many sequential and parallel programming models, as well as orchestration techniques that merge classical computing, client-server concurrency, web-based interfaces, and event-based programming.

HipHop is an Esterel-based [25] orchestration language embedded into the Hop language [77] and system. Hop is a scheme-based multi-tier language to develop complex web applications with a single source code for the server and client, making code migration and client/server communication fully transparent. HipHop is used to orchestrate asynchronous activities launched by Hop, by providing a synchronous view and control of them. HipHop is based on synchronous concurrency and preemption primitives, which are known to be key components for the modular design of complex temporal behaviors. It adds the possibility of orchestrating complex concurrent behaviors into Hop. Compared to Esterel, it is a much more dynamic language, whose programs are Hop values that can be constructed on the fly and run by an interpreter that implements the constructive causality of Esterel. HipHop can be used both on the server and client side for maximal flexibility.

Our model partly takes the idea of GALS to specify both synchronous and asynchronous communication. The main difference is that we specify the synchronous components as timed specifications (a set of clocks and clock relations) instead of FSMs so that we are able to take advantage of the TimeSquare tool to check system properties. Moreover, in our model, the synchronous communications are specified by the coincidence relations between clocks, while the asynchronous communications are modeled by channels in which precedence relations are applied on two clocks. Compared to HipHop, we both handle asynchronous events and multi-tier structure, but the different aims drive us to different directions. They focus on orchestrating complex concurrent behaviours for web applications, while we need not only the correctness of system behaviours, but also take account the time constraints of these behaviours. Thus, we go further to investigate some time-constrained models. Timed-automata is the first one we would like to investigate that is famous for specifying and verifying the time constraints of real-time systems.

2.3 BIP Framework

BIP (Behaviour Interaction Priority) [14] is a framework for the incremental composition of heterogeneous components. It allows building complex systems by the coordinating the behaviour of a set of atomic components. The BIP framework provides constructs for dealing with parametric and hierarchical descriptions as well as for expressing timing constraints associated with behaviour.

BIP supports a component-based modeling methodology based on the theory that components are obtained as the superposition of three independent layers. The lowest layer describes the behaviour of a component (basic components) as a set of transitions (i.e. a finite state automaton extended with data); the intermediate layer includes a set of connectors describing the interactions between transitions of the layer underneath; the upper layer consists of a set of priority rules describing scheduling policies for interactions.

Such a layering offers a clear separation between component behaviour and structure of a system (interactions and priorities). The states inside a component denote control locations where the component waits for interactions. A transition is an execution step from one control location to another. Each transition has an associated condition that enables this transition and an action that is executed at this transition.

In BIP, all actions executed by transitions are written in C/C++. The BIP language provides additional structural syntactic constructs for defining component behaviour, interactions and priorities. BIP supports the construction of sub-systems and allows developers compose systems by layered application of interactions and priorities [51]. A hierarchical structure can be built by composing components from atomic one that consists of a set of ports (for the synchronization with other components), a set of transitions and a set of local variables. There is a clear separation between behaviour (the finite-state machines) and composition glue (stateless interactions and priorities).

“BIP encompasses heterogeneity. It provides a powerful mechanism for structuring interactions involving strong synchronization (rendezvous) or weak synchronization (broadcast). Synchronous execution is characterized as a combination of properties of the three layers.” — taken from the reference [15].

The BIP framework consists of a language and a toolset including a frontend for editing and parsing BIP programs and a dedicated platform for model validation. The platform consists of an Engine and software infrastructure for executing models. It allows state space exploration and provides access to model checking tools of the IF toolset [32] such as Aldebaran [31] and the D Finder tool [18]. This permits to validate BIP models and ensure that they meet properties such as deadlock-freedom, state invariants [18] and schedulability.

Real-time (RT) BIP [1] is an extension of the BIP component-based design language to continuous time model closely related to timed automata

[4]. In addition to offering syntax and semantics for the time-aware modeling of concurrent systems, the real-time BIP also envisions a general model-based implementation method for safety-critical multicore systems. This method is based on the use of two models: (1) an abstract model representing the behaviour of real-time software with user-defined timing constraints; (2) a physical model representing the behaviour of the real-time software running on a given platform.

The BIP and its real-time extension RT-BIP are currently supported by an extensible toolset including a concrete modeling language together with associated analysis and implementation. The toolset provides functional validation, model transformation and code generation features.

However, modeling timed components in BIP involves references to a specific "tick" port expressing the passage of (discrete) time, and such "tick" events must be synchronized between various components of a system before computing worst case execution time (WCET) or task period properties. With contrast to this approach, we do not want to use one clock to synchronize the components, but rather embedded multiple logical clocks into a model to specify the synchronous and asynchronous communications by building logical clock relations.

2.4 Timed-automata

Timed-automata [4] is a widely studied formalism for timed systems and is famous for modelling the behaviour of real-time systems. It provides a simple and powerful way to annotate state transition graphs with time constraints using finite real-value clocks. A timed automaton is a finite automaton extended with a finite set of real-valued clocks. It can be seen as classical finite state automata with clock variables and logical formulas on the clock (temporal constraints) [17]. A timed automaton accepts timed words — infinite sequences in which a real-valued time of occurrence is associated with each symbol. Transition tables in automata are extended to timed transition tables so that they can read timed words. A finite set of (real-valued) clocks are involved in each transition table. The clocks in timed-automata are initialized to zero when a system is started and then increase at the uniform rate counting time with respect to a fixed global time frame. Each clock can be separately reset to zero [3]. The clocks keep track of the time elapsed since the last reset. When an automaton makes a state-transition, the choice of the next state depends upon the input symbol read.

Each transition is associate with a clock constraint, and require that the transition may be taken only if the current values of the clocks satisfy this constraint. The constraints on the clock variables are used to restrict the behaviour of the automaton. There are two types of clock constraints: constraints associated with transitions and constraints associated with locations. The constraints associated with transitions make use of guards. A guard is a Boolean combination of integer bounds on clocks and clock differences. A transition can be taken when the clock values satisfy the guard labeled on it. The constraints associated with locations are called invariants and they specify the amount of time that may be spent in a location. The invariant "true" for a location means there are no constraints for the time spent in the location. Semantics for a time automaton are defined as "a transition system where a state or configuration consists of the current location and the current values of clocks" [17].

In Automata, delays can be established for transitions. The delays are counted from a physical clock which should be reset when it starts counting. Since the clock in timed-automata is dense time, the discretization of delay in timed-automata is investigated so that qualitative behaviour of circuit can be preserved [11].

Timed automata can be used to model and analyse the timing behaviour of computer systems, e.g., real-time systems or networks. Methods for checking both safety and liveness properties have been developed and intensively studied over the last 20 years. It has been shown that the state reachability problem for timed automata is decidable, which makes this an interesting sub-class of hybrid automata. Extensions have been extensively studied, among them stopwatches, real-time tasks, cost functions, and timed games. Closure properties, decision problems as well as automatic verification of real-time requirements were considered in timed-automata. There exists a variety of tools to input and analyse timed automata and extensions, including the model checkers UPPAAL [16], Kronos, and the schedulability analyser TIMES. These tools are becoming more and more mature, but are still all academic research tools.

Timed-automata can be a good reference for building and verifying timed models. The strong theory basis and verification tools support many industry implementations. Compared to timed-automata, we do not directly use real-valued clocks whose values increase all with the same speed. Instead, we build our model with two steps: We first define a finite set of logical clocks whose ticks happen in terms of the occurrences of actions. Thus, the distance between two adjacent ticks may not be the same. And the speed of two logical clocks may not be comparable. We specify system behaviours and constraints by timed specifications, which help us to specify and check system safety properties like deadlock. Then, we assign timestamps for clock ticks in terms of a reference clock. The reference clock for our model is still a logical clock that provides a time base for other logical clocks. By taking advantage of the reference clock, we are able to check time properties such as latency in terms of the timestamps of the reference clock. Therefore, our model is flexible to fit for distributed systems that have no common physical time base. Meanwhile, time properties can be checked under the assumption of a reference clock.

2.5 Timed Petri Nets

Another famous semantic model for real-time systems is timed petri nets. Timed petri nets (TdPNs) [83] is one of several mathematical modeling languages for the description of distributed systems. It is widely used for the modeling and analysis of concurrent systems with time-dependent behavior like communication systems. It includes a set of directed bipartite graphs, in which the nodes represent transitions (i.e. events that may occur, signified by bars) and places (i.e. conditions, signified by circles). The directed arcs describe which places are pre- and/or post- conditions for which transitions (signified by arrows). Each arc associates with an interval (or bag of intervals). In TdPNs, each token has an age. This age is initially set to a value belonging to the interval of the arc which has produced it or set to zero if it belongs to the initial marking. Afterwards, ages of tokens evolve synchronously with time. A transition may be fired if tokens with age belonging to the intervals of its input arcs may be found in the current configuration.

In most timed petri nets models, transitions determine time delays [84] [86] [57]. In only a few models, time delays are determined by places and/or arcs [79]. Three types of delay are discussed in timed petri net, deterministic, nondeterministic, and stochastic delays. Many of the older timed petri net models, such as [87], [74], [79], [94], use deterministic delays i.e., the delay assigned by a transition, place, or arc is fixed. Deterministic delays allow for simple analysis methods but have limited applicability. In most cases, delays correspond to the duration of activities which are typically variable. Therefore, fixed delays are often less appropriate. There are two ways to describe the variability. One way is to assume constraints on delays (e.g., it takes less than 2 seconds to send a notification). Another way is to assume a probability distribution for each delay. Most models use time intervals to specify the duration of the delay. Such a model introduced by Merlin [69] [68] in the early seventies. Other models [84] [85] [27] that use interval timing have been proposed. Some timed petri nets models, such as [49] [2], proposed stochastic delays in the sense that each delay is described by a probability distribution.

Another way to classify the types of delay used in a timed petri net model is to distinguish between discrete and continuous delays. Most discrete models use the natural numbers as the time domain. Continuous models typically use the set of non negative real numbers as the time domain. Nearly all timed petri nets allow for continuous delays. There exist several analysis tools, such as TINA or Romeo. TINA (TIme Petri Net Analyzer) [28] is a toolbox for the edition and analysis of petri nets and timed petri nets. Moreover, an approach has been proposed in the work [50] to translate UML-MARTE Activity Diagrams to time petri net (TPN) with the aim of verifying efficiently time properties (synchronization, schedulability, boundedness, WCET, etc) in real time embedded system. This work focuses on how to define TPN based formal semantics for UML-MARTE Activity Diagrams to avoid the core problem of state space explosion in model checking. TPN is selected as verification model, because of the maturity of both its theory and the associated TINA toolset, as well as its powerful capacity to express temporal semantics.

Compared to timed petri nets, we use a different way to build a timed model. First, we build our model by means of label transition systems (LTSs) to model system behaviours. Our model graph comprises a set of states, with arcs between them labeled by the activities of a system. Second, by translating LTSs to timed specifications, we actually build a unified way to specify system behaviours and logical constraints, and pave the way to use the tool TimeSquare. For real-time constraints, we also define delays, but our delays are neither in states nor in labels. We encode delays in each action. These delays are non-deterministic delay in the sense that the delay bounds are specified as (logical) time intervals. We choose action-based LTSs to model our systems with two reasons. 1) Our goal is to check the correctness of system communication behavior, not to verify the correctness of programming computations. So we hide unnecessary detail information like state variables, but highlight the information that related to communication behaviours like actions. 2) Action-based LTSs help us to build a compact and hierarchical model since the states of them are abstract nodes.

2.6 AADL

Then we investigate two model-based engineering (MBE) tools for real-time systems: AADL and MARTE (extended from UML). They automate the analysis and facilitate the modeling of software architecture. UML was conceived as a way to model functional structures of software (data, interaction, and evolution); while AADL is a way to model and analyze runtime architecture.

The SAE architecture analysis & design language (AADL) [44] is a programming language not only to define the textual representation of software architecture but also (and more importantly) to formally define the syntax and static semantics. In addition to textual representation, AADL allows the software designer to depict the system graphically. It simplifies the way of designing and analyzing the software and hardware architecture of performance-critical real-time systems.

Descriptions in AADL comply with the syntax and semantics of the language and can be verified by the syntactic and semantic analyzer of the language to ensure that the description is analyzable and consistent. In other words, constructs in a model are checked by the compiler to verify that they are "legal". Verification of the descriptions checks that a program is properly structured, consistent, and semantically correct. AADL provides an extension construct called annex to add complementary description elements for different kinds of analysis. These annexes are embedded in the descriptions of its core language. AADL analysis tools, for example "open source AADL tool environment (OSATE) [52], implement annexes as parsers, resolvers, and semantic checkers. They execute the basic checking of the core language and provide full consistency verification.

AADL focuses on runtime architecture modeling and analysis. Runtime architecture is the software structure that defines the final execution sequence of instructions. This software structure called software component is defined by threads, processes, processors, and their interactions (data, event, and event data communication). Runtime architecture provides the software system with specific quality attributes such as timeliness, fault-tolerance, or security.

AADL language semantics, enforced by compilation techniques, provide a clear execution semantics that is defined as a hybrid automaton. A hybrid automaton is a mathematical model for describing how software and physical processes interact. The AADL hybrid automata are hierarchical finite state machines with real-valued variables that denote the time. They define, unambiguously, the specific combinations of events that trigger or stop the execution of the different elements of the model. Temporal constraints, expressed as state invariants and guards over transitions, define when the discrete transitions occur.

AADL offers a binding mechanism to assign software components (data, thread, process, etc.) to execution platform components (memory, processor, buses, devices, etc.). Each software component can define several possible bindings and properties that may have different values depending on the actual binding. Execution platform components support the execution of threads, the storage of data and code, and the communication between threads. This execution model in AADL encodes the most effective structures used by embedded systems developers and assumed by the theory of real-time systems. Concurrent executions are modeled using threads managed by a scheduler. The dispatch protocol (periodic, aperiodic, sporadic and background) determines when an active thread executes its computation.

In AADL, communications can be immediate or delayed. A immediate communication means that the dispatch time for sending thread and receiving thread is the same. A delayed communication means that the value from the sending thread is transmitted at its deadline and is available to the receiving thread at its next dispatch. For delayed communications, additional constraints are needed to get a deterministic schedule. Several criteria can be considered, like for instance, the size of the buffer used for the communication, or applying a well-known scheduling policy, like Earliest Deadline

First (EDF).

AADL permits software and execution platform components to be organized into hierarchical structures with well-defined interfaces. An AADL description is almost always hierarchical, with the topmost component being an AADL system that contains, for example, processes and processors, where the processes contain threads and data, and so on. Besides, components may be hierarchical, i.e. they may contain other components. Compared to other modeling languages, AADL defines low-level abstractions including hardware descriptions. These abstractions are more likely to help design a detailed model close to the final product.

Even though AADL supports multiform time models, since it focuses on the runtime architecture, the functional structure is extracted away. It results in the lack of model elements to describe the application itself, independently of the resources. Besides, AADL requires large amount of details to capture even simple systems. Comparing to AADL, UML activities allow for a description of the application, actions executed sequentially or concurrently, without knowing, at first, whether actions are executed by a periodic thread or a subprogram. So in the next section we investigate MARTE: a UML extension on real-time systems.

2.7 MARTE

The UML profile for Modeling and Analysis of real-time and embedded systems (MARTE) [92] is a special extension of UML for modeling real-time embedded systems. MARTE defines a broadly expressive Time Model to provide for a generic timed interpretation of UML models. The time model is based on partial ordering of instants. MARTE precisely defines a semantics within UML profile rather than allowing tools, possibly incompatible with other tools of the same domain, to support time modeling. MARTE OMG specification introduces a time structure inspired from time models of the concurrency theory [33] and proposes a new clock constraint specification language (CCSL) to specify, within the context of UML, usual logical and chronometric time constraints.

The clock constraint specification language (CCSL) [7] has been introduced to specify timed annotations on UML diagrams and thus provides them with formally defined timed interpretations. CCSL offers a general set of notations to specify causal, chronological and timed properties and has been used in various sub-domains [65] [66] [9]. CCSL is intended to be used at various modeling levels following a refinement strategy. It allows both coarse, possibly non-deterministic, infinite, unbounded specifications at the system level but also more precise specifications from which code generation, schedulability and formal analysis are possible. Thus, MARTE, as a profile of UML, presents a time model in a more precise and clear manner than UML for the design of real-time embedded systems (RTES). And it provides more precise expression of domain-specific phenomena such as mutual exclusion mechanisms, concurrency, deadline specifications, and so on.

In MARTE, time can be physical, and considered as dense or discretized, but it can also be logical, and related to user-defined clocks. Time may even be multiform, allowing different times to progress in a non-uniform fashion, and possibly independently to any (direct) reference to physical time. In real world technical systems, special devices, called clocks, are used to measure the progress of physical time. In MARTE, a clock, which can be chronometric or logical, is a model giving access to the time structure. MARTE qualifies a clock referring to physical time as a chronometric clock, emphasizing on the quantitative information attached to this model. A logical clock mainly addresses concrete instant ordering, making reference to a timebase.

MARTE explicit time model with powerful logical time constraints allows to specify precisely and thoroughly the scheduling aspects of application elements. Timed processing is a generic concept for modeling activities that have known start and finish times, or a known duration. For a timed message, start and finish events are respectively named as sending and receipt events. A delay is a special kind of timed action that represents a null operation lasting for a given duration. It is used to obtain delayed signals according to a faster clock. For example "Clock C = A delayedFor n on B" specifies a clock C that has all its instants coincident with the n^{th} instant of B that follows an instant of A. The MARTE time model allows multiform/polychronous time modeling, which is inspired by synchronous languages. It supports modeling and analysis of component-based architectures, as well as a variety of different computational paradigms (asynchronous, synchronous, and timed). MARTE enables the specification of not only real-time constraints but also other embedded systems characteristics, such as memory capacity and power consumption. Furthermore, MARTE can be used to check the communication and causality path correctness by introducing event relations into models. Paper [82] proposed a technique for transforming MARTE/CCSL mode behaviors into timed automata so that a system can be checked by the model-checking tool UPPAAL. This approach enables verification of both logical and chronometric properties of the system.

MARTE, as a standard model-based description for real-time and embedded systems, provides a way to specify several aspects of embedded systems, ranging from large software systems on top of an operating system to specific hardware designs. It provides a support to capture structural and behavioral, functional and non-functional aspects by including CCSL. However, MARTE, as an extension of UML, keeps some drawbacks from it. As we know, UML provides a set of diagrams to depict software structures graphically. These diagrams appeal to practitioners and help them tackle complex software structures. Even though its individual diagrams are useful to depict software structures, UML cannot fully define the relationships between diagrams. The diagrams are developed as separate entities that express different aspects of the software, not as parts of a common construct. Thus, when using MARTE, a designer is able to model a system with multiple functional, runtime, and hardware diagrams. Then, connections between the diagrams are used to model the allocation of entities from one diagram to another. However, the consistency across diagrams is largely left to be resolved by the designer. Besides, UML is large and complex. It comprises many different concepts and semantics that we do not need. Since we mainly focus on communication behaviours and would like to keep the semantic as simple as possible,

2.8 STeC

Spatio-temporal consistence language (STeC) [39] is a spatio-temporal consistence language for real-time systems. It provides a location-triggered specification in which agents are specified with location and time constraints. The spatio-temporal consistence means that an agent, e.g., a mobile device, executes a task when it arrives at a required location or time. The location is an abstract concept, which can be a physical address, an IP address, a channel or others.

Similar to CSP (Communicating Sequential Processes), STeC handles the interactions between agents by two atomic communication commands "Send" and "Get". The difference between CSP and STeC is that STeC includes time and location variables. And STeC defines guards as actions and statuses of agents as well as their logical compositions. STeC handles two kinds of interrupts: time break and interaction break. Following the Dijkstra's guard style, in STeC, nondeterministic choice phase is guarded by communications. Syntax and semantics of the language have been proposed to address the issue of spatial-temporal consistence. Based on it, STeC defines denotational semantics [89] for describing distributed systems with time and location constraints. The language specifies the time and location constraints for each action, and then computes the execution time of processes.

STeC language is able to specify real-time systems especially for the consistence of location and time. However, our model has a quite different goal compared to STeC. Our model timed-pNets mainly focuses on time properties. We set the time information as parameters that rely on a reference clock. Even though we need check car's locations in our use-case, but such data is not treated at the same level as the time information. This is quite different from what the STeC does by adding location constraints. Moreover, since we mainly focus on modeling the communication behavior of distributed systems, we abstract location information as parameters and highlight the synchronous and asynchronous communications, which also lead us taking different way to model timed-systems.

2.9 Conclusion

These previous efforts are of importance since they provide crucial insights on building timed-models for real-time systems. Their mechanisms and strategies contribute to build our model.

Discrete-event models inspire the use of specifying system behaviours by taking advantage of events. The events that trigger the communications can be used to build a logical view of system behaviour. The investigation of globally asynchronous locally synchronous (GALS) model including HipHop help us to have a deep understanding of handling synchronous and asynchronous communications. They provide sophisticated mechanism for coordinating the synchronous and asynchronous communications. However, since we use pNets as our untimed framework, we do not take FSM (as GALS did) to model the synchronous component. Instead, we wrap system events into logical clocks and design the timed specifications (a set of logical clocks and clock relations) to model synchronous and asynchronous communications. Moreover, GALS focuses on orchestrating complex concurrent behaviours and discusses the correctness of system behaviours, but we need to take account the time constraints of these behaviours to analyze the system time properties. BIP, as a framework for the incremental composition of heterogeneous components, helps us to understand how they introduce the time concept into its model. With contrast to BIP that takes a special port for the synchronization of its components, we choose to build synchronous relations between events so that we can flexible setting the synchronization on components and furthermore on events.

The other timed model like timed-automata, timed petri net, MARTE and AADL provide us a broad view on modelling real-time systems. For example, timed-automata succeeds on using real-valued clocks (whose values increase all with the same speed) to specify time constraints. It is very popular in industry for modelling real-time embedded systems. However, since we would like to avoiding using common physical time in our model, we choose a different way to model time constraints. In our model, we wrap events as logical clocks and then set time constraints by means of these clocks and clock relations. This idea is close to the MARTE model that introduces CCSL as its timed model. However, MARTE is a framework in the software level to model real-time system. Since we would like to build a low level behaviour model in the sense that it is able to express behaviour mechanisms for different various of languages or component models, we choose pNets model and introduce timed specification into it. Other time related systems like STeC are also be investigated to see how they specify time and location constraints for actions.

These previous works provide us a global view of the current situation of timed models, and their mechanism of handling time and asynchronous communications.

Chapter 3 pNets With Timed-Actions and Logical Constraints

In this chapter, we solve two issues. One is how to define timed-actions and introduce them into pNets. Another one is how to define logical clocks and clock relations so that logical time constraints can be specified in our model.

We propose timed-actions by adding time variables into actions. The variables are used to record delay time of actions. We then define a logical clock as a set of occurrences of a timed-action. A logical clock is a mechanism for capturing chronological and causal relationships in distributed systems. Usually multiple logical clocks in a system are dependent. We define clock relations to specify the dependence and interactions among these logical clocks. Thanks to logical clocks and clock relations, our model can keep track of the order of timed-actions that occur at each process, and ensure that these timed-actions are assigned by consistent logical times.

The chapter is structured as follows. Section 3.1 introduces the definitions of timed-actions, logical clocks, clock relations as well as our timed model. Then in section 3.2, we take a simple use case from ITS to demonstrate the formalism of our model. TimeSquare tool is used to simulate the system and check its properties. In the end, in section 3.3, we give a conclusion for this chapter.

3.1 Model Building

3.1.1 Timed Actions

We follow the pNets assumption on Action algebra $\mathcal{L}_{\mathcal{A},\mathcal{P}}$ which includes all required operators for building action expressions in the language ($\mathcal{P}$ is a set of parameters used to build open expressions, typically expressing data variables) [13]. In our model, we define $\mathcal{L}_{\mathcal{A},\mathcal{P},\mathcal{T}}$ as the timed-action algebra, in which $\mathcal{T}$ is a set of (discrete) timed variables. We denote for example α ($\in \mathcal{L}_{\mathcal{A},\mathcal{P},\mathcal{T}}$) as an action name, then we consider α, $!\alpha(m)$ and $?\alpha(m)$ as timed-actions. α means that the timed-action executes locally but not delivers messages. $!\alpha(m)$ ($m \in \mathcal{P}$) is denoted as sending a message and $?\alpha(m)$ ($m \in \mathcal{P}$) as receiving a message.

Definition 1 (Timed-Actions) timedAction Let $\mathcal{T}$ be a set of discrete time variables with domains in the natural numbers $\mathbb{N}$. $\mathcal{B}_{\mathcal{T}}$ is the set of closed intervals (bounds) over time variables. The Timed-action Algebra $\mathcal{L}_{\mathcal{A},\mathcal{T},\mathcal{P}}$ is an action set built over $\mathcal{T}$ and $\mathcal{P}$. We call $\alpha(p)^{t|b} \in \mathcal{L}_{\mathcal{A},\mathcal{T},\mathcal{P}}$ a timed-action in which $\alpha \in \mathcal{A}$ is an action, $p \in \mathcal{P}$ is a parameter, $t \in \mathcal{T}$ is a time variable describing a time delay before the action can be executed, $b \in \mathcal{B}_{\mathcal{T}}$ is a delay bound of t.

We set $\alpha^0 = \alpha$, which means the action α is always ready. As an example, $a^{t|[1,3]}$ means the action a cannot be executed until t units times are passed.

Since the next two chapters (chapter 3 and chapter 4) do not discuss the delay bounds, for simplification, we do not represent them in the two chapters. However, the bound intervals will be exposed in the chapter 5 where we will investigate the delay bounds.

3.1.2 Logical Constraints

We define a *Clock* as a sequence of occurrences of a timed-action. The clock, in the sense of CCSL, is a logical clock. The logical clock means the distance between occurrences is not related with the passage of physical time.

Definition 2 (Clock) A Clock C_α is a sequence of occurrences of a timed-action $\alpha(p)^t$. We write:
$C_\alpha = \{\alpha(p_1)^{t_{\alpha_1}}\_1, \alpha(p_2)^{t_{\alpha_2}}\_2, \ldots, \alpha(p_i)^{t_{\alpha_i}}\_i, \ldots\}$ $(i \in \mathbb{N})$, in which $\alpha(p_i)^{t_{\alpha_i}}\_i$ denotes the i^{th} occurrence on clock C_α.

For simplification, in this thesis, an occurrence $\alpha(p_i)^{t_{\alpha_i}}\_i$ can be denoted as α_i for short when not ambiguous.

Clock Relations A Clock Relation defines the relation between two clocks. We take the syntax and semantics of clock relations from [64], which is a language to express time constraints by defining clock relations in timed models. The clock relations include: $=$ (coincidence), $\prec$ (strict precedence),$\preceq$(precedence), $\subseteq$ (subclock), $\sharp$ (exclusion). They are defined as follows:

- $C_\alpha = C_\beta$ (C_α coincides with C_β), which means clock C_α ticks if and only if clock C_β ticks.
- $C_\alpha \prec C_\beta$ (C_α strictly precedes C_β), which means $\forall k$ $(k \in N)$, the k^{th} occurrence of C_α strictly precedes the k^{th} occurrence of C_β.
- $C_\alpha \preceq C_\beta$ (C_α precedes C_β), which similar to the previous one. The only difference is that the clock C_α can tick as late as C_β ticks.
- $C_\alpha \subseteq C_\beta$ (C_α is a subclock of C_β), which means clock C_β must tick at the same time as clock C_α ticks.
- $C_\alpha \sharp C_\beta$ (C_α excludes C_β), which means none of their occurrences coincide.

3.1.3 Introduce Logical Clocks into pNets Model

In pNets, the leaves are pLTSs. To construct pNets with logical clock constraints, we first introduce logical clocks into pLTS [13]. These logical clocks are built from timed-actions. The following definition represents a Logical pLTS in which each transition is triggered by a timed-action.

Definition 3 (Logical pLTS) A Logical pLTS is a tuple $< P, S, s_0, A, C, \rightarrow >$, where

- P is a finite set of parameters
- S is a set of states
- $s_0 \in S$ is the initial state
- A is a set of timed-actions
- C is a set of logical clocks over the timed-action set A
- $\rightarrow$ is the set of transitions: $\rightarrow \subseteq S \times A \times S$. We write $s \overset{\alpha}{\rightarrow} s'$ for $(s, \alpha, s') \in \rightarrow$, in which $\alpha \in A$, $C_\alpha \in C$.

The next definition extends the classical pNets definition from [13] by introducing clock constraints. The pNets retain a hierarchical structure and a parameterization of subnets: holes in a pNet can be instanciated by a variable number of subnets (e.g. a number of logical pLTSs). Then synchronisation vectors allow very flexible and expressive multi-way synchronisation mechanisms, that naturally we extend here with clock constraints.

Definition 4 (Clock Constrained pNet) A Clock Constrained pNet is a tuple $< P, A_G, R_G, J, C, \widetilde{O}_J, \widetilde{R}_J, \overrightarrow{V} >$, where:

- $P = \{p_i / p_i \in Dom_i\}$ is a finite set of parameters
- $A_G \subseteq \mathcal{L}_{\mathcal{A},\mathcal{T},\mathcal{P}}$ is a set of global actions
- C is a set of clocks for all timed-actions
- R_G is a set of relations between actions taken from each subnet
- J is a countable set of argument indexes: each index $j \in J$ is called a hole and is associated with a sort $O_j \subseteq \mathcal{L}_{\mathcal{A},\mathcal{T},\mathcal{P}}$ and a set of clock constraints $\widetilde{R}_J$
- $\overrightarrow{V} = \{\overrightarrow{v}\}$ is a set of synchronous vectors of the form:

* (binary communication) $\overrightarrow{v} =< ..., !a^{t1}_{[k_{i1}]}, ..., ?a^{t2}_{[k_{i2}]}, ... >\rightarrow (a^{t_g}_g)$, in which $a^{t_g}_g \in A_G, k_{i1} \in Dom_1, k_{i2} \in Dom_2, !a^{t1} \in O_{i1}, !a^{t2} \in O_{i2}$, $C_{!a^{t1}}, C_{!a^{t2}}, C_{a^{t_g}_g} \in C$, $t_g = max\{t1, t2\}$
* or (visibility) $\overrightarrow{v} =< ..., a^{t1}_{[k1]}, ... >\rightarrow (a^{t_g}_g)$, in which $a^{t_g}_g \in A_G, k1 \in Dom_1, a^{t1} \in O_{i1}, C_{!a^{t1}} \in C, t_g = t1$

Remark: We define the model in a form inspired by the synchronisation vectors of Arnold and Nivat [10], that we use to synchronise clocks from different processors. One of the main advantages of using its high abstraction level is that almost all interaction mechanisms encountered so far in the process algebra literature become particular cases of a very general concept: synchronisation vectors. We structure the synchronisation vectors as parts of network. Contrary to synchronisation constraints, the network allows dynamic reconfigurations between different sets of synchronisation vectors. In our model, we define two kinds of synchronous vectors. One is a binary communication vector. The vector represents the communication of two holes through timed-action $!a^{t1}_{[k_{i1}]}$ and $?a^{t2}_{[k_{i2}]}$. The two timed-actions that come from different holes stay between two symbols “<” and “>”. The last element of the vector appears behind the symbol “→”. It is a global timed-action generated by this synchronous vector. Another vector (called visibility) makes a local timed-action (e.g. $a^{t1}_{[k1]}$) visible by generating a global timed-action (e.g. $a^{t_g}_g$).

3.2 Simulation

In this section we build a timed model for the use case named Vehicle-to-Infrustructure Communication given in section 1.5.1 in page 22. Fig. 3.1 presents its architecture in which cars and infrastructures are distributed nodes. Every Car consists of three sub components: a sensor, a controller and a brake component. Sensors are used to detect the current locations and speeds of cars and to receive control signals from infrastructures. Controllers receive signals from sensors and then call brake components to exe-

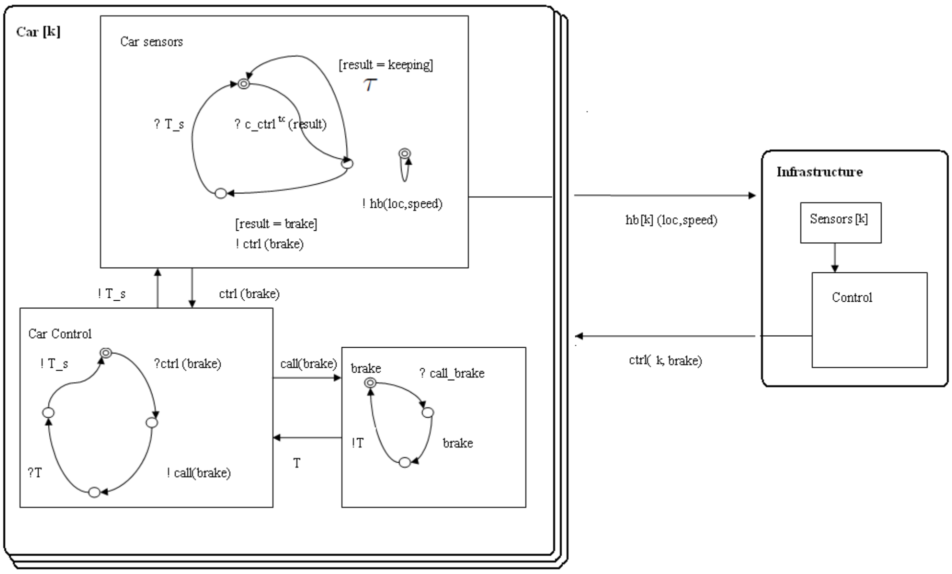

Fig. 3.1: Timed-pNets architecture with details of the car's subcomponents

cute brake operations if necessary. Local communications between these sub components of cars are synchronized in the sense that sending actions and receiving actions coincide. The LTSs of these sub components are shown in Fig. 3.1. We specify sensors by two LTSs: one describes periodical emissions of heartbeat signals to report the locations and speeds of cars; another describes reactions to control signals.

3.2.1 Formalisation of the Architecture

Here we explain how to formalize this use case. We build our model with two holes: one is for receiving an arbitrary number of cars; another represents a single Infrastructure. We assume that the communications between the two holes are asynchronous, while in each hole, the communications between its subcomponents (e.g. Sensors, Controllers) are synchronous. We list the formalisation of this system as follows.

$< P, A_G, R_G, J, \widetilde{C}_J, \widetilde{O}_J, \widetilde{R}_J, \overrightarrow{V} >$

$P = \{k : \mathbb{N}, loc : \mathbb{R}, speed : \mathbb{R}, brake : bool\}$

$A_G = \{CI\_hb^{t_h}(k, loc, speed), CI\_ctrl^{t_{ct}}(k, brake)\}$

$J = \{car[k], infrastructure\}$

$O_{Car} = \{!c\_hb^{t_{hb\_c}}(loc, speed), ?c\_ctrl^{t_{ct\_c}}(brake), !call(brake), T_s, ...\}$

$O_{Infrastructure} = \{?I\_hb^{t_{hb\_I}}(k, loc, speed), !I\_ctrl^{t_{ct\_I}}(k, brake),$
$!Isensor\_hb^{t_{hb\_I}}[k](loc, speed),$
$?Icontrol\_hb_I^{t_I}(k, loc, speed), ...\}$

$R_G = \{!c\_hb^{t_{hb\_c}}[k](loc, speed) \prec ?I\_hb^{t_{hb\_I}}(k, loc, speed);$
$!I\_ctrl^{t_{ct\_I}}(k, brake) \prec ?c\_ctrl^{t_{ct\_c}}[k](brake);\}$

$\overrightarrow{V} :< O_{car}[k], O_{infrastrcuture} > \rightarrow A_{Car\_infrastrcuture}$
$= <!c\_hb^{t_{hb\_c}}[k](loc, speed), ?I\_hb^{t_{hb\_I}}(k, loc, speed) > \rightarrow CI\_hb^{t_h}(k, loc, speed);$
$<?c\_ctrl^{t_{ct\_c}}[k](brake), !I\_ctrl^{t_{ct\_I}}(k, brake) > \rightarrow CI\_ctrl^{t_{ct}}(k, brake).\}$

An interesting point is that the Infrastructure receives independent heartbeats from the Cars, that are subsequently interleaved within the Infrastructure structure. This is expressed by a clock relation on the link between the sensors and control in the Infrastructure structure: $!Isensor\_hb^{t_{hb\_I}}[k](loc, speed) \subseteq ?Icontrol\_hb_I^{t_I}(k, loc, speed)$. This relation tells that the heartbeat signals transmitted by the k^{th} Sensor component are the subset of the heartbeat signals received by the Control component.

Finally, we take a Car Sensor component as an example to represent its clock relations:

$R_{CarSensor} = \{hb(loc, speed) \triangleq idealClockdiscretizedByrate$ (1);
$(\tau \sharp !ctrl(brake))$ (2);
$?c\_ctrl(result) \prec (\tau \wedge !ctrl(brake))$ (3);
$!ctrl(brake) \prec ?T\_s$ (4);
$(?T\_s[i] \vee \tau[i]) \prec ?c\_ctrl(result)[i+1]$ (5);$\}$

where (1) describes that heartbeat signals are sent periodically; (2) indicates that the events τ and $!ctrl(brake)$ are exclusive; (3) denotes that the event $?c\_ctrl(result)$ always precedes the event τ and $!ctrl(brake)$; (4) tells us that the event $!ctrl(brake)$ precedes the event $?T\_s$; (5) explains that the events in the i^{th} cycle precedes those in the $(i+1)^{th}$ cycle.

3.2.2 Result

We use TimeSquare [41] to simulate the clock relations and check its logic correctness. The input of TimeSquare is a CCSL file including clock relations, bound requirements and properties. The tool proceeds with a symbolic simulation, and generates a trace model. Output files (text and graph) are generated to display the traces and eventually show if properties are satisfied.

In our use-case, in order to check time properties, apart from the clock relations, we also need to specify boundary requirements. Let a heartbeat interval be "hi" that is defined as the distance between two adjacent heartbeat occurrences ($hb\_(i+1) - hb\_i$). We set communication delay bounds, computation delay bounds and a deadline requirement as follows:

- The minimum and maximum communication delays between the cars and the infrastructure are $(1/5)hi$ and $(3/5)hi$;
- The computation delays are no more than $(2/5)hi$;
- Each heartbeat signal should be processed before sending the next heartbeat.

Through this simulation, we check if all heartbeat signals finally can be processed before their deadlines. We formalize the property as $(?T\_s\_i \prec hb\_(i+1)) \vee (\tau\_i \prec hb\_(i+1))$, which means that the action $?T\_s$ or τ of the i^{th} cycle occurs before the heartbeat signal of the $(i+1)^{th}$ cycle.

The result of this simulation is shown in Fig.3.2, in which a red vertical line is the deadline of a cycle. The figure tells that the property is not satisfied since an occurrence of the action $?T\_s$ is later than its deadline. One reason that cause the failure might be the large latency of communication delay or computation delay. After we modify the maximum computation boundary from "$(2/5)hi$" to '$(1/5)hi$", we found out that the property is satisfied.

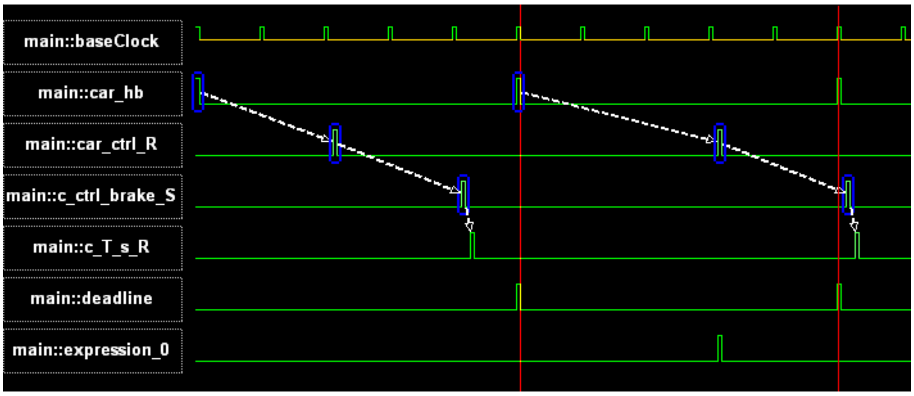

Fig. 3.2: property checking

3.3 Conclusion

In this chapter, we defined a novel behavior semantic model by introducing logical clocks and clock relations to pNets model.

In this new model, logical clocks are derived from timed-actions. Clock relations are specified in terms of the logical relations of timed-actions. Besides, we take advantage of synchronous vectors to flexibly specify synchronous communications.

A simple use case taken from Intelligent Transport Systems is used to explain our approach, including how to formalize the system, how to check time properties by TimeSquare tool. From the result of the simulation, we conclude that this new approach helps to check system logical correction as well as some time properties.

However, the different ways to specify local constraints (by synchronous vectors) and global constraints (precedence relations) make it more difficult to build a hierarchical structure. Besides, this model is not so compact since synchronous vectors and global relations handle time constraints on actions.

Therefore, in the next chapter, based on this first attempt, we will improve the current one to make it more compact. Besides, we will take care of the structure of the model so that it will be flexible enough to adapt the component-based design approaches.

Chapter 4 Timed-pNets Model

This chapter represents a communication behavioural semantic model called timed-pNets that is an extension of the previous model we proposed in the chapter 3.

The main contributions of this chapter are as follows. First, we develop the extended model timed-pNets with a tree-style hierarchical structure. Its leaves are represented by timed-pLTSs. Its non-leaf nodes (called timed-pNet nodes) are synchronisation devices that synchronize the behaviours of subnets (these subnets can be leaves or non-leaf nodes). Second, we let all nodes (leaves or non-leaf nodes) associate with timed specifications. A timed specification is a set of logical clocks and clock relations. By proposing the solutions of translating timed-pLTSs and timed-pNets to timed specifications, we can analyze our model by investigating the hierarchical timed specifications. Third, we design channels to model asynchronous communications instead of directly using precedence relations. Thus, by using synchronous vectors and channels, we can specify the communication behaviours (synchronous and asynchronous communications). Last but not least, we update the synchronous vectors from action-based synchronous vectors to clock-based synchronous vectors so that we can handle a set of synchronous actions for each vectors.

The use case "Vehicle-to-Vehicle Communication" taken from the section 1.5.2 in page 23 is used to explain the timed-pNets model including the notations, definitions and theorems. In the end, we simulate the system and check the validity of this model.

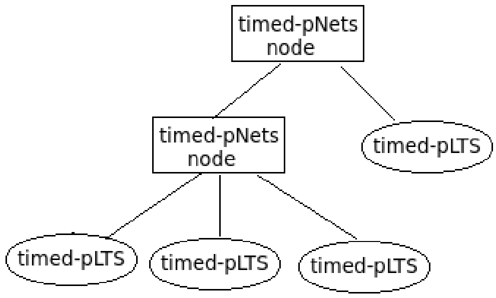

Fig. 4.1: Timed-pNets tree structure

4.1 Context and problematic

In the previous chapter we proposed our first attempt on the time constrained model, including the notions of logical clocks imported from CCSL. A set of clock relations were designed to describe the system constraints. However, this model is not sufficient to build hierarchical timed specifications starting from timed-pLTSs.

In this chapter, we enhance the compositional aspects of our specification methodology: a system is modelled as hierarchy of timed-pNets as Fig.4.1, where leaves are timed-pLTSs, i.e. finite state transition systems with logical clocks on the transitions, and nodes are synchronisation devices. Products between subnets can be synchronous (modelling local components sharing synchronous clocks), or involve asynchronous communications between unrelated events, that we model as channels.

From such a hierarchical model, we propose procedures for:

- at the bottom level, analyzing timed-pLTSs, and build the timed specifications (sets of clocks and clock constraints) encoding its temporal behaviours;

- for each timed-pNets node, building an abstract timed specification (= at level N), from its lower-level timed specifications (level N-1).

One important point is that Timed Specifications (TSs) are logical characterizations, that can be either provided by the application designer, or computed from the model. The consequence is that the two procedures above can be used arbitrarily in a bottom-up fashion, starting with detailed timed-pLTS and assembling them in a compatible way; or in a top-down fashion, constructing TSs for abstract timed-pNets, using their holes TSs as hypotheses in an assume-guarantee style, and providing later some specific (compatible) implementations for these holes in various contexts.

In the end, we are able to use the TimeSquare tool [41] to simulate the possible executions of timed specifications.

This rest of the chapter is organized as follows. Section 4.2 describes the meaning of timed specifications including the formal definitions of timed-actions, logical clocks and their relations. Then we give the definition of timed-pLTSs in section 4.3. In section 4.4, we discuss how to build timed-pNets. The procedure of generating timed specifications from timed-pLTSs and timed-pNets are presented in section 4.5. The issue of checking the compatibility of timed-pNets is discussed in section 4.6. In section 4.7 we discuss how to build multi-layer timed-pNets systems. Then in section 4.8 we represent the simulations by using the TimeSquare tool. Finally, the chapter ends with conclusions and future researches.

4.2 Timed Specification

In this section, we present the preliminary denotations and definitions of timed-actions, logical clocks, clock relations and timed specifications. We shall use the example presented in section 1.5.2 in page 23 to illustrate all definitions and results.

We define a *logical clock* as a sequence of occurrences of a timed-action. The clock, in the sense of CCSL, is a logical clock. The logical clock means that the distance between occurrences is not related with the passage of real time.

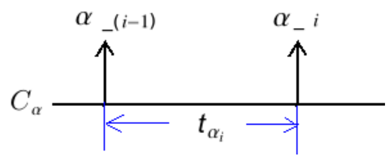

Fig. 4.2: count the delay t_{α_i} when C_α is an independent clock

Definition 5 (Logical Clock) A Logical Clock C_α is a sequence of occurrences of a timed-action $\alpha(p)^t$. We write:
$C_\alpha = \{\alpha(p_1)^{t_{\alpha_1}}\_1, \alpha(p_2)^{t_{\alpha_2}}\_2, \ldots, \alpha(p_i)^{t_{\alpha_i}}\_i, \ldots\}$ $(i \in \mathbb{N}^+)$, in which $\alpha(p_i)^{t_{\alpha_i}}\_i$ denotes the i^{th} occurrence of clock C_α.

For simplification, in our thesis, an occurrence $\alpha(p_i)^{t_{\alpha_i}}\_i$ can be denoted as α_i for short when not ambiguous.

The assignment of the delay variable t_{α_i} in each occurrence $\alpha(p_i)^{t_{\alpha_i}}\_i$ can be different. The delay variable captures the minimum time (delay) that an action must wait before it can occur after the previous action. More precisely when a clock is independent (has no precedence relation with another clock), the delay is counted from the previous occurrence of the same action as shown in the Fig. 4.2. If a clock C_β directly precedes a clock C_α, then the delay of the i^{th} occurrence of the timed-action α is counted from the i^{th} occurrence of the timed-action β as shown in the Fig. 4.3. The relation of coincidence (discussed in the next subsection) does not effect on the way of counting the delay. For example, if there is another clock C_γ that coincides with the clock C_α, then the delay t_{α_i} is still be counted as shown in the Fig.4.3.

For convenience, we define here two clock expressions, time shift, and filtering:

Definition 6 (Clock Offset) Let C_α be a clock built over a timed-action α, $C_\alpha[i]$ be the i^{th} occurrence of the clock C_α. The n^{th} offset of the clock C_α is the clock defined as: $C_\alpha^{\Delta(n)} = \{C_\alpha[n+1]\_1, C_\alpha[n+2]\_2, \ldots, C_\alpha[n+i]\_i, \ldots\}$.

From the definition we can see that the $(n+1)^{th}$ occurrence of C_α becomes the first occurrence of the new clock $C_\alpha^{\Delta(n)}$, and so on.

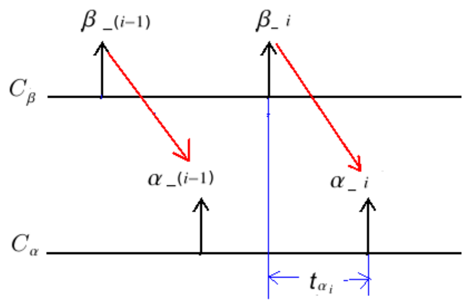

Fig. 4.3: count the delay t_{α_i} when $C_\beta \prec C_\alpha$

Definition 7 (Clock Filtering) Assume N' is a subset of $\mathbb{N}$. Let C_α be a clock built over a timed-action α. The new clock that is filtered from the clock C_α by N' is denoted as $C_\alpha^{N'} = \{C_\alpha[i_1]\_1, C_\alpha[i_2]\_2, \ldots C_\alpha[i_k]\_j, \ldots\}(i_1, i_2, \ldots i_k, \ldots \in N', i_1 < i_2 < \ldots < i_k, \ldots, j, k \in \mathbb{N})$.

For convenience, we will write the filter N' either as a boolean function over $\mathbb{N}$, or as a subset of $\mathbb{N}$, e.g.: $C_\alpha^{\{2n-1\}_{n\in\mathbb{N}}}$ accepts only the odd occurrences of the clock C_α. $C_\alpha^{\{n\geq 8\}}$ filters out the first 8 occurrences.
So if $C_\alpha = \{\alpha(p_1)^{t_{\alpha_1}}\_1, \alpha(p_2)^{t_{\alpha_2}}\_2, \ldots, \alpha(p_i)^{t_{\alpha_i}}\_i, \ldots\}$,
then $C_\alpha^{\{2n-1\}_{n\in\mathbb{N}}} = \{\alpha(p_1)^{t_{\alpha_1}}\_1, \alpha(p_3)^{t_{\alpha_3}}\_2, \ldots \alpha(p_{(2n-1)})^{t_{\alpha(2n-1)}}\_n, \ldots\}$. $C_\alpha^{\{n\geq 8\}} = \{\alpha(p_8)^{t_{\alpha_8}}\_1, \alpha(p_9)^{t_{\alpha_9}}\_2, \ldots\}$.

Finally we define Timed Specifications: a timed specification is composed of a set of logical clocks, together with a set of clock relations, expressing the temporal ordering constraints between the clocks. This is an abstract specification in the sense that it captures just enough information to check the time safety (validity of time requirements) of a system, and the compatibility relations required for assembling sub-systems together. In the next sections we shall describe procedures to compute the timed specifications of systems (timed-pLTSs and timed-pNets), and to check the compatibility.

Definition 8 (Timed Specification) Let $\mathcal{I}_C$ be the set of occurrences of the clock C. A Timed Specification is a pair $< \mathfrak{C}, \mathcal{R} >$ where $\mathfrak{C}$ is a set of clocks, $\mathcal{R}$ is a set of clock relations on $\bigcup_{C\in\mathfrak{C}} \mathcal{I}_C$.

4.2.1 Syntax and Semantic of Clock Relations

A Clock Relation defines the relation between two clocks. With respect to the original definition of clock relations in CCSL [7], we have slightly different goals, and different needs. In particular we do not need exclusion (that is most important with some families of reactive formalisms). We do not define "subclock" relation in this paper because we need a more concrete way to define how to build a new subclock from original one. Instead, we defined "clock filtering" which can specify the way of selecting action occurrences. Therefore, here we only define two relation operations ('$\prec$', '$=$') to describe the different dependence relations between clocks.

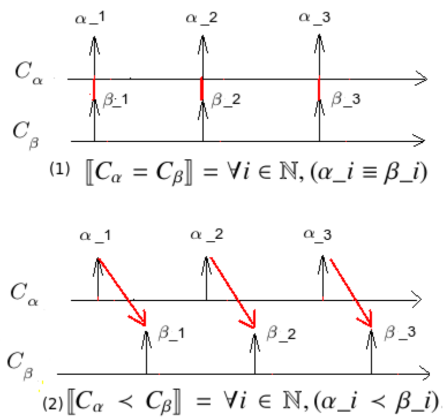

Fig. 4.4: Constraints

- The relation '$C_\alpha = C_\beta$' (C_α coincides with C_β) describes the strict synchronization of clocks. It means that the occurrence of C_α appears if and only if the occurrence of C_β appears. In other words, the clock C_α and C_β tick at the same time. Formally, $[\![C_\alpha = C_\beta]\!] = \forall i \in \mathbb{N}, (\alpha\_i \equiv \beta\_i)$ (shown in Fig. 4.4(1)). This operator can naturally be used to describe synchronous communications.

- The relation '$C_\alpha \prec C_\beta$' (C_α precedes C_β) describes the precedence relation of clocks. It says that the action β from the clock C_β cannot occur until the corresponding action α in the clock C_α occurs. In another word, clock C_α ticks always earlier than clock C_β. Formally $[\![C_\alpha \prec C_\beta]\!] = \forall i \in \mathbb{N}, (\alpha\_i \prec \beta\_i)$. As shown in Fig. 4.4(2), the i^{th} occurrence of the clock C_α always appears earlier than the i^{th} occurrence of the clock C_β. The relation usually relates to the causality induced by an asynchronous communication.

4.2.2 Properties of the logical clock relations

Not surprisingly, these relations have their expected properties: coincidence is an equivalence relation, and precedence is a strict preorder.

Proposition 1 (Properties of the Coincidence Relation "="). *Given a set of clocks C . The relation "=" on the set C is reflexive, symmetric and transitive.*

Proof: This follows from the fact that "$\equiv$" is an equivalence relation on timed-action occurrences.
(1) Choose any clock $C_\alpha \in \mathfrak{C}$. Let its i^{th} ($i \in \mathbb{N}$) occurrence be α_i. Obviously, $\forall i$, the occurrence α_i coincides with itself. So we know $C_\alpha = C_\alpha$; the coincidence relation is reflexive. (2) Now choose another clock $C_\beta \in \mathfrak{C}$. If we have the relation $C_\alpha = C_\beta$, then we know that $\forall i \in \mathbb{N}$, $\alpha\_i \equiv \beta\_i$, which means the action α occurs if and only if the action β occurs. According to the symmetric relation of the operator "$\equiv$", we know that the action β occurs if and only if the action α occurs. So we have $\forall i \in \mathbb{N}$, $\beta\_i \equiv \alpha\_i$. We know $C_\beta = C_\alpha$; the coincidence relation is symmetric. (3) choose another clock $C_\gamma \in \mathfrak{C}$. If we have the relations $C_\alpha = C_\beta$ and $C_\beta = C_\gamma$, then $\forall i \in \mathbb{N}$, $\alpha\_i \equiv \beta\_i \wedge \beta\_i \equiv \gamma\_i$. From the transitivity relation of "$\equiv$", we infer $\forall i \in \mathbb{N}, \alpha\_i \equiv \gamma\_i$; so we know $C_\alpha = C_\gamma$; the coincidence relation is transitive. $\square$

Proposition 2 (The properties of Precedence Relation "$\prec$"). *Given a clock set $\mathfrak{C}$. The relation "$\prec$" on the set $\mathfrak{C}$ is transitive, but not reflexive, not symmetric.*

This follows from the same properties on the relation $\prec$ on occurrences. The proofs are similar to those of Proposition 1.

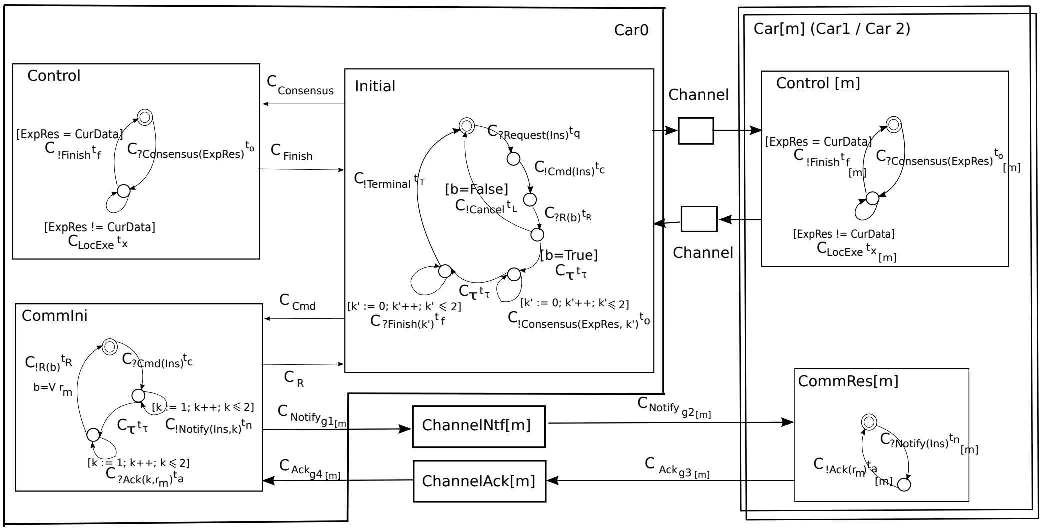

Fig. 4.5: Communication Behaviour Model of Cars Insertion Scenario

Proposition 3 (Substitutivity of "="). *Given four clocks C_α, C_β, C_γ, C_η which are built on the timed-action α, β, γ and η separately. Let $C_\alpha = C_\beta$ and $C_\gamma = C_\eta$. If $C_\alpha \prec C_\gamma$, then we have $C_\beta \prec C_\eta$.*

Proof. According to the coincidence definition, $C_\alpha = C_\beta \Rightarrow \forall i, \alpha\_i \equiv \beta\_i$, and $C_\gamma = C_\eta \Rightarrow \forall i, \gamma\_i \equiv \eta\_i$. If $C_\alpha \prec C_\gamma$, then according to the precedence definition, we know $\forall i, \alpha\_i \prec \gamma\_i$, which means the action α always occurs earlier than the action γ. Since $\forall i, \alpha\_i \equiv \beta\_i$ tells us the action α occurs if and only if the action β occurs, so we know β always occurs earlier than γ ($\forall i, \beta\_i \prec \gamma\_i$). Similar, since $\forall i, \gamma\_i \equiv \eta\_i$ tells us the action γ occurs if and only if the action η occurs, so we furthermore have the relation $\forall i, \beta\_i \prec \eta\_i$. According to precedence relation definition, we get $C_\beta \prec C_\eta$. □

Example 1 In this example, we illustrate how to represent timed-actions, clocks, and clock relations for our "car inserting" scenario presented in the section 1.5.2 in page 23. As shown in the Fig. 4.5, on-board car systems are modeled by several components including "Initial", "CommIni" "CommRes", "Control", etc. (In this figure we only show the components that participate in the protocol.) In the example, the procedure starts with a user's request by sending an *"insertion"* order (encoded here as a "$!Request(Ins)^{t_q}$" timed-action) to the "Initial" component. Then the procedure runs in two phases:

(1) The agreement phase: $car0$ sends a $notify(Ins)$ message to the other two cars, and waits for their answers. This phase is managed by the "CommIni" process, that communicates to the "ComRes" processes of other cars through asynchronous channels. In the model, there is one such channel for each type of messages, and for each pair of communicating processes; we use the parametrized structure of pNets to represent such families of processes, e.g. "channelNtf[m]" in the figure 4.5. The "CommIni" process is in charge of collecting the answers from the other cars asynchronously, and sending the final decision to "Initial". If it is negative, then "Initial" aborts and signals $Cancel$ to the user, otherwise we go to the next phase.
(2) The execution phase: this phase is triggered and controlled directly by the "Initial" process. It sends $C_{!Consensus(ExpRes)^{t_o}}$ to all cars including itself to initiate the execution and to tell them the final expected result ("ExpRes"). The "Control" process of each car is in charge of the local Execution of the movement (that we leave unspecified here), till the expected result is observed ($[ExpRes = CurData]$). Then the $!Finish$ signals are collected by "Initial", and termination is notified to the user.

We use label transition systems (LTSs) to model each component. Each transition will be triggered by a clock. Precedence relations are used to specify the causality relations of LTSs. For example, in the "CommRes" component, the clock "$C_{?notify(Ins)^{t_n}}$" occurs earlier than the clock "$C_{!ack(r_m)^{t_a}}$". We denote the clock relation as "$C_{?notify(Ins)^{t_n}} \prec C_{!ack(r_m)^{t_a}}$". For simplification, in the following sections, we will omit the parameters and time variables when expressing a clock relation if it is not ambiguous. For example, we use the short version "$C_{?notify} \prec C_{!ack}$" instead of "$C_{?notify(Ins)^{t_n}} \prec C_{!ack(r_m)^{t_a}}$".

In this use-case, we assume for simplicity that the communication inside a car is synchronous (in realistic modern car systems, this hypothesis would have to be refined, since the onboard systems include several process communicating through data buses). Here, the timed-action "$!Cmd(par)^{t_c}$" in the "Initial" process and the timed-action "$?Cmd(par)^{t_c}$" in "CommIni" are always synchronous when the two components communicate and transmit the message "par". So the two clocks coincide
($C_{Initial.!Cmd(par)^{t_c}} = C_{CommIni.?Cmd(par)^{t_c}}$).

By contrast, the communications between two different cars are asynchronous (typically over some wireless ad-hoc network). For this we insert a specific asynchronous channel (built as a special timed-pLTS) between cars for each type of messages exchanged between them.

The two mechanisms illustrate our approach to model heterogeneous (synchronous/asynchronous communication) systems. In the next section, we show how we formalise this by using the timed-pNets formalism.

4.3 Timed-pLTS

This section introduces timed transition systems (timed-pLTSs), including the special cases: channels. We illustrate each definition with a piece of the running example.

Definition 9 (Timed-pLTS) A Timed-pLTS is a tuple $< P, S, s_0, A, \mathfrak{C}, \rightarrow >$, where

- P is a finite set of parameters
- S is a set of states
- $s_0 \in S$ is the initial state
- A is a set of timed-actions
- $\mathfrak{C}$ is a set of clocks over the timed-action set A
- $\rightarrow$ is the set of transitions: $\rightarrow \subseteq S \times \mathfrak{C} \times S$. We write $s \xrightarrow{C_\alpha} s'$ for $(s, C_\alpha, s') \in \rightarrow$, in which $\alpha \in A, C_\alpha \in \mathfrak{C}$.

Example 2 Consider the "CommIni" component in Fig. 4.6. The clock relations will correspond to the precedence (causality) relations between the transitions of the LTS, with a special case for the loops on states s_1 (a state for sending notifications) and s_2 (a state for receiving "ack" signals), where the communication events are indexed by $k \in [1..N]$ (N is the (fixed) number of neighbors of the initiating car (here $N = 2$)). The first loop on s_1 means that $car0$ sends a notification signal to car1 and car2 separately. The second loop on s_2 means that $car0$ collects "ack" signals from car1 and car2. Moreover,

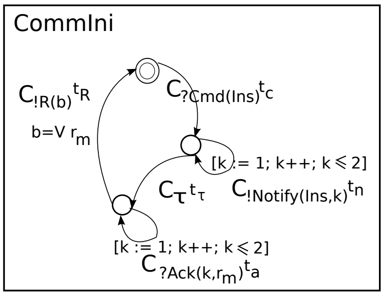

Fig. 4.6: The timed-pLTS of the CommIni component

we use the silent action τ to build a clock $C_{\tau}^{t_{\tau}}$ that labels the transition to state s_2 when the component finishes sending two notifications. We build the timed-pLTS elements as:

- Parameters $P = \{k, Ins, r_m, b, N\}$,
- Action algebra $A = \{?Cmd(par)^{t_c}, !notify(par)^{t_n}, ?ack(k, r_m)^{t_a}, !R(b)^{t_R}, \tau^{t_{\tau}}\}$
- Clocks $\mathfrak{C} = \{C_{?Cmd}, C_{!notify}, C_{?ack}, C_{!R}, C_{\tau}\}$
- (we do not detail the clock relations here, they can be easily deduced from the figure)

Note that the system designers only need specify the timed-pLTSs. The clock relations can be automatically deduced from the timed-pLTSs (see section 4.5.1).

Channels. We introduce channels to model asynchronous communication behaviours. A channel is defined as a special transition system with two timed-events: one for receiving messages, another for sending messages. The two events have a precedence constraint which models the delay of message transmission. For simplification, the channel definition here just describes a simple one place asynchronous buffer, sufficient to illustrate the heterogeneity of synchronous and asynchronous communications. More realistic asynchronous mechanisms are possible (e.g. n-places buffers, lousy channels, or ProActive/GCM request queues with futures [34] but they are not the topics of this thesis).

Definition 10 (Channel) A <u>channel</u> is a transition system with tuple $< P, S, A, \mathfrak{C}, \prec, \rightarrow >$ in which

- P is a finite set of parameters,
- S is state set in which $S = \{s_{empty}, s_{data}\}$,
- $A = \{?in(par)^{t_i}, !out(par)^{t_o}\}$ $(par \in P)$ is the timed-action set,
- $\mathfrak{C}$ is a set of clocks over timed-actions A,
- $\rightarrow$ is a set of two transitions: $s_{empty} \xrightarrow{C_{?in}} s_{data}$ and $s_{data} \xrightarrow{C_{!out}} s_{empty}$.

In the channel definition, the timed-action $?in(par)^{t_i}$ is an action for receiving messages from one component, while the timed-action $!out(par)^{t_o}$ is an action for sending the messages to another component as shown in Fig. 4.7.

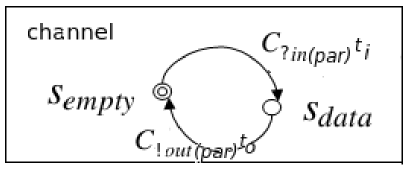

Fig. 4.7: The timed-pLTS of channel Component

4.4 Timed-pNets

Finally we define Timed-pNets, that are our main structure used to combine sub-systems to build bigger systems. Similar to the original (untimed) pNets, a Timed-pNet is a generalized composition operator, defining the synchronization between a number of subsystems (holes). In timed-pNets, holes are characterized by action algebra (a sort); here it is complemented by a Timed Specification. Building a timed-pNet tree representing a full system requires filling holes with (compatible) sub-nets.

Definition 11 (Timed-pNets) A <u>Timed-pNet</u> is a tuple $< P, A_G, \mathfrak{C}_G, J, \widetilde{A}_J, \widetilde{\mathfrak{C}}_J, \widetilde{\mathcal{R}}_J, \overrightarrow{V} >$, where:

- P is a finite set of parameters,
- A_G is a set of global timed-actions, and $\mathfrak{C}_G$ is the set of global clocks that are built over A_G,
- J is a countable set of argument indexes: each index $j \in J$ is called a hole and is associated with a set of local timed-actions A_j, and an associated Timed Specification $< \mathfrak{C}_j, \mathcal{R}_j >$.
- $\overrightarrow{V} = \{\overrightarrow{v}\}$ is a set of synchronization vectors of the form:

- (binary communication between holes j_1 and j_2)
  $\overrightarrow{v} =< \ldots, C_{!\alpha}, \ldots, C_{?\alpha}, \ldots > \rightarrow C_g$, [1]
  in which $\{C_{!\alpha} = C_{?\alpha} = C_g\}, C_g \in \mathfrak{C}_G, C_{!\alpha} \in \mathfrak{C}_{j_1}, C_{?\alpha} \in \mathfrak{C}_{j_2}, j_1, j_2 \in J$,

- or (visibility from hole j)
  $\overrightarrow{v} =< \ldots, C_{\alpha}, \ldots > \rightarrow C_g$, in which $\{C_{\alpha} = C_g\}, C_g \in \mathfrak{C}_G, C_{?\alpha} \in \mathfrak{C}_j, j \in J$.

  Furthermore, each global clock can be generated by only one synchronization vector:
  $\forall \quad \overrightarrow{v_i}, \overrightarrow{v_{i'}} \in \overrightarrow{V}, C_{gi} = C_{gi'} \implies \overrightarrow{v_i} = \overrightarrow{v_{i'}}$
  (C_{gi}(resp. $C_{gi'}$) be a global clock generated by the vector $\overrightarrow{v_i}$ (resp. $\overrightarrow{v_{i'}}$), $i, i' \in \mathbb{N}$)

Remark: We define Nets in a form inspired by the synchronisation vectors of Arnold and Nivat [10], that we use to synchronise clocks from different processors. One of the main advantages of using its high abstraction level is that almost all interaction mechanisms encountered so far in the process algebra literature become particular cases of a very general concept: synchronisation vectors. We structure the synchronisation vectors as parts of network. Contrary to synchronisation constraints, the network allows dynamic reconfigurations between different sets of synchronisation vectors. In our timed-pNets, we define two kinds of synchronous vectors. One is the

[1] where "..." represents an arbitrary number of holes that do not participate in this synchronization

communication vector ($< \ldots, C_{!\alpha}, \ldots, C_{?\alpha}, \ldots > \to C_g$). The vector represents the communication of two holes through clock $C_{!\alpha}$ and $C_{?\alpha}$. The two local clocks that come from different holes are put between the two symbols "$<$" and "$>$". The last element of the vector appears behind the symbol "$\to$", and specifies the global clock generated by this synchronous vector. Another vector ($< \ldots, C_{\alpha}, \ldots > \to C_g$) makes the local clock C_{α} visible by generating a global clock C_g. For both kinds of synchronous vectors, the local clocks (that appear between "$<$" and "$>$") are transparent to the upper layer nodes. Only the global clocks (the last elements in the synchronous vectors) can be observed from the upper level. These global clocks can be used for building a higher level timed-pNets node.

Moreover, from the definition 11 we can see that the synchronous vectors only catch the coincidence relations between clocks (for describing synchronous communications), which makes our timed-pNets models cannot directly specify asynchronous communications. So when modelling asynchronous communications, we need to introduce channels into systems. The two subsystems that asynchronously communicate with each other are connected by a channel in which a communication delay is modelled. Example 3 shows us how to take advantage of channels to specify asynchronous communications.

Notations for parameterized systems. In practice, we use parametric notations, both for holes and for synchronization vectors, making the notations more compact and more user-friendly. These are only abbreviations, their meaning must be understood as a (finite) expansion of the structure.

Using such abbreviations, for a pNet in which j_1, j_2, j are parametric holes with indexes k_1, k_2, k, with respective domains Dom_1, Dom_2, Dom, the synchronization vectors will look like:

- binary communication

  Depending on the combination of actions from j_1 and j_2, this vector will generate a family of global clocks indexed by a parameter k, that is a function of k_1 and k_2. The domain of k is a subset of the product

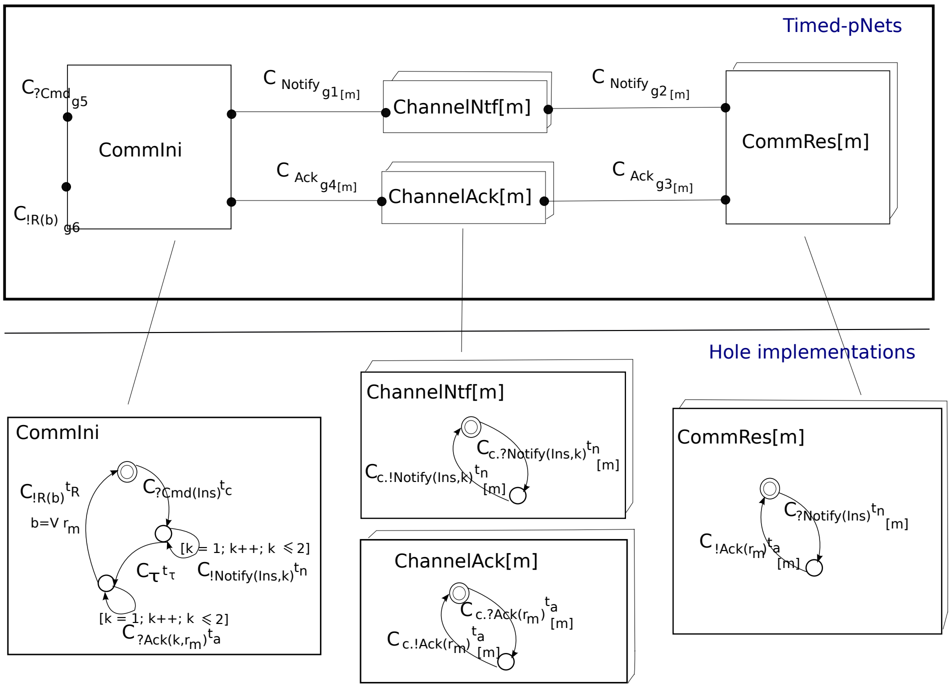

Fig. 4.8: A Timed-pNets with one of its implementations

$Dom_1 \times Dom_2. < ..., C_{!\alpha[k_1]}, ..., C_{?\alpha[k_2]}, ... > \rightarrow C_{g[k]}$,
in which $\{C_{!\alpha[k_1]} = C_{?\alpha[k_2]} = C_{g[k]}\}$, $C_{g[k]} \in \mathfrak{C}_G, C_{!\alpha[k_1]} \in \mathfrak{C}_{j_1}, C_{?\alpha[k_2]} \in \mathfrak{C}_{j_2}$

- visibility

  Each visible action from hole j generates a corresponding global clock. $< ..., C_{\alpha[k]}, ... > \rightarrow C_{g[k]}$, in which $\{C_{\alpha[k]} = C_{g[k]}\}, C_{\alpha[k]} \in \mathfrak{C}_j, C_{g[k]} \in \mathfrak{C}_G$.

Example 3 We go on the use case to illustrate how to build a timed-pNets model. To make the example smaller, we have extracted here the respective "communication" subNets of 2 cars, and the channels on which they communicate, and we show how to build a pNet encoding this small subsystem.

As shown in the Fig.4.8, the subsystem consists of components "CommIni", "CommRes[m]", "ChannelNtf[m]" and "ChannelAck [m]". The components "ChannelNtf[m]" and "ChannelAck[m]" are channels in which the parameter "[m]" denotes to which car the corresponding channel transmits

data. By using the parameter "m", we give a more compact representation of the model. According to our scenario, $car0$ sends a notification to $car1$ (resp. $car2$) via "ChannelNtf[1]" (resp."ChannelNtf[2]"), and then $car1$ (resp.$car2$) answers an "ack" to $car0$ via "ChannelAck[1]" (resp. ChannelAck[2]"). So in the upper layer timed-pNets nodes, we can link these components by building synchronous vectors. For example:
- the vector[2] $< -, C_{!ack_{[1]}}, -, C_{c.?ack_{[1]}} > \rightarrow C_{ack_{g3_{[1]}}}$ represents the communication between the components "CommRes[1]" and "ChannelAck[1]" and generates the global clock "$C_{ack_{g3_{[1]}}}$". Notice that even though we actually have 7 subnets (CommIni, CommRes[1], CommRes[2], ChannelNtf[1], ChannelNtf[2], ChannelAck[1], ChannelAck[2]), by using parameters, we represent our pNet and its synchronous vectors with only 4 holes.
- the vector $< C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}}, -, C_{c.?notify_{[1]}}, - > \rightarrow C_{notify_{g1_{[1]}}}$ represents the communication between the components "$CommIni$" and "$ChannelNtf[1]$" and builds a global clock "$C_{notify_{g1_{[1]}}}$" (remember $C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}}$ is the clock built from the clock $C_{!notify}$ by choosing the occurrences with odd indexes).

Following the timed-pNets definition, we can formalize this timed-pNets as follows:

- $P = \{k, Ins, m, r_m, b\}$,
- $A_G = \{notify(Ins, k)_{g1_{[m]}}^{t_{g1}}, notify(Ins, k)_{g2_{[m]}}^{t_{g2}}, ack(r_m, k)_{g3_{[m]}}^{t_{g3}}, ack(r_m, k)_{g4_{[m]}}^{t_{g4}}, ?Cmd(Ins)_{g5}^{t_{g5}}, !R(b)_{g6}^{(t_{g6}}\}$
- $\mathfrak{C}_G = \{C_{notify_{g1_{[m]}}}, C_{notify_{g2_{[m]}}}, C_{ack_{g3_{[m]}}}, C_{ack_{g4_{[m]}}}, C_{?Cmd_{g5}}, C_{!R_{g6}}\}$
- $J = \{CommIni, CommRes[m], ChannelNtf[m], ChannelAck[m]\}(m := 1, 2)$

Next we formalize the Timed Specifications of these holes as:

- For the hole "CommIni":

  $A_{CommIni} = \{?Cmd(Ins)^{t_c}, !notify(Ins, k)^{t_n}, ?ack(k, r_m)^{t_a}, !R(b)^{t_R}\}$

  $\mathfrak{C}_{CommIni} = \{C_{?Cmd}, C_{!notify}, C_{?ack}, C_{!R}\}$

[2] where "−" represents a single hole that does not participate in this synchronization

$$\mathcal{R}_{CommIni} = \{C_{?Cmd} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify}, \quad C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{!notify},$$
$$C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack}, \quad C^{\{2s\}_{s\in\mathbb{N}}}_{!notify} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{?ack},$$
$$C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack} \prec C^{\{2s\}_{s\in\mathbb{N}}}_{?ack}, \quad C^{\{2s\}_{s\in\mathbb{N}}}_{?ack} \prec C_{!R} \prec C^{\Delta(1)}_{?cmd}\}$$

- For the hole "CommRes[m]" (m := 1, 2):

$$A_{CommRes[m]} = \{?notify(Ins,k)^{t_n}_{[m]}, !ack(k,r_m)^{t_a}_{[m]}\}$$

$$\mathfrak{C}_{CommRes[m]} = \{C_{?notify_{[m]}}, C_{!ack_{[m]}}\}$$

$$\mathcal{R}_{CommRes[m]} = \{C_{?notify_{[m]}} \prec C_{!ack_{[m]}} \prec C^{\Delta(1)}_{?notify_{[m]}}\}$$

- For the hole "ChannelNtf[m]" (m := 1, 2):

$$A_{ChannelNtf[m]} = \{c.?notify(Ins,k)^{t_{n1}}_{[m]}, c.!notify(Ins,k)^{t_{n2}}_{[m]}\}$$

$$\mathfrak{C}_{ChannelNtf[m]} = \{C_{c.?notify_{[m]}}, C_{c.!notify_{[m]}}\}$$

$$\mathcal{R}_{channelNtf[m]} = \{C_{c.?notify_{[m]}} \prec C_{c.!notify_{[m]}} \prec C_{c.?notify^{\Delta(1)}_{[m]}}\}$$

- For the hole "ChannelAck[m]" (m := 1, 2):

$$A_{ChannelAck[m]} = \{c.?ack(k,r_m)^{t_{a1}}_{[m]}, c.!ack(k,r_m)^{t_{a1}}_{[m]}\}$$

$$\mathfrak{C}_{ChannelAck[m]} = \{C_{c.?ack_{[m]}}, C_{c.!ack_{[m]}}\}$$

$$\mathcal{R}_{channelAck[m]} = \{C_{c.?ack_{[m]}} \prec C_{c.!ack_{[m]}} \prec C^{\Delta(1)}_{c.?ack_{[m]}}\}$$

In the end, we specify the synchronous vectors:

$\overrightarrow{V} = \{$

$V_1 :< C^{\{2s-1\}_{s\in\mathbb{N}}}_{!notify(Ins,k=1)}, -, C_{c.?notify(Ins)_{[1]}}, - > \to C_{notify_{g1_{[1]}}},$

$V_2 :< -, C_{?notify_{[1]}}, C_{c.!notify_{[1]}}, - > \to C_{notify_{g2_{[1]}}},$

$V_3 :< -, C_{!ack_{[1]}}, -, C_{c.?ack_{[1]}} > \to C_{ack_{g3_{[1]}}},$

$V_4 :< C^{\{2s-1\}_{s\in\mathbb{N}}}_{?ack(k=1,r_m)}, -, -, C_{c.!ack(r_m)_{[1]}} > \to C_{ack_{g4_{[1]}}}$

$V_5 :< C^{\{2s\}_{s\in\mathbb{N}}}_{!notify(Ins,k=2)}, -, C_{c.?notify(Ins)_{[2]}}, - > \to C_{notify_{g1_{[2]}}},$

$V_6 :< -, C_{?notify_{[2]}}, C_{c.!notify_{[2]}}, - > \to C_{notify_{g2_{[2]}}},$

$V_7 :< -, C_{!ack_{[2]}}, -, C_{c.?ack_{[2]}} > \to C_{ack_{g3_{[2]}}},$

$V_8 :< C^{\{2s\}_{s\in\mathbb{N}}}_{?ack(k=2,r_m)}, -, -, C_{c.!ack(r_m)_{[2]}} > \to C_{ack_{g4_{[2]}}}$

$V_9 :< C_{?Cmd}, -, -, - > \to C_{?Cmd_{g5}},$

$V_{10} :< C_{!R}, -, -, - > \to C_{!R_{g6}}\}$

Discussion: The Timed specification of holes. Let us now argue how the timed specifications of this upper-level timed-pNet holes may have been specified, in a top-down approach, before building their timed-pLTS implementations. This, intuitively, is done from the informal description of the scenario and the knowledge of the top level component and communication architecture:

Take the "CommIni" component as an example, the scenario related to the component is:

(1) the component "CommIni" gets a change-lane request by clock $C_{?cmd}$ from the "Initial" component;

(2) the component "CommIni" sends requests by clock $C_{!notify}$, in sequence, to $car1$ and $car2$ to get agreements;

(3) the component "CommIni" collects results from $car1$ and $car2$ by clock $C_{?ack}$;

(4) the component reports result to "Initial" component by clock $C_{!R}$.

Since the step (1) happens earlier than the step (2), the clock $C_{?cmd}$ must precede the clock $C_{!notify}$. Then, in our use case, the component "CommIni" sends notification signal twice, so we have clock relation $\{C_{?Cmd} \prec C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{!notify}^{\{2s\}_{s\in\mathbb{N}}}\}$. In generally, if there are N neighbors, the clock relation should be $\{C_{?Cmd} \prec C_{!notify}^{\{Ns-(n-1)\}_{s\in\mathbb{N}}} \prec C_{!notify}^{\{Ns-(n-2)\}_{s\in\mathbb{N}}} \prec \ldots \prec C_{!notify}^{\{Ns\}_{s\in\mathbb{N}}}\}$. Similar to the step (2), since the component receives "ack" signal twice, so we have the clock relation $\{C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}}\}$. Furthermore, the clock $C_{!notify}$ in step (2) should precede the clock $C_{?ack}$ in step (3), so we have the relations $C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}}$ and $C_{!notify}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}}$. Finally the scenario goes to the step (4), we have the relation $\{C_{?ack}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{!R}\}$. Since the scenario is repeatable, we specify the clock relation $\{C_{!R} \prec C_{?cmd}^{\Delta(1)}\}$. In the end, we conclude:

$$\begin{aligned}\mathcal{R}_{\{CommIni\}} = \{&C_{?Cmd} \prec C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}},\quad C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{!notify}^{\{2s\}_{s\in\mathbb{N}}},\\ &C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}},\quad C_{!notify}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}},\\ &C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}},\quad C_{?ack}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{!R} \prec C_{?cmd}^{\Delta(1)}\}\end{aligned}$$

In the section 4.6, we will show that these Timed Specifications are indeed fulfilled by the corresponding timed-pLTS “CommIni”, “ComRes”, “ChannelNtf”, and “ChannelAck”.

4.5 Generating Timed Specification

4.5.1 Generating TS of timed-pLTS

As we see in the Fig.4.8, timed-pLTSs are concrete implementations of those holes. In order to check the compatibility, we need to generate timed specifications for those concrete timed-pLTSs. Here we propose rules to automatically generate a timed specification from the LTS part of a timed-pLTS. More precisely, given the action algebra and the transition relations of a timed-pLTS, we compute its set of clocks, and the relations between these clocks.

This procedure runs in 4 phases as shown in the Fig. 4.9. The inputs of the procedure include a timed-pLTS and a set of rules that tell how to set the occurrence relations and its index functions. In step 1, we traverse the timed-pLTS and generate a “symbolic” table that gathers all possible causally related pairs of transitions of the timed-pLTS, and the corresponding relations between clock occurrences. In step 2 we go through the symbolic table and build a “concrete” table in which each column represents one specific “round” of execution through the symbolic table (with concrete index assignments). In the concrete table, guards of the timed-pLTS can be resolved, so some of the symbolic transitions may be eliminated. In step 3 we generate a general formula for each relation. In the end (step 4), we lift those occurrence relations to clock relations, and generate the Timed Specification.

4.5.2 Auxiliary functions: Pre/Post sets

Before describing Step 1, we need to define the functions computing the pre/post sets of the timed-pLTS states.

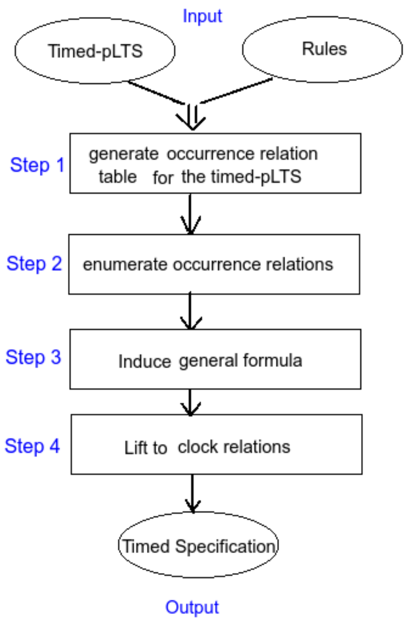

Fig. 4.9: Steps for generating the TS of a timed-pLTS

For a timed-pLTS transition system $< P, S, s_0, A, \mathfrak{C}, \rightarrow>$, we denote $PreAct(s, s')$ the set of direct preceding timed-action occurrences of s from s'; and $PostAct(s, s')$ the set of direct succeeding timed-action occurrences of state s towards state s'. Then we denote $PreAct(s)$ (resp. $PostAct(s)$) as the set of all direct preceding (resp. succeeding) timed-action occurrences of state s. Furthermore, we define $PreActIndex(s)$ (resp. $PostActIndex(s)$) as the sum of the indexes of the set of preceding (resp. succeeding) timed-action occurrences of state s. The sum corresponds to the cases where branches in the LTS allow some executions to go several times through alternative transitions out of some states. Formally:

Definition 12 (Preceding Timed-Action Occurrences) Let $< P, S, s_0, A, \mathfrak{C}, \rightarrow>$ be a timed-pLTS transition system. For $s \in S$ and $\alpha(p)^{t_\alpha} \in A, (p \in P)$, the direct preceding timed-action occurrence of s is defined as $PreAct(s, s') = \{\alpha\_i | s' \xrightarrow{C_\alpha} s, \alpha\_i \in C_\alpha, \} \ (s, s' \in S)$. The set of direct preceding timed-action occurrences of s is defined as $PreAct(s) = \bigcup_{s' \in S} PreAct(s, s')$. Furthermore, we denote the index of a preceding timed-action occurrence as $PreActIndex(s, s') = \{i | s' \xrightarrow{C_\alpha} s, \alpha\_i \in C_\alpha(s, s' \in S)\}$, and the sum of

State	Transition	Occurrence Relations	Index Assignment
s_0	$tr0 : s_2 \xrightarrow{C_{!R}} s_0 \xrightarrow{C_{?Cmd}} s_1$	$!R\_m \prec ?Cmd\_n$	$f_{tr0} : n = m + 1$
s_1	$tr1 : s_0 \xrightarrow{C_{?Cmd}} s_1 \xrightarrow{C_{\tau}} s_2$	$?Cmd\_n \prec \tau\_r$	$f_{tr1} : r = n$
	$tr2 : s_0 \xrightarrow{C_{?Cmd}} s_1 \xrightarrow{C_{!Notify}} s_1$	$?Cmd\_n \prec !notify\_i$	$f_{tr2} : i := i + 1$
	$tr3 : s_1 \xrightarrow{C_{!Notify}} s_1 \xrightarrow{C_{!Notify}} s_1$	$!notify\_i \prec !notify\_(i + 1)$	
	$tr4 : s_1 \xrightarrow{C_{!Notify}} s_1 \xrightarrow{C_{\tau}} s_2$	$!notify\_i \prec \tau\_r$	$f_{tr4} : r = n$
s_2	$tr5 : s_1 \xrightarrow{C_{\tau}} s_2 \xrightarrow{C_{!R}} s_0$	$\tau\_r \prec !R\_m$	$f_{tr5} : m = r$
	$tr6 : s_1 \xrightarrow{C_{\tau}} s_2 \xrightarrow{C_{?Ack}} s_2$	$\tau\_r \prec ?ack\_j$	$f_{tr6} : j := j + 1$
	$tr7 : s_2 \xrightarrow{C_{?Ack}} s_2 \xrightarrow{C_{?Ack}} s_2$	$?Ack\_j \prec ?ack\_(j + 1)$	
	$tr8 : s_2 \xrightarrow{C_{?Ack}} s_2 \xrightarrow{C_{!R}} s_0$	$?Ack\_j \prec !R\_m$	$f_{tr8} : m = r$

Fig. 4.10: Time assignment for the Timed-pLTS “Car.CommIni”

the indexes of a set of preceding timed-action occurrences of state s as $PreActIndex(s) = \sum_{s' \in S} PreActIndex(s, s')$.

Definition 13 (Succeeding Timed-Action Occurrences) Let $< P, S, s_0, A, \mathfrak{C}, \rightarrow>$ be a timed-pLTS transition system. For $s \in S$ and $\alpha(p)^{t_\alpha} \in A, (p \in P)$, the direct succeeding timed-action occurrence of state s is defined as $PostAct(s, s') = \{\alpha\_i | s \xrightarrow{C_\alpha} s', \alpha\_i \in C_\alpha\}$, $(s, s' \in S)$. The set of direct succeeding timed-action occurrences of state s is defined as $PostAct(s) = \bigcup_{s' \in S} PostAct(s, s')$. Furthermore, we denote the index of a succeeding timed-action occurrence as $PostActIndex(s, s') = \{i | s \xrightarrow{C_\alpha} s', \alpha\_i \in C_\alpha\}$, $(s, s' \in S)$, and the sum of the indexes of a set of succeeding timed-action occurrences of s as $PostActIndex(s) = \sum_{s' \in S} PostActIndex(s, s')$.

4.5.3 Relations and assignment rules

The computation in Step 1 is based on a set of rules identifying specific configurations of the states in the timed-pLTS traversal. For each such configuration, we define a rule that expresses the relation(s) between the set of preceding and succeeding clock occurrences of the current state, and the changes in the clock occurrence indexes.

The main configurations are: initial state, in which we have to initialize indexes, and increase an index each time when the system goes through a new global round; standard state in which we register the increase of one of the involved index; and looping states, in which we have to take care of the guards for entering/leaving loops, in terms of a specific "loop counter".

We define a restrictive notion of looping states, which is reasonable configurations for timed analysis. A looping state may have one or more loops of arbitrary length, but coming back to the same state. And each loop must start with a transition with a guard taking the precise form of a "loop counter" control, namely `[k=1; k++; k ≤ kMax]` for some counter variable `k`, in which `kMax` may be a natural number, or a variable. Loop guards can share a loop counter (see e.g. Fig. 4.11), so several loops will be executed the same number of times; otherwise different loop counters must be independent. Of course one could imagine more complex structures for our timed-pLTSs, but this restriction already covers a lot of interesting cases, and make the generation of the Times Specification easier.

In these rules, for simplification, we represent relations on two sets (S_1 (resp. S_2) is a set of occurrences of clocks): $S_1 \prec S_2$ means $\forall \alpha_m \in S_1, \beta_n \in S_2, \alpha_m \prec \beta_n$ $(m, n \in \mathbb{N})$.

(1) **Initial state**. If $PreAct(s_0) \notin \emptyset$, then $PreAct(s_0) \prec PostAct(s_0)$,
[**Assign:** $PostActIndex(s_0) \Leftarrow PreActIndex(s_0) + 1$];

(2) **Standard state**. $\forall s \backslash s_0$, $PreAct(s) \prec PostAct(s)$,
[**Assign:** $PostActIndex(s) \Leftarrow PreActIndex(s)$];

(3) **Looping state**. $\forall s$, if $\exists \alpha . s \xrightarrow{C_\alpha} s$ and the loop executes N times, then

(3.1) go inside the loop
$PreAct(s) \prec \alpha\_i$,
[**Assign:** $i := i + 1$]

(3.2) stay in the loop,
$\alpha\_i \prec \alpha\_(i + 1)$

(3.3) leave the loop:

(3.3.1) leave the loop to another loop, e.g. $\exists \beta . s \xrightarrow{C_\beta} s$ ($\beta\_j \in PostAct(s, s) \backslash \alpha\_i$):
$\alpha\_i \prec \beta\_j$,
[**Assign:** $j := j + 1$]

(3.3.2) to one post-action out of $PostAct(s, s_0)$:
$\alpha\_i \prec PostAct(s) \backslash PostAct(s, s_0)$,
[**Assign:**
$PostActIndex(s) \Leftarrow PreActIndex(s)$].

(3.3.3) to one post-action in $PostAct(s, s_0)$:
$\alpha\_i \prec PostAct(s, s_0)$,
[**Assign:**
$PostActIndex(s) \Leftarrow PreActIndex(s) + 1$].

4.5.4 The Method for Generating Timed Specification

This subsection introduces a method of generating a timed specification from a timed-pLTS. We state two algorithms and 4 steps.

Step 1: generate occurrence relations table

The algorithm 1 uses the rules above to build an occurrence relation table. More precisely each row in the table lists a specific pair of Pre/Post transitions of a state, with the corresponding occurrence relation and the index increase function deduced from the corresponding rule.

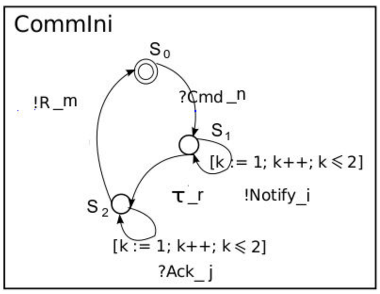

Fig. 4.11: Simplification of CommIni Component

Algorithm 1 Generate occurrence relations table

Input: a timed-pLTS graph and rules.

Output: A table of occurrence relation with its index assignment function.

for each state s_i in LTS graph **do**
  for each pair (s_1, s_2) such that $s_1 \xrightarrow{C_1} s_i \xrightarrow{C_2} s_2$ **do**
    insert a row with $State = s_i$, $Transition = s_1 \xrightarrow{C_1} s_i \xrightarrow{C_2} s_2$.
    if $s_i = s_0$ **AND** s_i has no self-loop **then**
      apply case (1) rules, adding the relations and assignments in the corresponding rows.
    end if
    if $s_i \neq s_0$ **AND** s_i has no self-loop **then**
      apply case (2) rules
    end if
    if s_i includes one self-loop **then**
      if $s_i = s_0$ **then**
        apply case (1), (3.1), (3.2) and (3.3.3) rules
      else
        apply case (2), (3.1), (3.2) and (3.3.2) rules
      end if
    else
      if $s_i = s_0$ **then**
        apply case (1), (3.1), (3.2), (3.3.1) and (3.3.3) rules
      else
        apply case (2), (3.1), (3.2), (3.3.1) and (3.3.2) rules
      end if
    end if
  end for
end for

Example 4 Let us take the "CommIni" component from Fig. 4.6 as an example. We first transform Fig. 4.6 into Fig. 4.11 by removing all parameters but adding index variables. Then we generate occurrence relations for each state. For example, we take the state "s_0", from the timed-pLTS graph we get the transitions $s_2 \xrightarrow{C_{!R}} s_0 \xrightarrow{C_{?Cmd}} s_1$. According to the rule (1) we have $!R\_m \prec ?Cmd\_n$ and the assignment $n = m + 1$ ($n, m \in \mathbb{N}$). Take the state s_1 as another example. Since it includes a self-loop, we apply the rules (2), (3.1), (3.2) and (3.3.2). When a transition directly brings to the next state without passing the loop, according to the rule (2), we have the relation $?Cmd\_n \prec\ \tau\_r$ and the assignment $r = n$. When a transition enters the loop, according to the rule (3.1), we have the relation $?Cmd\_n \prec !notify\_i$ and the assignment $i := i + 1$ ($i \in \mathbb{N}$). When a transition stays in the loop, according to the rule (3.2), we can get the relation "$!notify\_i \prec !notify\_i + 1$" ($i \in \mathbb{N}$). Then when a transition leaves the loop, according to the rule (3.3.2), we have the relation $!notify\_i \prec \tau\_r$ and the assignment $r = n$ ($r \in \mathbb{N}$).

Step 2: Enumerate occurrence relations

Now we go through the symbolic occurrence table built in step 1 and build a "concrete" table in which each column represents one specific "round" of execution through the symbolic table (with concrete index assignments). In the concrete table the guards of the timed-pLTS can be resolved, so some of the symbolic transitions (rows of the table) may be eliminated.

In the guards (including the loop control guards), there may be some parameters occurring in a symbolic form. Before we run the algorithm in step 2, we need to instantiate these parameters, to be able to compute the guards. In particular the maximum value of the loop counters (in our use-case, corresponding to the number of neighbor cars) must be fixed.

Moreover, we must set a bound (N) to the number of rounds that we shall unfold in the algorithm. This bound should be large enough for the generalization procedure in step 3 to work properly.

For each round of traveling, we compute a set of occurrence relations. The indexes of these occurrences tell the (logical) times of the actions that have occurred till this round. For loops, the loop control guard says that if a transition satisfies the initial condition "$k = 1$", then the transition goes into the loop. Each time after executing the loop, the variable k increases by 1. Then the transition continues to execute the loop till the condition $k \leq kMax$ is not satisfied.

We present algorithm 2 to enumerate these relations. The results of the algorithm are illustrated in the table in Fig. 4.12 in which the r^{th} column presents a set of occurrence relations in the round r, and the j^{th} rows presents a sequence of relations on two clock occurrences.

Example 5 Take the component "commIni" as an example, we enumerate its occurrence relations. Let all occurrence index variables initially be 0 ($m, n, r, i, j := 0$) and the loop control variable k be 1 ($k := 1$). Starting from s_0, we get the transition $tr0 : s_2 \xrightarrow{C_{!R}} s_0 \xrightarrow{C_{?Cmd}} s_1$. From the first line of the Fig. 4.10, we get $n = 1$ (because $m = 0$ and $f_{tr0} : n = m + 1$) and so we get the relation $!R\_0 \prec ?Cmd\_1$. Then the transition goes to s_1. Since $k := 1$, the transition goes into the self-loop. So we get the transition $tr2 : s_0 \xrightarrow{C_{?Cmd}} s_1 \xrightarrow{C_{!Notify}} s_1$. From the third line of Fig. 4.10, we can compute $i = 1$ (because $f_{tr3} : i := i + 1$) and then we get the relation $?Cmd\_1 \prec !Notify\_1$. According to the loop control, we know k increases by 1 ($k + +$), so $k = 2$. Since the condition $k \leq 2$ still is satisfied, the transition goes into the self-loop again. According to the transition $tr3 : s_1 \xrightarrow{C_{!Notify}} s_1 \xrightarrow{C_{!Notify}} s_1$, then we get the relation $!notify\_1 \prec !notify\_2$. Then k increases by 1 ($k + +$), so at this time $k = 3$ that cannot satisfy the condition $k \leq 2$. So the transition goes out of the loop, then we have $tr4 : s_1 \xrightarrow{C_{!Notify}} s_1 \xrightarrow{C_{\tau}} s_2$. According to the table 4.10, we know $r = 1$ (because $f_{tr4} : r = n$). Then the state s_2 is similar as the state s_1. In the end of this inner loop we get the first column of the Fig. 4.12. Remark that the rows corresponding to transitions $tr1$ and $tr5$ from Fig. 4.10 have been eliminated in this process, because the corresponding loops cannot exit immediately. Then by repeating the second round, third round, etc, we can get the relations listed in the second column, the third column of Fig. 4.12, etc., until we reach to the column N.

1^{st} round	2^{nd} round	3^{rd} round	s^{th} round	...	clock relations
$!R\_0 \prec ?Cmd\_1$	$!R\_1 \prec ?Cmd\_2$	$!R\_2 \prec ?Cmd\_3$	$!R\_(s-1) \prec ?Cmd\_s$	...	$C_{!R} \prec C^{\Delta(1)}_{?Cmd}$
$\boxed{?Cmd\_1 \prec !notify\_1}$	$\boxed{?Cmd\_2 \prec !notify\_3}$	$\boxed{?Cmd\_3 \prec !notify\_5}$	$\boxed{?Cmd\_s \prec !notify\_(2s-1)}$	...	$C_{?Cmd} \prec C^{\{2s-1\}}_{!notify}$
$!notify\_1 \prec !notify\_2$	$!notify\_3 \prec !notify\_4$	$!notify\_5 \prec !notify\_6$	$!notify\_(2s-1) \prec !notify\_2s$	...	$C^{\{2s-1\}}_{!notify} \prec C^{\{2s\}}_{!notify}$
$!notify\_2 \prec \tau\_1$	$!notify\_4 \prec \tau\_2$	$!notify\_6 \prec \tau\_3$	$!notify\_2s \prec \tau\_s$	...	$C^{\{2s\}}_{!notify} \prec C_{\tau}$
$\tau\_1 \prec ?ack\_1$	$\tau\_2 \prec ?ack\_3$	$\tau\_3 \prec ?ack\_5$	$\tau\_s \prec ?ack\_(2s-1)$	...	$C_{\tau} \prec C^{\{2s-1\}}_{?ack}$
$?ack\_1 \prec ?ack\_2$	$?ack\_3 \prec ?ack\_4$	$?ack\_5 \prec ?ack\_6$	$?ack\_(2s-1) \prec ?ack\_2s$	...	$C^{\{2s-1\}}_{?ack} \prec C^{\{2s\}}_{?ack}$
$?ack\_2 \prec !R\_1$	$?ack\_4 \prec !R\_2$	$?ack\_6 \prec !R\_3$	$?ack\_2s \prec !R\_s$	...	$C^{\{2s\}}_{?ack} \prec C_{!R}$

Fig. 4.12: Steps 2-3-4: Unfold rounds, generalize, and deduce clock relations

Step 3: Generalize the occurrence relations

In table 4.12, in each line we get a sequence of occurrence relations. To induce the corresponding general relation, we transfer the problem to finding a general formula for a sequence of nature numbers. We could use here standard arithmetic method (e.g. Neville's algorithm [**?**]) that are able to deduce polynomial formulas generating natural number sequences. However, such a general approach would make difficult to estimate the minimum number of unfoldings required for finding the general formula. But in fact, due to our hypothesis on the independence of the loop control counters, the formula we seek here will be linear in the clock indexes, and the length of unfolding may be estimated from the maximum value of the loop indexes. A proof of this property, and a detailed estimation of the bound, is out of the scope of this thesis. The result of generalisation is shown in "column s" in the Fig. 4.12.

Example 6 Let us go on the Fig. 4.11 as an example. Since the loop counter is 2, so we need unfold the relations at most for 3 times. As shown in the second line of the table 4.12, the sequence of occurrence indexes of the clocks $C_{?Cmd}$ and $C_{!Notify}$ are $\{1, 2, 3\}$ and $\{1, 3, 5\}$. According to the Neville's algorithm, we can get the general formulas for the clock $C_{?Cmd}$ as $a_n = n$, and for the clock $C_{!Notify}$ as $a_n = 2n - 1$. So in the second line, the relation of the s round ($\forall s < 0$) is $?Cmd\_s \prec !Notify\_\{2s-1\}$.

Algorithm 2 Unfold occurrence relation table

Input: A symbolic occurrence table with a clock set C with n clocks. $C = \{C_1, C_2, \ldots C_n\}$

Output: enumerate occurrence relations of N rounds in the matrix R[j][r], in which j is the index of rows and r is the index of columns (rounds).

for all C_i **do**
  $Indexof(C_i) := 0$ {initialisation}
end for
set var j, r :=0
var $s := s_0$
set var $C_\alpha :=$ anyone from PreAct(s)
set var $C_\beta :=$ one from PostAct(s) that satisfies a certain guard
set var $s' \leftarrow \{s' | s' \xrightarrow{C_\alpha} s\}$
set var $s'' \leftarrow \{s'' | s \xrightarrow{C_\beta} s''\}$
while $r \leq N$ **do**
  while $C \neq \emptyset$ **do**
    if $s = s_0$ **then**
      $r++; j := 0$
    end if
    for all row in table **do**
      if $tr = s' \xrightarrow{C_\alpha} s \xrightarrow{C_\beta} s''$ **then**
        $Indexof(C_\beta) \leftarrow$ compute by f_{tr}
        $R[j][r] = \alpha\_Indexof(C_\alpha) \prec \beta\_Indexof(C_\beta)$
        $C \leftarrow C - C_\alpha - C_\beta$
        $j++$
        $s' \leftarrow s$; $s \leftarrow s''$; $s'' \leftarrow$ one from PostAct(s) that satisfies a certain guard;
        $C_\alpha := C_\beta$
        $C_\beta := \{C_\beta | s \xrightarrow{C_\beta} s''\}$
      end if
    end for
  end while
  reset C with n clocks $C = \{C_1, C_2, \ldots C_n\}$
end while

Step4: lifting to clock relations

In the last step, we lift the concurrence relations to clock relations, using the clock operators "lift" and "filter" from definitions 6 and 7. This step is straightforward, and the result is shown in the last column of Fig. 4.12.

4.5.5 Generating TS of timed-pNets

A timed-pNets node actually consists of a set of holes (J) with timed specifications (TS_j), synchronous vectors (V_i), and global clocks ($\mathfrak{C}_G$) generated from the synchronous vectors. Therefore, generating the external timed specification for a timed-pNets node (called global timed specification TS_g) boils down to compute the global clock relations from the local timed-specifications of its holes (TS_j) and the coincidence relations deduced from the synchronous vectors (V_i), using the properties on clock relations from section 4.2.2. Formally:

Definition 14 (Global Clock Relation Set) Given a timed-pNet $T\text{-}pNets =< P, A_G, \mathfrak{C}_G, J, \widetilde{A}_J, \widetilde{\mathfrak{C}}_J, \widetilde{\mathcal{R}}_J, \overrightarrow{V} >$ The global time specification of $T\text{-}pNets$ is the pair $< \mathfrak{C}_G, \mathcal{R}_G >$, where $\mathcal{R}_G$ is the Global Clock Relation Set deduced from:
- all local clocks relations $\mathcal{R}_j$ from its holes,
- the (coincidence) relations deduced from all its synchronization vectors,
- symmetry and transitivity of coincidence, transitivity of precedence.

During this logical saturation process, it may happen that contradictory relations are deduced, when 2 clocks would be proved both coincident and precedent, or precedent both ways. This we call a conflict:

Definition 15 (Clock Conflicts) Given a timed specification $< \mathfrak{C}, \mathcal{R} >$:
- two clocks C_α and C_β in $\mathfrak{C}$ are in conflict if either $C_\alpha = C_\beta \wedge (C_\alpha \prec C_\beta \vee C_\beta \prec C_\alpha) \in \mathcal{R}$ or $C_\alpha \prec C_\beta \wedge C_\beta \prec C_\alpha \in \mathcal{R}$
- the Global Clock Conflict Set of a timed-pNet is the set of pairs of clocks in conflict in its Global Clock Relation Set.

Example 7 Let us take the Fig. 4.8 as an example. From the user specification in example 3 (page 72), we know the clock relations of these holes are:

- $\mathcal{R}_{\{CommIni\}} = \{C_{?Cmd} \prec C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}}, \quad C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{!notify}^{\{2s\}_{s\in\mathbb{N}}}, \quad C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}}, \quad C_{!notify}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}}, \quad C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}}, \quad C_{?ack}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{!R} \prec C_{?cmd}^{\Delta(1)}\}$

- $\mathcal{R}_{\{ChannelNtf[m]\}} = \{C_{c.?notify_{[m]}} \prec C_{c.!notify_{[m]}} \prec C_{c.?notify_{[m]}^{\Delta(1)}}\}$,

- $\mathcal{R}_{\{ChannelAck[m]\}} = \{C_{c.?ack_{[m]}} \prec C_{c.!ack_{[m]}} \prec C_{c.?ack_{[m]}}^{\Delta(1)}\}$,

- $\mathcal{R}_{\{CommRes[m]\}} = \{C_{?notify_{[m]}} \prec C_{!ack_{[m]}} \prec C_{?notify_{[m]}}^{\Delta(1)}\}$.

Besides, we derive the clock relations from the synchronous communications defined by synchronous vectors as:

- $\mathcal{R}_{V_1} = \{C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} = C_{c.?notify_{[1]}} = C_{notify_{g1_{[1]}}}\}$,

- $\mathcal{R}_{V_2} = \{C_{c.!notify_{[1]}} = C_{?notify_{[1]}} = C_{notify_{g2_{[1]}}}\}$,

- $\mathcal{R}_{V_3} = \{C_{!ack_{[1]}} = C_{c.?ack_{[1]}} = C_{ack_{g3_{[1]}}}\}$,

- $\mathcal{R}_{V_4} = \{C_{c.!ack_{[1]}} = C_{?ack}^{\{2s-1\}} = C_{ack_{g4_{[1]}}}\}$,

- $\mathcal{R}_{V_5} = \{C_{!notify}^{\{2s\}_{s\in\mathbb{N}}} = C_{c.?notify_{[2]}} = C_{notify_{g1_{[2]}}}\}$,

- $\mathcal{R}_{V_6} = \{C_{c.!notify_{[2]}} = C_{?notify_{[2]}} = C_{notify_{g2_{[2]}}}\}$,

- $\mathcal{R}_{V_7} = \{C_{!ack_{[2]}} = C_{c.?ack_{[2]}} = C_{ack_{g3_{[2]}}}\}$,

- $\mathcal{R}_{V_8} = \{C_{c.!ack_{[2]}} = C_{?ack}^{\{2s\}} = C_{ack_{g4_{[2]}}}\}$,

- $\mathcal{R}_{V_9} = \{C_{?Cmd} = C_{?Cmd_{g5}}\}$,

- $\mathcal{R}_{V_{10}} = \{C_{!R} = C_{!R_{g6}}\}$.

Take the relation between the global clocks $C_{notify_{g1_{[1]}}}$ and $C_{notify_{g2_{[1]}}}$ as an example. They are generated by the synchronous vectors V_1 and V_2. From the relations of hole $ChannelNtf_{[1]}$ and the relations of these two vectors, we can get the formula $C_{notify_{g1_{[1]}}} =_{(\mathcal{R}_{V_1})} C_{c.?notify_{[1]}} \prec_{\mathcal{R}`_{ChannelNtf[1]})} C_{c.!notify_{[1]}} =_{(\mathcal{R}_{V_2})} C_{notify_{g2_{[1]}}}$. In the end, we conclude $C_{notify_{g1_{[1]}}} \prec C_{notify_{g2_{[1]}}}$.

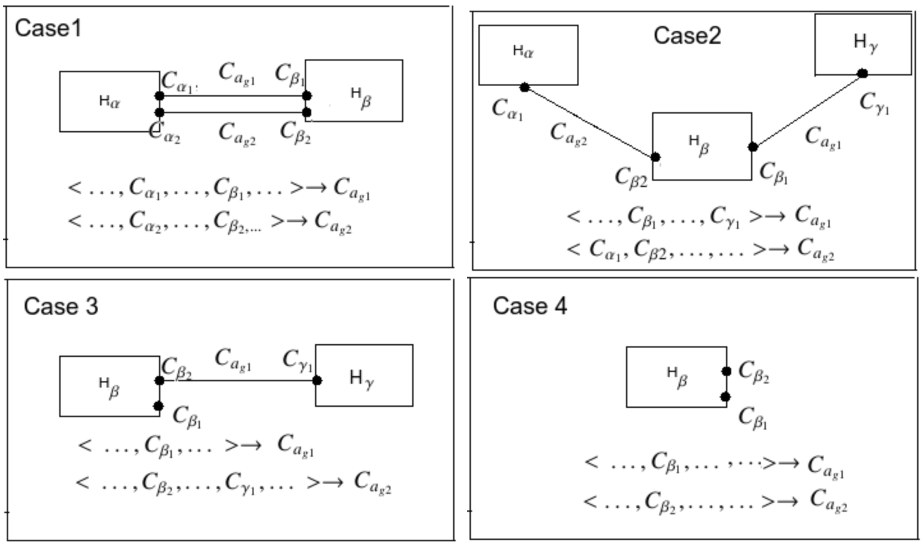

Fig. 4.13: The 4 cases of theorem 1

The formal definition above is not very practical. The following theorem defines the case analysis procedure, and states its correctness (all relations computed are correct). The next theorem will prove its completeness. In one particular case, this case analysis procedure may detect a local conflict between two global actions, more precisely between two synchronization vectors representing communication between the same 2 holes. In this case, we shall signal the conflict, but produce no relations between these actions. Other types of conflicts could be created by configurations involving more than 2 holes. These cannot be detected at the level of this case-analysis procedure; a full conflict detection procedure is out of the scope of this thesis.

Theorem 1 (Global clock relation analysis) Given a timed-pNet $T\text{-}pNets = < P, A_G, \mathfrak{C}_G, J, \widetilde{A}_J, \widetilde{\mathfrak{C}}_J, \widetilde{\mathcal{R}}_J, \overrightarrow{V} >$. Let H_α, H_β, H_γ be three holes of $T\text{-}pNets$ and $C_{H_\alpha}, C_{H_\beta}, C_{H_\gamma} \subset \widetilde{\mathfrak{C}}_J$ be the sets of clocks of holes H_α, H_β and H_γ. Let the clocks $C_{\alpha_1}, C_{\alpha_2} \in C_{H_\alpha}$, the clocks $C_{\beta_1}, C_{\beta_2} \in C_{H_\beta}$, the clock $C_{\gamma_1} \in C_{H_\gamma}$, with $C_{H_\alpha} \bigcap C_{H_\beta} \bigcap C_{H_\gamma} = \emptyset$). For each pair of global clocks $C_{a_{g1}}$ and $C_{a_{g2}}$, we enumerate the pairs of synchronization vectors able to generate them, and match them with the following cases (note that both pairs $(C_{a_{g1}}, C_{a_{g2}})$ and $(C_{a_{g2}}, C_{a_{g1}})$ will be enumerated, so we do not consider symmetric conditions in the cases below). Each match may add a clock relation in the Global

Clock Relation Set $\mathcal{R}$:

- (**Case1:**) If the global clocks $C_{a_{g1}}$ and $C_{a_{g2}}$ are generated from synchronous vectors
  $< \ldots, C_{\alpha_1}, \ldots, C_{\beta_1}, \ldots > \to C_{a_{g1}}$ and
  $< \ldots, C_{\alpha_2}, \ldots, C_{\beta_2}, \ldots > \to C_{a_{g2}}$,
  which are related to two holes C_{H_α} and C_{H_β} as shown in Fig. 4.13(1), then
  - if $C_{\alpha_1} = C_{\alpha_2} \wedge C_{\beta_1} = C_{\beta_2}$ then $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$.
  - if $C_{\alpha_1} \prec C_{\alpha_2} \wedge C_{\beta_1} \prec C_{\beta_2}$ then $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$.
  - if $C_{\alpha_1} = C_{\alpha_2} \wedge C_{\beta_1} \prec C_{\beta_2}$ or if $C_{\alpha_1} = C_{\alpha_2} \wedge C_{\beta_2} \prec C_{\beta_1}$ then conflict found.
- (**Case2:**) If the global clock $C_{a_{g1}}$ and $C_{a_{g2}}$ are generated from the synchronous vectors
  $< \ldots, C_{\beta_1}, \ldots, C_{\gamma_1} > \to C_{a_{g1}}$ and
  $< C_{\alpha_1}, C_{\beta_2}, \ldots, \ldots > \to C_{a_{g2}}$, which are related to three holes C_{H_α}, C_{H_β} and C_{H_γ} as shown in Fig. 4.13, then
  - if $C_{\beta_1} = C_{\beta_2}$ then $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$,
  - if $C_{\beta_1} \prec C_{\beta_2}$ then $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$.
- (**Case3:**) If the global clock $C_{a_{g1}}$ and $C_{a_{g2}}$ are generated from the synchronous vectors
  $< \ldots, C_{\beta_1}, \ldots > \to C_{a_{g1}}$ and
  $< \ldots, C_{\beta_2}, \ldots, C_{\gamma_1}, \ldots > \to C_{a_{g2}}$;; as shown in Fig. 4.13(3). then
  - if $C_{\beta_1} = C_{\beta_2}$ then $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$,
  - if $C_{\beta_1} \prec C_{\beta_2}$ then $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$.
- (**Case4:**) If the global clock $C_{a_{g1}}$ and $C_{a_{g2}}$ are generated from the synchronous vectors $< \ldots, C_{\beta_1}, \ldots > \to C_{a_{g1}}$ and $< \ldots, C_{\beta_2}, \ldots, \ldots > \to C_{a_{g2}}$ as shown in Fig. 4.13(4). then
  - if $C_{\beta_1} = C_{\beta_2}$ then $(C_{a_{g1}} = C_{a_{g2}}) \in \mathcal{R}$,

– if $C_{\beta_1} \prec C_{\beta_2}$ then $(C_{a_{g1}} \prec C_{a_{g2}}) \in \mathcal{R}$.

- (**Otherwise**) In any other case, this pair of clocks is **NOT** directly related in $\mathcal{R}$

Proof. For each of the cases, we prove that the deduced relation is indeed correct with respect to definition 14.

- **Case1:** From the two synchronous vectors $< \ldots, C_{\alpha_1}, \ldots, C_{\beta_1}, \ldots > \rightarrow C_{a_{g1}}$,
  $< \ldots, C_{\alpha_2}, \ldots, C_{\beta_2}, \ldots > \rightarrow C_{a_{g2}}$,
  we know that $C_{\alpha_1} = C_{\beta_1} = C_{a_{g1}}$ and $C_{\alpha_2} = C_{\beta_2} = C_{a_{g2}}$. (1) If $C_{\alpha_1} = C_{\alpha_2} \wedge C_{\beta_1} = C_{\beta_2}$, according to the transitivity property of "=", we get the relation $C_{a_{g1}} = C_{a_{g2}}$.
  (2) If $C_{\alpha_1} \prec C_{\alpha_2} \wedge C_{\beta_1} \prec C_{\beta_2}$, then we have $C_{a_{g1}} = C_{\alpha_1} \prec C_{\alpha_2} = C_{a_{g2}}$. So using substitutivity of = w.r.t. $\prec$, we get the relation $C_{a_{g1}} \prec C_{a_{g2}}$.

- **Case2:** From the two synchronous vectors $< \ldots, C_{\beta_1}, \ldots, C_{\gamma_1} > \rightarrow C_{a_{g1}}$ and $< C_{\alpha_1}, C_{\beta_2}, \ldots, \ldots > \rightarrow C_{a_{g2}}$,
  we know that $C_{\beta_1} = C_{\gamma_1} = C_{a_{g1}}$ and $C_{\alpha_1} = C_{\beta_2} = C_{a_{g2}}$. (1) If $C_{\beta_1} = C_{\beta_2}$, then according to the transitivity property of "=", we know that $C_{a_{g1}} = C_{a_{g2}}$. (2) If $C_{\beta_1} \prec C_{\beta_2}$, since $C_{a_{g1}} = C_{\beta_1} \prec C_{\beta_2} = C_{a_{g2}}$, then we have the relation $C_{a_{g1}} \prec C_{a_{g2}}$.

- **Case3 and Case4:** The proofs are similar to Case2.

□

Example 8 Let us take again the Fig. 4.8 as an example to compute the clock relation between $C_{notify_{g2_{[1]}}}$ and $C_{ack_{g3_{[1]}}}$. We know the two global actions are generated by the vectors V_2: $< \ldots, C_{c.!notify_{[1]}}, \ldots, C_{?notify_{[1]}}, \ldots > \rightarrow C_{notify_{g2_{[1]}}}$ and $V_3 : < \ldots, C_{!ack_{[1]}}, \ldots, C_{c.?ack_{[1]}}, \ldots > \rightarrow C_{ack_{g3_{[1]}}}$. So we are in the case 2). Moreover, from $TS_{\{CommRes_{[1]}\}}$ we know that $C_{?notify_{[1]}} \prec C_{!ack_{[1]}}$. Therefore, we conclude $C_{notify_{g2_{[1]}}} \prec C_{ack_{g3_{[1]}}}$.

Theorem 2 (Completeness) There exist four and only four combinations of synchronous vectors, as listed in Theorem 1, for deducing a relation between a pair of global clocks.

Proof. From the timed-pNets definition, we know that there are two ways to build a global clock: binary communication and visibility. So there are 3 combinations:

(1) both global clocks are generated by binary communication

(2) one global clock is generated by binary communication and another one is generated by visibility

(3) both global clocks are generated by visibility

Now we analyze the three situations one by one. Given a timed-pNet $T\text{-}pNet = < P, A_G, \mathfrak{C}_G, J, \widetilde{A}_J, \widetilde{\mathfrak{C}}_J, \widetilde{\mathcal{R}}_J, \overrightarrow{V} >$.
(1) Let $< \ldots, C_\alpha, \ldots, C_\beta > \rightarrow C_{g1}$ and $< \ldots, C_\gamma, \ldots, C_\eta > \rightarrow C_{g2}$ (C_α, C_β, C_γ,$C_\eta \in \widetilde{\mathfrak{C}}_J$) be two synchronous vectors generating the global clocks C_{g1} and C_{g2}. Obviously the four local clocks C_α, C_β, C_γ,C_η cannot be in one hole since the synchronous vectors build binary communications between holes. If the four local clocks come from two holes, then the possible combinations are C_α and C_γ are in one hole, the other two are in another hole. Or C_α and C_η are in one hole, the other two are in another hole. Case 1 of the theorem 1 covers the both situations. If the four local clocks come from three holes, then any two local clocks that come from different synchronous vectors must be in one hole, and the rest two local clocks are in other two different holes. For example, C_α and C_γ are in one hole, the other two are in other two holes separately. Case 2 of the theorem 1 covers the situation. Furthermore, the four local clocks cannot be in 4 holes (or more than 4 holes). Otherwise there is no local clock relations in $\widetilde{\mathcal{R}}_J$ can be used to deduce global clock relations. Therefore, no direct clock relation can be built between C_{g1} and C_{g2}.
(2) Let $< \ldots, C_\alpha, \ldots, C_\beta > \rightarrow C_{g1}$ and $< \ldots, C_\gamma, \ldots, > \rightarrow C_{g2}$ (C_α, C_β, $C_\gamma \in \widetilde{\mathfrak{C}}_J$) be two synchronous vectors to generate the global clocks C_{g1} and C_{g2}. Similar to the proof in the previous situation, the three local clocks cannot be in one hole and cannot be in 3 holes or more. So the only possible

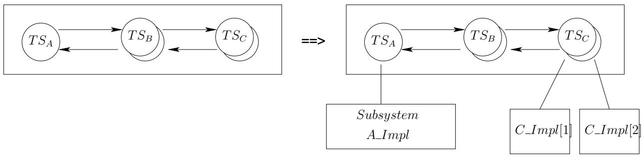

Fig. 4.14: Partial instantiation of a Timed-pNets subsystem

combination is that C_γ is in the same hole that one of the others. Case 3 of the theorem 1 covers the situation.
(3) Let $< \ldots, C_\alpha, \ldots, > \rightarrow C_{g1}$ and $< \ldots, C_\gamma, \ldots, > \rightarrow C_{g2}$ (C_α, $C_\gamma \in \widetilde{\mathfrak{C}}_J$) be two synchronous vectors to generate the global clocks C_{g1} and C_{g2}. The two local clocks cannot be in 2 different holes. Otherwise there is no local relation can be find between them. So the only possible situation is the two local clocks are in the same hole. Case 3 of the theorem 1 covers the situation.
In conclusion, if the relation of two global clocks C_{g1}, $C_{g2} \in \mathfrak{C}_G$ can be deduced by the local clock relations from $\widetilde{\mathcal{R}}_J$, then the four cases listed in the theorem 1 cover all possible combinations of synchronous vectors. $\square$

4.6 Compatibility

When assembling timed-pNets, the architect has to ensure that the timed-pLTS that will be plugged into a hole indeed matches the hole Timed Specification. The ultimate goal is to provide a refinement-based approach: timed properties proved on an open (abstract) timed-pNet system will be preserved by refinement of Timed Specifications. One of the basic tool for building such refinement is to ensure the compatibility of a subsystem with the enclosing holes before composing the system. E.g. in Fig. 4.14, the Timed Specification (TS) of the subsystem "*A_Impl*" must be compatible with TS_A, and each of the "*C_Impl*" must be compatible (individually) with TS_C.

Our notion of compatibility will be based on the inclusion relations between the Clock relation sets. Before giving its formal definition, we introduce the concepts of "Saturated relation set" and "Relation set inclusion".

Definition 16 (Saturated Relation Set) Let $TS =< C, R >$ be a timed specification with a set of clocks C and a set of relations R. The saturated relation set (denoted as R^+) is the clock relation set R augmented by all relations possibly deduced from R, by transitivity of precedence and reflexivity, symmetry, and transitivity of coincidence.

For example, if $R = \{c_1 \prec c_2 \prec c_3\}$ $(c_1, c_2, c_3 \in C)$, then according to the transitivity property of the relation $\prec$, we can get a new relation set $R^+ = \{c_1 \prec c_2 \prec c_3, c_1 \prec c_3, c_1 = c_1, c_2 = c_2, ...\}$

Definition 17 (Inclusion of time specifications) Given two timed specifications $TS_1 =< C_1, R_1 >$ and $TS_2 =< C_2, R_2 >$. Let R_1^+ (resp. R_2^+) be a set of saturated relations in the TS_1 (resp. TS_2). We say TS_2 includes TS_1 (denoted as $TS_1 \ll TS_2$) if and only if $C_1 \subseteq C_2 \wedge$ $R_1 \subseteq R_2^+$.

According to the definition, $TS_1 \ll TS_2$ means that the relation existing in the timed specification TS_1 must exist in TS_2 or can be deduced from the relations in TS_2. For example, assume $TS_1 = \{c_1 \prec c_3\}$, $TS_2 = \{c_1 \prec c_2 \prec c_3\}$. According to the transitivity property of the "$\prec$", we can get the the saturated relation set of the TS_2 as $R^+ = \{c_1 \prec c_2 \prec c_3, c_1 \prec c_3, c_1 = c_1, c_2 = c_2, ...\}$. Since the relation in TS_1 can be deduced from the relations in TS_2, we say TS_2 includes TS_1 ($TS_1 \ll TS_2$).

Lemma 1 If $TS_1 =< C_1, R_1 >$ and $TS_2 =< C_2, R_2 >$ are two timed specifications, then $TS_1 \ll TS_2 \implies R_1^+ \subseteq R_2^+$

Proof. Taken any two relation r_1, $r_1' \in R_1$. Let $r_1^+ \in R_1^+$ be the relation deduced from the two relations r_1, r_1' in terms of the property P proposed in section 4.2. Assume $r_1^+ \notin R_2^+$. Since $TS_1 \ll TS_2$, from the definition of inclusion we know $R_1 \subseteq R_2^+$. Furthermore, we know $r_1, r_1' \in R_2^+$. So in the set R_2^+ we can get the relation r_1^+ by using the same property P. So we have $r_1^+ \in R_2^+$ that is contradict with our assumption. Therefore, we have $r_1^+ \in R_2^+$. Moreover, because $r_1^+ \in R_1^+$, so $R_1^+ \subseteq R_2^+$. □

Definition 18 (Compatibility) Let TS be the timed specification of a timed-pNets hole H, and TS' be the timed specification of an implementation H_Impl. We say H_Impl is compatible with H, denoted by H_Impl $\sqsubseteq H$ if and only if $TS \ll TS'$.

Theorem 3 Let TS be the timed specification of hole H. Let TS'_1 (resp. TS'_2) be the timed specification of an implementation H_Impl_1 (resp. H_Impl_2). If $H\_Impl_1 \sqsubseteq H$ and $TS'_1 \ll TS'_2$, then $H\_Impl_2 \sqsubseteq H$.

Proof. Assume $TS'_1 =< C'_1, R'_1 >$, $TS'_2 =< C'_2, R'_2 >$ and $TS =< C, R >$. Let R'^{+}_1 (resp. R'^{+}_2, R^{+}) be the saturated relation from TS'_1 (resp. TS'_2, TS). Since $H\_Impl_1 \sqsubseteq H$, according to the compatibility definition, we have $TS \ll TS'_1$. Furthermore, according to the Inclusion definition, we have $R \subseteq R'^{+}_1$. Moreover, because we know that $TS'_1 \ll TS'_2$, according to the Lemma 1, we have $R'^{+}_1 \subseteq R'^{+}_2$. According to the set theory, we know that $R \subseteq R'^{+}_2$. Finally, according to the Inclusion and compatibility definition, we get $H\_Impl_2 \sqsubseteq H$. □

4.7 Assembling multi-layer timed-pNets system

After generating a timed specification for a timed-pNets node, we can use the generated timed specification to prove that it would be compatible with the specification of a hole of a higher-level timed-pNet node. This way, a layered tree structure can be built as shown in the Fig. 4.15. In this structure, each layer uses an abstraction of its lower layer. The clocks in the lower layer (at level N) are transparent to its abstract layer (at level N+1) in which only holes with its timed specification (TS_j), synchronous vectors (V_i) and global clocks (C_g) can be seen.

As we have already mentioned, this construction can be done in a very flexible way either bottom-up or top-down. The result timed-pNet system can be open (if it still contains some unfilled holes at the leaves), or closed if all holes are filled with timed-pNets and timed-pLTS.

Example 9 We now have all elements required for checking the compatibility of our timed-pLTSs with the holes of the upper layer pNet. Let us look at "CommIni" as an example:

- the relation set of the hole "CommIni" for open timed-pNets is $\mathcal{R}_{CommIni} =$

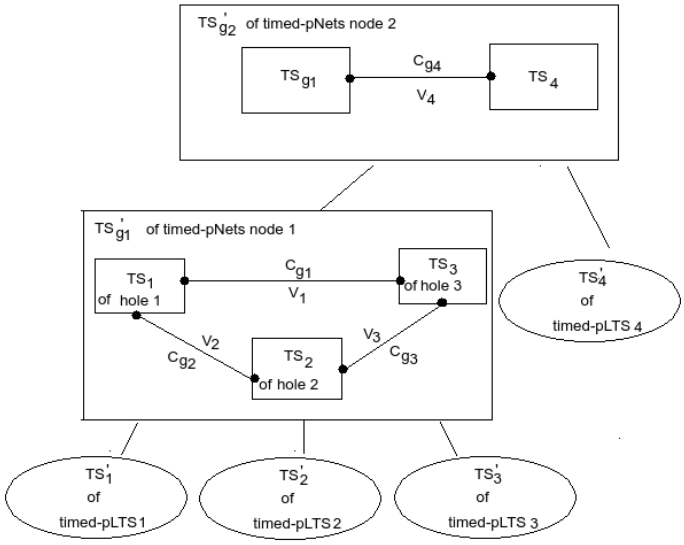

Fig. 4.15: Layered Structure

$\{C_{?Cmd} \prec C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}},\ C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{!notify}^{\{2s\}_{s\in\mathbb{N}}},\ C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}},\ C_{!notify}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}},\ C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}},\ C_{?ack}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{!R} \prec C_{?cmd}^{\Delta(1)}\}$,

- the relation set of the "CommIni" timed-pLTS component from Fig. 4.12 as $\mathcal{R'}_{Commini} = \{C_{?Cmd} \prec C_{!notify}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{!notify}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s-1\}_{s\in\mathbb{N}}} \prec C_{?ack}^{\{2s\}_{s\in\mathbb{N}}} \prec C_{\tau} \prec C_{!R} \prec C_{?cmd}^{\Delta(1)}\}$.

Since we can easily get $\mathcal{R}_{CommIni} \subseteq \mathcal{R'}_{Commini}$, according to Inclusion definition we have $TS_{\{CommIni\}} \ll TS'_{Commini}$. Therefore, from the compatibility definition, we know that the "CommIni" timed-pLTS is compatible with the hole "CommIni".

The validations that have been defined in our paper, namely the compatibility of hole composition, and the conflict detection between timed-pNets synchronization vectors, ensure some specific validity properties of the global Time Specification of the system, as defined by Definition 14. However, this does not mean that there cannot be more complex conflicts in the interaction between more than 2 holes of a timed-pNets, or more specific timed properties that can be computed from refined implementations of some sub-nets. In the next section, we show how to use simulation with the TimeSquare tool, to address such cases.

4.8 Simulation

In this section we explain how to use TimeSquare [41] to detect complex conflicts of timed-pNets. Two inputs are required by TimeSquare (see the Fig. 5.7). One is an open timed-pNets system. Another is a set of refined implementations. If a closed timed-pNets composed by those refined implementations has no conflict, we say the closed timed-pNets is safe. Otherwise, the TimeSquare reports violations, which means that conflicts exist in the closed timed-pNets system. Before running simulations, the two inputs are translated into timed specifications that are acceptable format for TimeSquare. The way of generating timed specification is described in section 4.5.

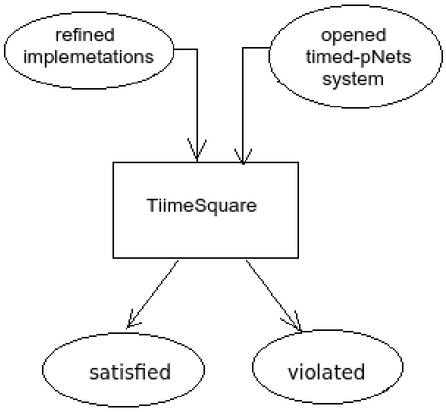

Fig. 4.16: Property Checking by TimeSquare

4.8.1 Simulation 1:

- We take the system shown in the Fig. 4.8 as an example. We first build an open timed-pNet node with the timed specifications of holes (TS: $TS_{\{CommIni\}}$, $TS_{\{ChannelNtf[m]\}}$, $TS_{\{ChannelAck[m]\}}$, $TS_{\{CommRes[m]\}}$) and synchronous vectors (V_i), by which we can generate global clock relations (we call it an abstract specification). From section 4.5.5, we can get the abstract specification $TS_g =< C_g, R_g >$ with $R_g =$

$\{C_{?Cmd_{g5}} \prec C_{notify_{g1_{[m]}}} \prec C_{notify_{g2_{[m]}}} \prec C_{ack_{g3_{[m]}}} \prec C_{ack_{g4_{[m]}}} \prec C_{!R_{g6}}; C_{notify_{g1_{[1]}}} \prec C_{notify_{g1_{[2]}}}; C_{ack_{g4_{[1]}}} \prec C_{ack_{g4_{[2]}}}\}$. Then we import the timed specifications of the refined implementations of those holes (TS': $TS'_{\{CommIni\}}$, $TS'_{\{ChannelNtf[m]\}}$, $TS'_{\{ChannelAck[m]\}}$, $TS'_{\{CommRes[m]\}}$) to replace TS. The timed-pNets node that composed by these refined implementations is called closed timed-pNets node. And its global clock relations is named concrete specification TS'_g.

- *Result of Simulation 1:* The Fig. 4.17 illustrates the concrete specification TS'_g. In this figure, each line represents a clock and the red arrows represent the precedence relations. For simplification, here we represent two cycles of simulation. From the figure we can see that the abstract specification TS_g is satisfied by the refined concrete system since we have $TS_g \ll TS'_g$.

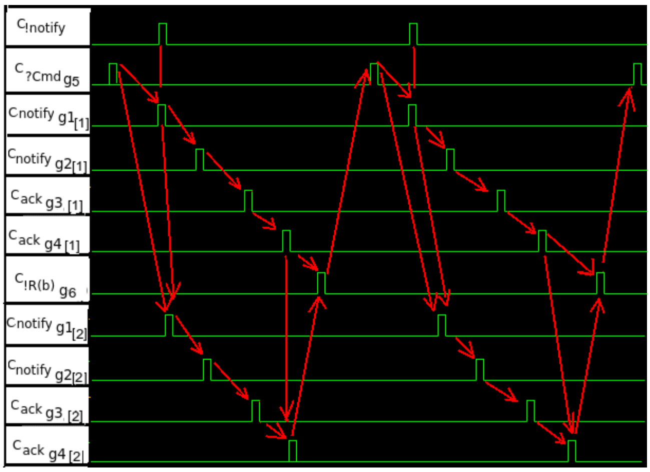

Fig. 4.17: system's specification checking

4.8.2 Simulation 2:

- In this simulation, we choose $TS'_{\{UpdatedCommIni\}} = \{C_{?Cmd} \prec C^{\{2s-1\}}_{!Notify} \prec C^{\{2s-1\}}_{?Ack} \prec C^{\{2s\}}_{!Notify} \prec C^{\{2s\}}_{?Ack} \prec C_{!R} \prec C^{\Delta(1)}_{?Cmd}\}$, $TS'_{\{UpdatedCommRes[m]\}} = \{C_{?NotifyInfo_{[m]}} \prec C_{ExchangeInfo_{[m]}} \prec C_{!Ack_{[m]}}\}$ and we add a synchronous vector between hole $CommRes[1]$ and $CommRes[2]$ to get a new relation $R_{V_{new}} = \{C_{ExchangeInfo_{[1]}} = C_{ExchangeInfo_{[2]}} = C_{ExchangeInfo_{g11}}\}$. Obviously, the updated implementation of hole $CommIni$ is compatible with the abstract timed specification of this hole $TS_{\{CommIni\}}$ since we have $TS_{\{CommIni\}} \ll TS'_{\{UpdatedCommIni\}}$. And the same to the other two holes $CommRes[m]$ since we have $TS_{\{CommRes[m]\}} \ll TS'_{\{UpdatedCommRes[m]\}}$.

- *Result of simulation 2:* By simulation, we found violations as shown in Fig.4.18.

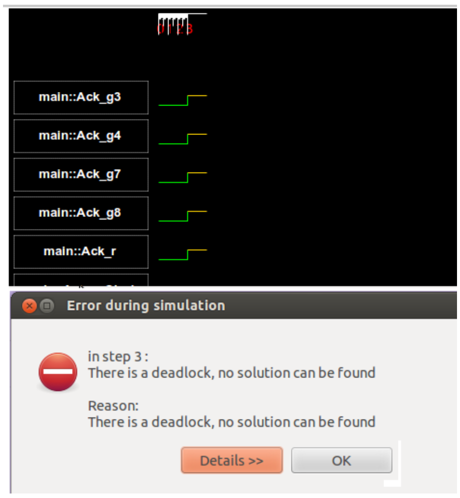

Fig. 4.18: Conflict Detected

- *Analyzing the result:* By analyzing our updated closed timed-pNets, we found the conflict is caused by a cycle represented in the Fig.4.19. In this Figure, according to the theorem 1, we can get the set of global relations as $\{C_{Notify_{g1_{[2]}}} \prec C_{Notify_{g2_{[2]}}} \prec C_{ExchangeInfo_{g7}} \prec C_{Ack_{g3_{[1]}}} \prec C_{Ack_{g4_{[1]}}}\}$. Obviously, relation $\{C_{Notify_{g1_{[2]}}} \prec C_{Ack_{g4_{[1]}}}\}$ is hold in terms of the transitivity property of precedence relations. However, by using the theorem 1 again, from the $TS'_{\{UpdatedCommIni\}}$ we can get the relation $\{C_{Ack_{g4_{[1]}}} \prec C_{Notify_{g1_{[2]}}}\}$ which is contradict with the relation $\{C_{Notify_{g1_{[2]}}} \prec C_{Ack_{g4_{[1]}}}\}$. To fix the conflict, we need to find another implementation that still compatible with these holes but without making conflicts. For our example, we can just simply change the $TS'_{\{UpdatedCommIni\}}$ to $TS'_{\{FixedCommIni\}} = \{C_{?Cmd} \prec C^{\{2s-1\}}_{!Notify} \prec C^{\{2s\}}_{!Notify} \prec C^{\{2s-1\}}_{?Ack} \prec C^{\{2s\}}_{?Ack} \prec C_{!R} \prec C^{\Delta(1)}_{?Cmd}\}$. And in the end, by simulation, no conflict exists any more.

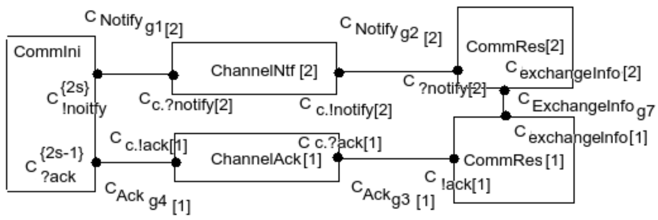

Fig. 4.19: system's specification checking

4.9 Conclusion

This chapter proposed a flexible time-related behavioral semantic model (called Timed-pNets) for modeling communication behavior of distributed systems. We specify a system with several components and communications between them. We are able to build a hierarchical tree structure for composing complicated component-based systems. The refinement and compatibility are considered in the chapter. An concrete example is given to represent how to build a hierarchical specification and how to refine the system. In the end, we use TimeSquare to check the compatibility of the refined system.

Three advantages are implied in our model: first, by introducing logical clock relations, timed-pNets model is able to specify the system's time-related communication behavior constrains without relying on physical common clock; second, by using timed specifications, our model is easy to be composed and has the capability of building a hierarchical structure; last but not least, our model can flexible model heterogeneous communication including synchronous and asynchronous communications by introducing channel LTS. We believe that the timed-pNets model is helpful for analyzing the time-related behaviors for distributed systems including cyber physical systems.

After checking the system compatibility, another interesting point is to check system's physical time constrains such as deadline property that expresses whether system communications can be successfully finished before a certain deadline. To check this, we shall choose a reference clock and specify the delay constrains in terms of the reference clock. In this chapter, even though we define delay variables for actions, we do not provide a way to specify delay constrains here. In the next chapter we will investigate the delay variables of timed-pNets model and check system time properties.

Chapter 5 Delay in Timed-pNets

In this chapter, we discuss the delay variables in timed-pNets. Since this model does not rely on common physical clocks, the delays from different subnets are uncomparable, which brings the difficulty of computing delays of the clocks in the upper layer. We solve the issue by introducing the concept of reference clocks and virtual timestamps so that delays can be calculated in terms of a reference clock specified by the users. Moreover, we define time constraint conflicts and investigate time properties like latency property.

5.1 Context and problematic

In the previous chapter, timed-pNets have been proposed to specify communication behaviours of heterogeneous distributed systems. This model is able to specify logical time constrains such as "action α must happen before action β" or "action α and action β must finish at the same time", etc. However, other requirements like "action α must occur $5ms$ later than action β" is difficult to express because our model lacks of common physical clocks. To solve the issue, we introduce the concept of reference clocks and virtual timestamps. A reference clock can be either chronometric or logical. In our daily life, an event is often expressed relative to another one, that is used as a reference. For example, "action α occurs twice as often as action β", or "action α must occur after action β occurs 5 times", or furthermore, "after action γ occurs, action α must occur after 5 occurrences of action β". For all these cases, if one action occurs more often, the others are impacted. This is the main idea of using reference clocks. In this context, physical time is a particular case of logical time where the time generated by a physical clock is taken as a reference. In CCSL, a time library predefines a clock type ($IdealClock$) and a clock (*idealClk*) whose type is $IdealClock$. *idealClk* is a dense chronometric clock with the second as time unit. This clock is assumed to reflect the evolutions of physical time. Based on this *idealClk*, for example, a reference clock with the period 1 ms (for milliseconds) can be defined as $RefCLK$ = *idealClk discretizedBy* 0.001. In our model, a reference can be defined by user like in CCSL or can be anyone chosen from the logical clock set of this model. No matter by which way, the link between the logical clock set and the reference clock should be clear.

In timed-pNets, delays specify the distances between two timed-actions. Before introducing a reference clock, actually the delays of timed-actions in different nodes are uncomparable. By introducing a reference clock and assigning virtual timestamps to those timed-actions, we can manage to compare those delays and compute them with mathematical operators (e.g. "+,

-").

The concept of virtual time for distributed system was brought into prominence by Lamport in 1978 [58]. In Lamport, virtual time is identified by the succession of events (and therefore is discrete). It does not "flow" by its own means like real time whose passage can not be escaped or influenced. The virtual timestamps in this thesis are little bit different than it. We define two dimension values for each timestamp: one represents the time when a timed-action occurs in terms of a reference clock, another represents the order of the occurrences of a timed-action. These virtual timestamps are not fixed in the sense that they can be reassigned in terms of the changes of system's timed specifications.

The delay of a timed-action describes the time that must elapse before the action can be executed. A delay bound constrains the minimal and maximal time delay the timed-action can accept. In order to keep the hierarchical structure of timed-pNets, all clocks (including the generated clocks in upper layer) have the same schema in the sense that they equip with delays and delay bounds. In this chapter, we propose a way to calculate the delays of global logical clocks and deduce the delay bounds of them from subnets.

In the end, we use TimeSquare to check correctness and latency properties. A latency property checks the minimal (or maximal) distance of two clocks in the sense that at least (at most) how much it takes for an occurrence of a clock to occur after the corresponding occurrence of another clock. The property is usually used to check if an action can occur during an expected time. For example, after sending a request to a system, a user is able to know if the system can give a response in time.

The rest of this chapter is organized as follows. In section 5.2 we introduce virtual timestamps. Section 5.3 represents the definition of time constraint conflicts. Then in section 5.4 we propose and prove a theorem allowing to compute delays and delay bounds of global logical clocks. Time properties and simulations are illustrated in section 5.5. In the end we give a conclusion of this chapter.

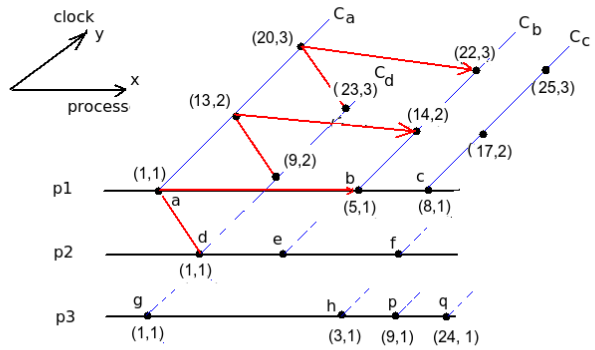

Fig. 5.1: Time Diagram

5.2 Virtual TimeStamps

We define a virtual timestamp as a pair of natural numbers: one represents when a timed-action occurs in terms of a reference clock (X-axis), another represents the order of the occurrences of a timed-action (Y-axis). Fig.5.1 shows us an example in which the timed-actions are assigned with virtual timestamps. In the figure, the processes are presented as solid black lines. The sequence of timed-actions executed in these processes are presented as solid black points on these black lines (X-axis). The actions in each process are totally ordered. The communications between processes are represented by clock relations. For example, in the Fig. 5.1, the clock C_a and clock C_d are coincident. We use a sequence of red lines to represent the coincidence relation of two clocks. Similarly, we use a sequence of red arrows to represent the precedence relations (e.g. $C_a \prec C_b$). We define the virtual timestamps and their assignment rules as follows.

Definition 19 (Virtual Timestamps) A virtual timestamp (denoted as $T(\alpha\_i)$) of a timed-action occurrence α_i is a pair of natural numbers (x_{α_i}, i) $(x_{\alpha\_i} \in \mathbb{N}, i \in \mathbb{N})$.

Definition 20 (Virtual Timestamp Assignment Rules) Let $T(\alpha\_i) \triangleq (x_{\alpha\_i}, i)$ be the virtual timestamp of the occurrence α_i of the clock C_α $(\alpha \in \mathcal{L}_{\mathcal{A},\mathcal{T},\mathcal{P}})$, and $T(\beta\_i) \triangleq (x_{\beta\_i}, i)$ be the virtual timestamp of the occurrence β_i of the clock C_β $(\beta \in \mathcal{L}_{\mathcal{A},\mathcal{T},\mathcal{P}})$. Then we have:

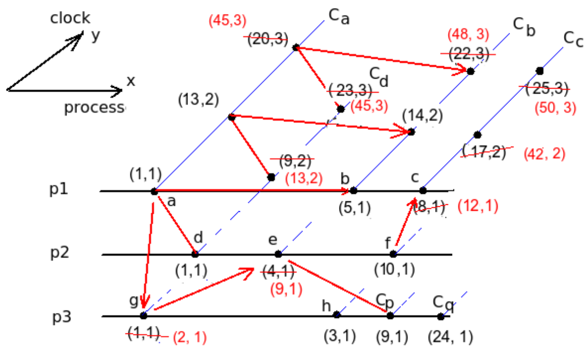

Fig. 5.2: Updated Time Diagram

- $C_\alpha = C_\beta \Rightarrow \forall i, x_{\alpha\_i} = x_{\beta\_i} := max(x_{\beta\_i}, x_{\alpha\_i})$

- $C_\alpha \prec C_\beta \Rightarrow \forall i, x_{\alpha\_i} < x_{\beta\_i}$ and $x_{\beta\_i} := max(x_{\alpha\_i}, x_{\beta\_i}) + t_{\beta_i}$ (the variable t_{β_i} presents the delay time from the occurrence α_i to β_i in terms of the reference clock that a user chose. $t_{\beta_i} \geq 1, t_{\beta_i} \in \mathbb{N}$)

Initially, for an independent clock (without any relation with other clocks), the X-axis value of the timestamps of the clock can be set with any natural number. The values will be updated according to the clock relations applied on this clock. Let us take a look on the Fig. 5.1, the clock C_b has relations with the clocks C_a and C_c ($C_a \prec C_b \prec C_c$), accroding to the assignment rules, we must have $x_{a\_1} < x_{b\_1} < x_{c\_1}$. In this figure, the timestampe of the first occurrence of clock C_b is initially set as (5, 1). However, it can also be intially set as $(7, 1)$ only if the value is larger than the timestamp of the first occurrence of clock C_a (in the figure $x_{a\_1} := 1$) and smaller than the timestamp of the first occurrence of clock C_c (in the figure $x_{c\_1} := 8$).

More clock constraints may be added because of new requirements. In this case, these timestamps in the Fig.5.1 may also be updated according to the assignment rules. For example, after we add other four clock relations ($C_a \prec C_g, C_g \prec C_e, C_f \prec C_c, C_e = C_p$), if the delay from C_f to C_c is 2, then the virtual timestamps are updated as shown in Fig.5.2 by following the rules.

5.3 Time Constraint Conflicts

Since timestamps may be updated because of new adding relations, clock delays are also updated, which may cause time constraint conflicts. For example, in Fig. 5.1, assume the delay bound of C_c is $[2, 5]$. Before we add the relation $C_f \prec C_c$, there is no time constraint conflict since $t_{C_c[1]} = 8 - 5 = 3 \in [2, 5]$. However, after adding this relation, as shown in the Fig. 5.2, we found out that $t_{C_c[1]} = 12 - 5 = 7 \notin [2, 5]$. Here we give a formal definition of time constraint conflicts.

Definition 21 (Time Constraints conflicts) Let C_α be a clock built on timed-action $\alpha(p)^{t_\alpha | b_{t_\alpha}}$ [1]. A time constraint conflict of clock C_α exists if $\exists i \in \mathbb{N}$, $t_{\alpha_i} \notin b_{t_{\alpha_i}}$.

5.4 Calculate Delays and Delay Bounds

In timed-pNets, non-leaf nodes are the synchronization devices of their subsystems. The delays and delay bounds of the global logical clocks in these non-leaf nodes are computed in terms of the local logical clocks in the subsystems. When building these non-leaf nodes, time constraint conflicts may happen. In this section, we discuss how to compute the delays and delay bounds of these global clocks in the non-leaf nodes so that we can check if time constraint conflicts exist.

According to the timed-pNets definition (see definition 11), local logical clocks coincide with the corresponding global logical clock. According to the virtual timestamp assignment rules, the virtual timestamps of these local clocks equal to the timestamps of their global clocks. Usually, the delay of a global clock could be the sum of delays of a sequence of local clocks along a causality path. Let us take Fig. 5.3 as an example. In this simple system, C_{g1} and C_{g2} are global clocks of C_α and C_β. The delay between the two

[1] where the delay bound b is exposed since we need to discuss it in this chapter. The definition of timed-action can be found in the chapter 3.

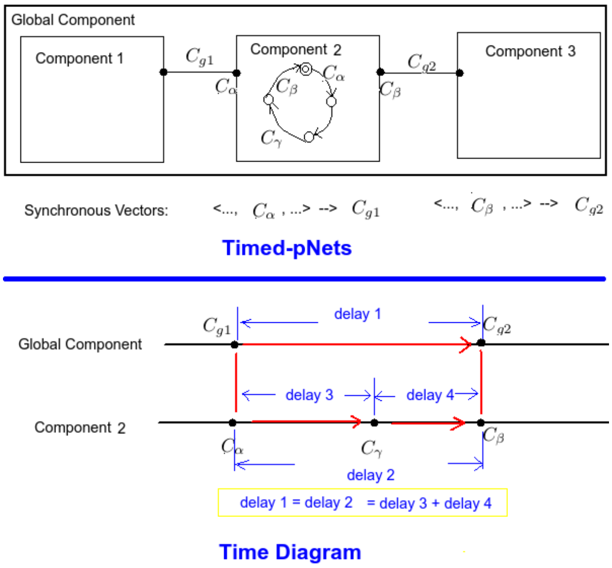

Fig. 5.3: A Small Example

global clocks is calculated from C_α to C_β along path $C_\alpha \rightarrow C_\gamma \rightarrow C_\beta$ as shown in the time diagram part in Fig. 5.3.

Since the delay of a global clock could be the sum of the delays of local clocks, in order to clearly define delays for global clocks, here we first give the definitions of causal clocks and causality paths.

5.4.1 Causal Clocks and Causality Paths

Definition 22 (Causal Clocks) Given a timed specification $TS :< \mathcal{C}, \mathcal{R} >$ with a set of clocks $\mathcal{C}$ and clock relations $\mathcal{R}$. Let $C_\alpha (\in \mathcal{C})$ be a clock. $C_{\angle\alpha} (\in \mathcal{C})$ is a causal clock of C_α if it satisfies:

(1.) relation $C_{\angle\alpha} \prec C_\alpha \in \mathcal{R}$,

(2.) $\nexists$ a clock $C_\gamma (C_\gamma \in \mathcal{C})$ with relation $C_{\angle\alpha} \prec C_\gamma \prec C_\alpha$.

For example, assume we have a timed specification $TS :< \mathcal{C}, \mathcal{R} >$ with clock set $\mathcal{C} = \{\mathcal{C}_\alpha, \mathcal{C}_\beta, \mathcal{C}_\gamma\}$ and relation set $\mathcal{R} = \{\mathcal{C}_\alpha \prec \mathcal{C}_\beta \prec \mathcal{C}_\gamma\}$. We say that C_α is a causal clock of C_β, but not a causal clock of C_γ.

Definition 23 (Causality Paths) Given a timed specification $TS :< \mathcal{C}, \mathcal{R} >$. A causality Path from clock C_0 to clock C_n (denoted as $p_{\{C_0 \to C_n\}}$) is a sequence of clocks with the conditions of:

(1.) starting from clock C_0

(2.) ending with clock C_n

(3.) $\forall C_i (i \in [0, n]), C_i$ is a causal clock of $C_{(i+1)}$

For example, in Fig. 5.2, $C_a \to C_b \to C_c$ is a causality path from C_a to C_c. $C_d \to C_e \to C_f \to C_c$ is a causality path from C_d to C_c.

Notice that we do not count $C_a \to C_d \to C_e \to C_f \to C_c$ as a causality path because 1) usually in our model the coincidence relations exist between two components for modelling synchronous communications. However we do not handle the delays between different components in local component level。 They will be handled in the upper level; 2) by including the paths with coincidence relations, we only increase the unnecessary paths that do not contribute to compute the delays.

5.4.2 Computing Delays of clocks

Here we define two kinds of delays. One is a simple clock delay that is the maximum time gap from the causal clocks of a clock to this clock. Another is a delay between two clocks that are connected by a path. The second one helps us to compute the delays of any two clocks that are not closed to each other but can be reached from one clock to another one following a causal path.

Definition 24 (Delays of Clock Occurrences) Given a timed specification $TS :< \mathcal{C}, \mathcal{R} >$. Let $\{C_k\}(k \in K \subset \mathbb{N})$ be the set of causal clocks of $C_\alpha(C_\alpha, C_k \in \mathcal{C})$ in the TS. The delay of the occurrence $C_\alpha[i](i \in \mathbb{N})$ is denoted as $t_{C_\alpha[i]}$, which describes a time delay before the occurrence $C_\alpha[i]$ can be executed. The delay is calculated from the corresponding occurrences of the causal clocks of C_α by the formula $t_{C_\alpha[i]} = max\{x_{C_\alpha[i]} - x_{C_k[i]} | k \in K\}$.

The delay variable of a timed-action captures the time (delay) that must elapse before the actions can be executed. In a logical clock, the delays of the different timed-action occurrences may be different. Let us take Fig. 5.2 as an example. The delay of the first occurrence of C_b (which is $t_{C_b[1]} = x_{C_b[1]} - x_{C_a[1]} = 5-1 = 4$) is different from the delay of the second occurrence of C_b (which is $t_{C_b[2]} = x_{C_b[2]} - x_{C_a[1]} = 14 - 13 = 1$). When a clock has more than one causal clocks, the delay of the clock takes the maximum value among the delays that come from all the causal clocks of the clock to this clock. Let us take the clock C_c as an example. Since it has two causal clocks C_b and C_f, the delay of the first occurrence of C_c is $t_{C_c[1]} = max\{(x_{C_c[1]} - x_{C_b[1]}), (x_{C_c[1]} - x_{C_f[1]})\} = max\{7, 2\} = 7$. Similarly, we can compute the delays of other occurrences.

Definition 25 (Delays along a Causality Path) Given a causality path $p_{\{C_0 \to C_n\}} = C_0 \to C_1 \to \ldots C_i \to \ldots \to \ldots C_n$. The delay from occurrence $C_0[r]$ to $C_n[r]$ along the causality path (denoted as $t_{p_{\{C_0[r] \to C_n[r]\}}}$) is defined as $t_{p_{\{C_0[r] \to C_n[r]\}}} = x_{C_n[r]} - x_{C_0[r]} (r \in \mathbb{N})$.

Let us take the path $C_d \to C_e \to C_f \to C_c$ as an example. The delay from $C_d[1]$ to $C_c[1]$ along the causality path $p_{C_d \to C_c}$ is $t_{p_{C_d[1] \to C_c[1]}} = 12 - 1 = 11$.

5.4.3 Computing Delay Bounds of Clocks

We define delay bounds as closed intervals over natural numbers. Three cases are discussed: the delay bound of a clock, the delay bound along a causal path and the delay bound along a set of causal paths. In the end, we propose theorem 4 to compute the delay bounds of global logical clocks from the local clocks in subsystems.

Definition 26 (The Delay Bound of a Clock) Given a clock C_α that is built on timed-action $\alpha(p)^{t_\alpha | b_{t_\alpha}}$. The delay bound of the clock C_α (denoted as b_{C_α}) is a closed interval $[l(b_{C_\alpha}), u(b_{C_\alpha})]$ over a set of natural numbers $\mathbb{N}$. The lower bound $l(b_{C_\alpha})$ is the minimal value of the closed interval. The upper bound of $u(b_{C_\alpha})$ is the maximal value of the closed interval. The clock delay bound applies to all occurrences $C_\alpha[i]$, formally $\forall i, b_{t_{\alpha_i}} = b_{C_\alpha}$.

Definition 27 (The Delay Bound along a Causality Path) Given a causality path $p_{\{C_0 \to C_n\}} = C_0 \to C_1 \to \ldots C_i \to \ldots \to \ldots C_n$. Let the delay bound of the clock $C_i (i \in [0, n])$ be $[l(b_{C_i}), u(b_{C_i})]$. Then the delay bound from C_0 to C_n along the causal path $p_{\{C_0 \to C_n\}}$ (denoted as $b_{p_{\{C_0 \to C_n\}}}$) is defined as $b_{p_{\{C_0 \to C_n\}}} = [\sum_{i \in [1,n]} l(b_{C_i}), \sum_{i \in [1,n]} u(b_{C_i})]$.

We take the causality path $p_{C_d \to C_c}$ as an example. Assume the delay bound of C_d, C_e, C_f and C_c are $[1, 3]$, $[3, 8]$, $[1, 7]$ and $[2, 9]$, then the delay bound of the causality path $b_{p_{\{C_d \to C_c\}}} = [6, 24]$.

Definition 28 (Delay Bound on a set of Causality Paths) Let $P_{\{C_0 \to C_n\}} = \{p^j_{\{C_0 \to C_n\}}\} (j \in \mathbb{N})$ be a set of causality paths from C_0 to C_n. Let the delay bound from C_0 to C_n on the j^{th} path be $b_{p^j_{\{C_0 \to C_n\}}}$. The delay bound from C_0 to C_n on the set of paths $P_{\{C_0 \to C_n\}}$ (denoted as $b_{P_{\{C_0 \to C_n\}}}$) is defined as $b_{P_{\{C_0 \to C_n\}}} = [max\{l(b_{p^j_{C_0 \to C_n}}) | j \in \mathbb{N}\}, min\{u(b_{p^j_{C_0 \to C_n}}) | j \in \mathbb{N}\}]$ in which $(l(b_{p^j_{C_0 \to C_n}})$ is the lower bound of $b_{p^j_{C_0 \to C_n}}$ and $u(b_{p^j_{C_0 \to C_n}})$ is the upper bound of $b_{p^j_{C_0 \to C_n}}$.

Example 10 Let us still take the Fig.5.2 as an example. The set $P_{\{C_a \to C_c\}}$ includes two paths $p^1_{C_a \to C_c} = C_a \to C_b \to C_c$ and $p^2_{C_a \to C_c} = C_a \to C_g \to C_e \to C_f \to C_c$. Assume the delay bound of C_a, C_b, C_g, C_e, C_f and C_c are $[1, 3]$, $[3, 19]$, $[2, 8]$, $[3, 8]$, $[1, 7]$ and $[2, 9]$. Then from definition 27 we know that $b_{p^1_{C_a \to C_c}} = [5, 28]$ and $b_{p^2_{C_a \to C_c}} = [8, 32]$. Then the delay bound of $P_{\{C_a \to C_c\}}$ can be computed as $b_{P_{C_a \to C_c}} = [max\{5, 8\}, min\{28, 32\}] = [8, 28]$.

Compute The Delay Bounds of Global Clocks in Timed-pNets

According to the definition of timed-pNets in chapter 4, a global logical clock is generated by at least one local logical clock. The delay of two global logical clocks can be calculated from their local clocks. And the two local clocks (one for generating the global clock, another one for generating the causal clocks of the global clock) must exist in one hole. The theorem 4 tells

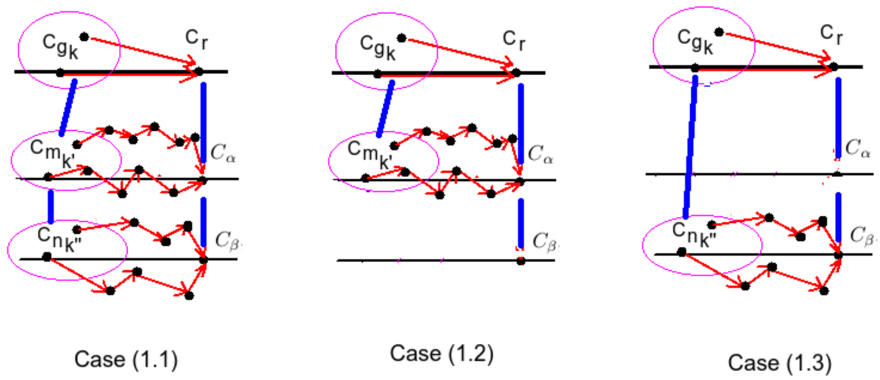

Fig. 5.4: Three cases in Theorem 4

us how to calculate the delay bounds of global clocks from their local clocks.

Theorem 4 (The Delay Bounds of Global Clocks) Given a timed-pNet $< P, A_G, \mathfrak{C}_G, J, \widetilde{A}_J, \widetilde{\mathfrak{C}}_J, \widetilde{\mathcal{R}}_J, \overrightarrow{V} >$. Assume that all local clocks (in the set $\widetilde{\mathfrak{C}}_J$) have no time constraint conflict. Consider a global clock C_γ and let $\mathbb{C}_g = \{C_{g_k}\}(k \in \mathbb{N})$ be the set of causal clocks of C_γ ($\mathbb{C}_g \subseteq \mathfrak{C}_G$, $C_\gamma \in \mathfrak{C}_G$, $\gamma \triangleq \gamma(p_\gamma)^{t_\gamma|b_{t_\gamma}}$).

(1) When $\overrightarrow{v} =< ..., C_\alpha, ..., C_\beta, ... > \rightarrow C_\gamma$. As shown in Fig. 5.4, let $\mathbb{C}_m = \{C_{m_{k'}}\}(k' \in \mathbb{N})$ be a set of local clocks that are in the same hole as C_α, and that contribute to generate the global clocks in $\mathbb{C}_g$. Let $\mathbb{C}_n = \{C_{n_{k''}}\}$ be a set of local clocks that are in the same hole as C_β, and that contribute to generate the global clocks also in $\mathbb{C}_g$.

(1.1) If $< \ldots, C_{m_{k'}}, \ldots, C_{n_{k''}}, \ldots > \rightarrow C_{g_k}$ as shown the case (1.1) in Fig. 5.4, then
$b_{C_\gamma} = [min\{min\{l(b_{P_{C_{m_{k'}}\rightarrow C_\alpha}})|k' \in \mathbb{N}\}, min\{l(b_{P_{C_{n_{k''}}\rightarrow C_\beta}})|k'' \in \mathbb{N}\}\},$
$max\{max\{u(b_{P_{C_{m_{k'}}\rightarrow C_\alpha}})|k' \in \mathbb{N}\}, max\{u(b_{P_{C_{n_{k''}}\rightarrow C_\beta}})|k'' \in \mathbb{N}\}\}]$ $(C_{m_{k'}} \in \mathbb{C}_m, C_{n_{k''}} \in \mathbb{C}_n, k', k'' \in \mathbb{N})$;

(1.2) If $< \ldots, C_{m_{k'}}, \ldots, \ldots, \ldots > \rightarrow C_{g_k}$ as shown the case (1.2) in Fig. 5.4, then

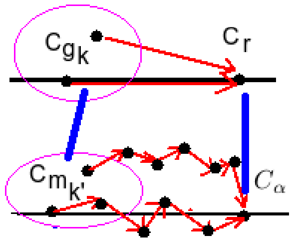

Fig. 5.5: Case 2 in Theorem 4

$b_{C_\gamma} = [min\{l(b_{P_{C_{m_{k'}}\to C_\alpha}})|k' \in \mathbb{N}\}, max\{u(b_{P_{C_{m_{k'}}\to C_\alpha}})|k' \in \mathbb{N}\}]$ $(C_{m_{k'}} \in \mathbb{C}_m, k' \in \mathbb{N})$,

(1.3) If $< \ldots, \ldots, \ldots, C_{n_{k''}}, \ldots > \to C_{g_k}$ as shown the case (1.3) in Fig. 5.4, then
$b_{C_\gamma} = [min\{l(b_{P_{C_{n_{k''}}\to C_\beta}})|k'' \in \mathbb{N}\}, max\{u(b_{P_{C_{n_{k''}}\to C_\beta}})|k'' \in \mathbb{N}\}]$ $(C_{n_{k''}} \in \mathbb{C}_n, k'' \in \mathbb{N})$,

(2) When $\overrightarrow{v} =< ..., C_\alpha, ..., \ldots, ... > \to C_\gamma$. Let $\mathbb{C}_m$ be a set of local clocks that in the same hole as C_α, and that contribute to generate the global clocks in $\mathbb{C}_g$ as shown in Fig. 5.5. Then $b_{C_\gamma} = [min\{l(b_{P_{C_{m_{k'}}\to C_\alpha}})|k' \in \mathbb{N}\}, max\{u(b_{P_{C_{m_{k'}}\to C_\alpha}})|k' \in \mathbb{N}\}]$ $(C_{m_{k'}} \in \mathbb{C}_m, k' \in \mathbb{N})$.

Proof. (1.1)Choose any occurrence of C_γ, for example, the i^{th} occurrence $C_\gamma[i]$ $(i \in \mathbb{N})$. According to definition 24 in page 108, $t_{\gamma\_i} = max\{x_{C_\gamma[i]} - x_{C_{g_k}[i]}|k \in \mathbb{N}\}(C_{g_k}[i] \in \mathbb{C}_g)$. Let L (resp. U) be the lower (resp. upper) bound of b_{C_γ}, that is
$(L = min\{min\{l(b_{P_{C_{m_{k'}}\to C_\alpha}})|k' \in \mathbb{N}\}, min\{l(b_{P_{C_{n_{k''}}\to C_\beta}})|k'' \in \mathbb{N}\}\})$;
$U = max\{max\{u(b_{P_{C_{m_{k'}}\to C_\alpha}})|k' \in \mathbb{N}\}, max\{u(b_{P_{C_{n_{k''}}\to C_\beta}})|k'' \in \mathbb{N}\}\})$.
To simplify the proof, we set $l(b_{P_{C_{m_{k'}}}}) < l(b_{P_{C_{m_{k'+1}}}})$ (resp.$l(b_{P_{C_{n_{k''}}}}) < l(b_{P_{C_{n_{k''+1}}}})$). Moreover, if $C_{m_{k'}}$ (resp. $C_{n_{k''}}$) generates clock C_{g_k}, then we let $k' = k'' = k$.

Assume $t_{\gamma\_i} < L$, then we have $max\{x_{C_\gamma[i]} - x_{C_{g_k}[i]}|k \in \mathbb{N}\} < L$. Let us take any causal clock from $\mathbb{C}_g$, for example C_{g_1} $(C_{g_1} \in \mathbb{C}_g)$, then we have $x_{C_\gamma[i]} - x_{C_{g_1}[i]} < max\{x_{C_\gamma[i]} - x_{C_{g_k}[i]}|k \in \mathbb{N}\} < L = min\{min\{l(b_{P_{C_{m_{k'}}\to C_\alpha}})|k' \in \mathbb{N}\}, min\{l(b_{P_{C_{n_{k''}}\to C_\beta}})|k'' \in \mathbb{N}\}\} \leq min\{l(b_{P_{C_{m_1}\to C_\alpha}}), l(b_{P_{C_{n_1}\to C_\beta}})\}$. According to the definition 20 in page 104, we have $x_{C_\alpha[i]} - x_{C_{m_1}[i]} = x_{C_\gamma[i]} - x_{C_{g_1}[i]}$. By the two formulas, we conclude $x_{C_\alpha[i]} - x_{C_{m_1}[i]} < min\{l(b_{P_{C_{m_1}\to C_\alpha}}), l(b_{P_{C_{n_1}\to C_\beta}})\} <$

$l(b_{P_{C_{m_1} \to C_\alpha}})$, which means that the delay of $C_\alpha[i]$ (from $C_{m_1}[i]$ to $C_\alpha[i]$) is less than its lower delay bound. It contradicts the fact that all local clocks have no time confict. So, we have $t_{\gamma\_i} \geq L$.

Similar, assume $t_{\gamma\_i} > U$, then we have $max\{x_{C_\gamma[i]} - x_{C_{g_k}[i]} | k \in \mathbb{N}\} > U$. Let us take clock C_{g_h} that satisfies $x_{C_\gamma[i]} - x_{C_{g_h}[i]} = max\{x_{C_\gamma[i]} - x_{C_{g_k}[i]} | k \in \mathbb{N}\}$. Then we have $x_{C_\gamma[i]} - x_{C_{g_h}[i]} = x_{C_\alpha[i]} - x_{C_{m_h}[i]} > U = max\{max\{u(b_{P_{C_{m_{k'}}} \to C_\alpha})|k' \in \mathbb{N}\}, max\{u(b_{P_{C_{n_{k''}}} \to C_\beta})|k'' \in \mathbb{N}\}\} \geq max\{u(b_{P_{C_{m_h}} \to C_\alpha}), u(b_{P_{C_{n_h}} \to C_\beta})\} \geq u(b_{P_{C_{m_h}} \to C_\alpha})$. So we get $x_{C_\alpha[i]} - x_{C_{m_h}[i]} > u(b_{P_{C_{m_h}} \to C_\alpha})$, which means the delay of $C_\alpha[i]$ (from $C_{m_h}[i]$ to $C_\alpha[i]$) is more than its upper delay bound. It contradicts the fact that all local clocks have no time confict. So we have $t_{\gamma\_i} \leq U$.

For the other cases (1.2), (1,3) and (2), their proofs are similar as the proof for (1.1). $\square$

Notice that we cannot use the theorem if constraint conflicts exist among the local clocks. To build a upper level of timed-pNets and compute the delay bounds of global clocks, we must first solve all conflicts in local holes. Besides, according to the timed-pNets definition 11 in the page 68, we know that each global clock can be generated by only one synchronization vector, so in our proof we just discuss a single vector not a set of vectors.

Example 11 Let us take Fig. 5.6 as an example. In this figure, $p1$, $p2$ and $p3$ are in one hole. $p4$ is in another hole. From the previous analysis, we know that the delay bound of the set of paths from C_a to C_c is [8, 28]. From the figure we can see that global clock C_{g2} is generated by synchronous vector $< ..., C_c, ..., C_v, ... > \to C_{g2}$. Global clock C_{g1} is generated by the synchronous $< ..., C_a, ..., C_u, ... > \to C_{g1}$. And clock C_{g1} is the causal clock of C_{g2}. Assume the delay bound from C_u to C_v is [7,18] as shown in the figure with green numbers. According to the case (1.1) in theorem 4, we can get the delay bound of the global clock C_{g2} is $b_{C_\gamma} = [min\{7, 8\}, max\{18, 28\}] = [7, 28]$.

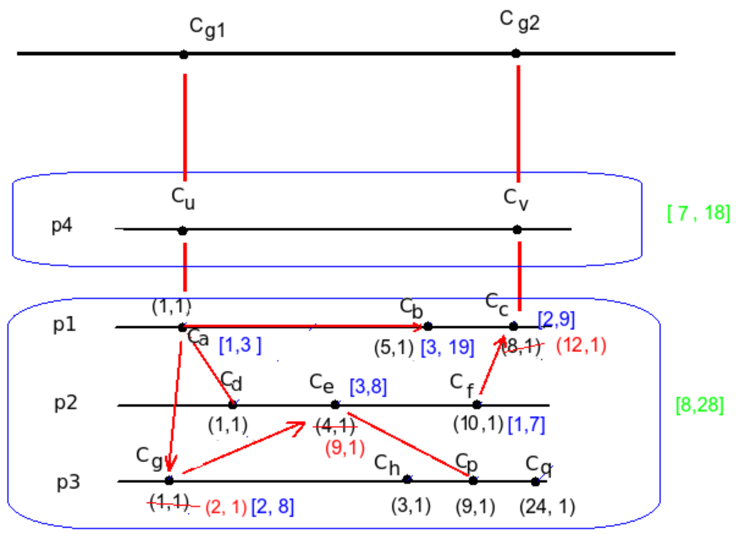

Fig. 5.6: Example of computing Global Delay Bound

5.5 Simulation

We simulate the system shown as the Fig. 4.8 in page 71 by means of the TimeSquare tool [41]. This tool is able to check system time constraint conflicts and time properties. Two input files are required by TimeSquare (see Fig. 5.7). One contains the system timed specifications deduced from Fig.4.8; another contains the system timed properties. We import a reference clock into the two files. For simplifcication, we choose a reference clock that ticks periodically. All delays and delay bounds of other logical clocks are specified in terms of this reference clock. For example, in our simulation, we assume that the delay bounds of all action occurrences are between $[1, 3]$ in the sense that the delays of those actions should stay between the first and the third occurrences of the reference clock. The simulation results tell us if the time properties are satisfied by the specifications. For simplification, . We do not fix their delays so that our model is more flexible. The properties we would like to check are as follows:

- (P1.) No conflict exists.

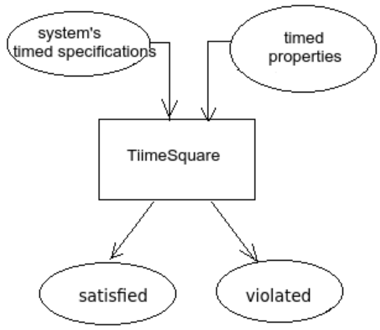

Fig. 5.7: Property Checking

- (P2.) The delay of the global clock $C_{notify_{g1}}$ is no more than 3. Formally, let $notify_{g1}\_i$ $(i \in \mathbb{N}) \triangleq notify_{g1}^{t_{notify_{g1}\_i} | b_{notify_{g1}}}$, then $\forall i \in \mathbb{N}, 1 \leq t_{notify_{g1}}\_i \leq 3$.

- (P3.) The minimal and maximal distance between clock $C_{?Cmd}$ and $C_{!R}$ are 6 and 11. We denote them as $MinDis(C_{?Cmd_{g5}}, C_{!R_{g6}}) = 6$ and $MaxDis(C_{?Cmd_{g5}}, C_{!R_{g6}}) = 11$.

5.5.1 Encode Properties into TimeSquare

Here we explain how to encode our properties into the TimeSquare. We translate the properties to the form that the TimeSquare tool can accept. We design bounded precedence relations (denoted as "$\prec_{[min,max]}$") that are precedence relations with minimal and maximal bounds. For example, for the property P2, we check if the delay of the clock $C_{notify_{g1}}$ is in the interval [1,3]. Since the delay of the clock $C_{notify_{g1}}$ captures the time that must elapse from the clock $C_{?Cmd_{g5}}$, checking the property P2 translates to check the bounded relation $C_{?Cmd_{g5}} \prec_{[1,3]} C_{notify_{g1}}$.

We use the "DelayFor" function provided in TimeSquare to create the bounded precedence relations. The "DelayFor" function has three parameters: 1) the causal clock of $C_{notify_{g1}}$ (in our example the clock is $C_{?Cmd_{g5}}$), 2) the base counter (in our case is a reference clock "$baseCounter$"), 3) the delay value to be set. We encode the bounded precedence relation function by following the steps:

- First we define minimal and maximal bound expressions. For example, in our case, we define two expressions "$minDelayBound \triangleq DelayFor(C_{?Cmd_{g5}}, baseCounter, 1)$" and "$maxDelayBound \triangleq DelayFor(C_{?Cmd_{g5}}, baseCounter, 3)$",

- Then we limit the clock $C_{notify_{g1}}$ into the bound by using precedence relations. For example, we set two precedence relations: "$minDelayBound \prec C_{notify_{g1}}$" and "$C_{notify_{g1}} \prec maxDelayBound$".

Similarly, the property P3 can be translated to $C_{?Cmd_{g5}} \prec_{[6,11]} C_{!R_{g6}}$.

5.5.2 Property Checking

We input the system timed specifications and properties into TimeSquare to check if a violation exists .

- TimeSquare reports us an error as shown in Fig. 5.8 when checking the property P1.

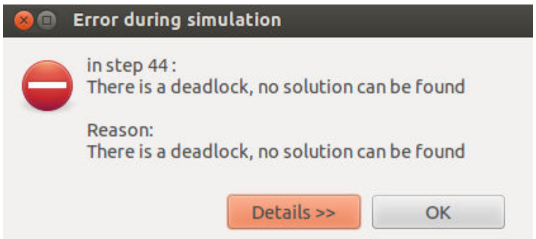

Fig. 5.8: Checking the property (1)

This error is caused by time constraint conflicts. Fig.5.9 represents a time diagram with possible virtual timestamps. In this figure, the blue numbers illustrate the virtual timestamps when those components are independent (without communications). These numbers are assigned randomly but following the virtual timestamp assignment rules 20. After composing those components by adding communications among

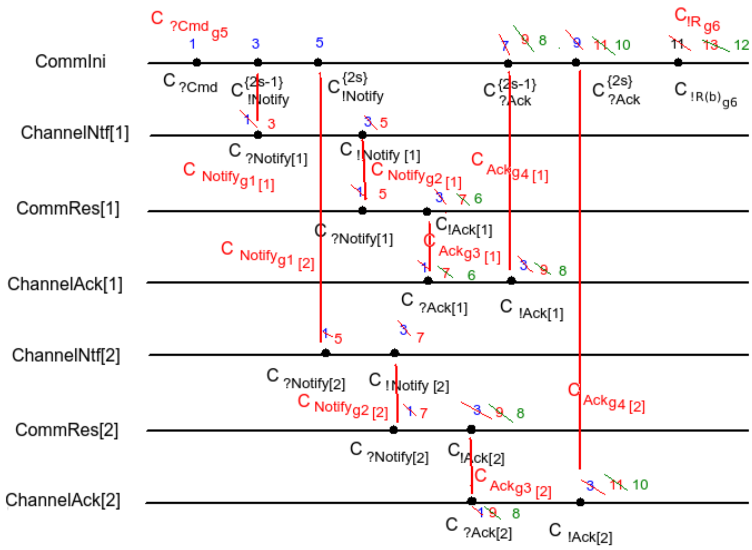

Fig. 5.9: Time Constraint Conflicts

them (represented by coincidence relations in terms of the synchronous vectors), those virtual timestamps are recomputed in terms of the assignment rules 20 as shown with red numbers. By analyzing those updated virtual timestamps, we can see that a time constraint conflict happens on the clock $C^{\{2s-1\}}_{?Ack}$ ($x_{C^{\{2s-1\}}_{?ack}[1]} - x_{C^{\{2s\}}_{!notify}[1]} = 9-5 = 4 \notin [1,3]$).

To fix the issue, we set the delay of $?Notify\_i$ in component "CommRes" to 1 (denoted as $\forall i \in \mathbb{N}, t_{?Notify\_i} := 1$). Moreover, we limit the delays of all clocks less than 2 except clock $C^{\{2s-1\}}_{?Ack}$ (formally, $\forall i \in \mathbb{N}, t_{\alpha\_i} \leq 2,\ C_\alpha \in \widetilde{C}_J \backslash C^{\{2s-1\}}_{?Ack}$). After redoing the simulation, we found out that no conflict exists. TimeSquare outputs VCD view as shown in Fig.5.10, in which the first row is the reference clock and the other rows are global logical clocks. The red arrows in this figure demonstrate the precedence relations of these clocks. For simplification, we take few clocks that will be used to explain the next two properties from the VCD view. And then we add white and blue lines for giving a clear explanation. The blue lines are used to separate the cycles. Here we list 5 cycles.

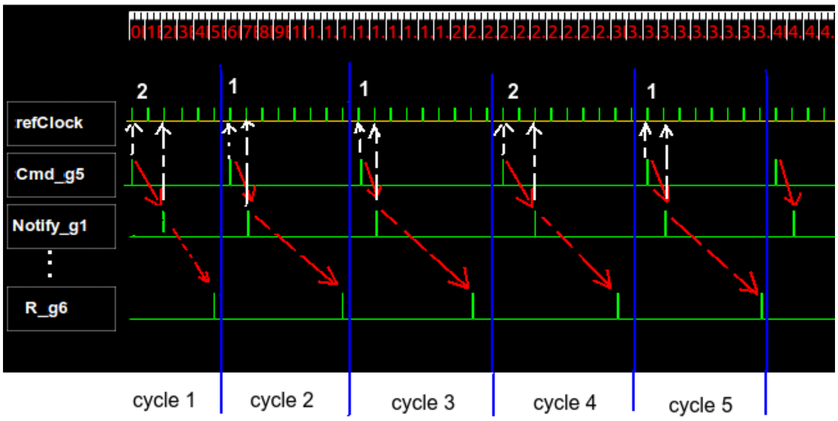

Fig. 5.10: Checking property P1 and P2

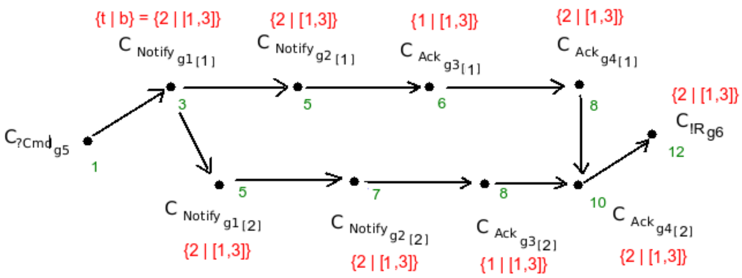

Fig. 5.11: The dependency graph of global clocks

- To check property P2, we encode $C_{?Cmd_{g5}} \prec_{[1,3]} C_{notify_{g1}}$ into TimeSquare as an assert. TimeSquare tool does not report any violation, which means the property is satisfied. This result can also be seen from the white arrows and white numbers in Fig.5.10, in which the the delays of the occurrence $Notify_{g1}\_i$ in these cycles are less than 3 $(\forall i \in \mathbb{N}, t_{notify_{g1}\_i} < 3)$.

Actually, from the Fig. 5.9 we can get the dependency graph of the system global logical clocks as shown in Fig. 5.11, in which the precedence relations of these global clocks are represented by arrows. According to the theorem 4, we can compute the delay bounds of these global clocks. Take the clock $C_{Notify_{g1}}$ as an example, we get the delay bound $b_{t_{Notify_{g1}[1]}} = [1,3]$ and the delay $t_{Notify_{g1}\_1} = x_{C_{Notify_{g1}}[1]} - x_{C_{?Cmd_{g5}}[1]} =$

2. The delays and delay bounds of other global clocks can also be calculated. And they are represented with red numbers in Fig.5.11. So we can also check the delay constraints of other global clocks as we did for the clock $C_{Notify_{g1}}$ in this property.

- To check property P3, we set assertion $C_{?Cmd_{g5}} \prec_{[6,11]} C_{!R_{g6}}$. TimeSquare does not report any error. But if we modify the property, for example, as $C_{?Cmd_{g5}} \prec_{[5,10]} C_{!R_{g6}}$, then we get an error reported from TimeSquare as shown in Fig.5.12.

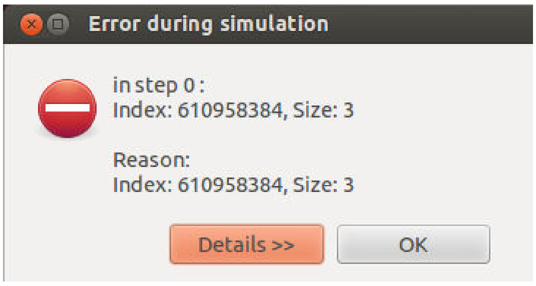

Fig. 5.12: Checking property P3

5.5.3 Discussion

From the simulation we can see that our model is able to check the time properties after we import a reference clock to this model. Compared to other real-time models such as timed-automata, we actually decouple the real-time from our model. In other words, if we choose chronometric clock as a reference clock, then our model can be used to analyze real-time systems. According to the paper [82], it is possible to transfer our model to automata. Since it is not the topic of our thesis, we do not investigate how to do it and so it is not clear about the comparison. It will be our future work. However, it is clear that the real advantage of our model is that even though we do not necessarily rely on real-time clock (or common physical clock), we still can analyze the system time properties if we choose a logical clock as the reference clock. This character makes our model fit for modelling distributed systems. Moreover, this decoupling also helps to release the work of refinement. Think about that the system requirement on time constraints may be changed, which may result to modify the system specification since from beginning, but if we import the reference clock in the end before we check the time properties, what we need to modify on the specification is just the links between the reference clock and other logical clocks.

5.6 Conclusion

In this chapter, we investigated the delay constraints of timed-pNets. We took an example from chapter 4 to explain how to compute the delays and the delay bounds of global logical clocks. In the end, we use TimeSquare to check time constraint conflicts and some latency properties. From the chapters 4 and 5, we conclude that our model is able to detect system's logical design errors, to check time constraint conflicts, and to verify time properties.

The flexibility and simplicity of the timed-pNets mainly due to the design of timed specifications that is the critical part of the model. However, the basic ways of building timed specifications introduced in chapter 4 are not enough to model some complex situations. For example, we can easily model a precedence relation on two clocks, in which the relation applies to all occurrences. But in reality, it may happen that the precedence relation of two clocks only applies to some occurrences of them. It is much more complicated to implement it by only using the precedence definition proposed in the chapter 4.

In next chapter we will introduce an extension of timed-pNets, which includes clock partition and clock union to simplify the way of generating the timed specifications for complicated situations.

Chapter 6 Extension of Timed-pNets

In this chapter, we design the concepts of clock partition and clock union to simplify the way of encoding timed specifications. The clock partition allows us to flexibly split the occurrences of timed-actions into groups so that the clock relations can be applied to the groups instead of to a single occurrence. We prove that the relations (precedence and coincidence relations) on partition clocks can be substituted by those relations on a set of filtered clocks, which illustrates the advantages of using partition clocks: simple and easy to understand. Another extension, Clock Union, provides us with a way to compose logical clocks. Usually it is used to specify the branches of transition systems. We apply the two concepts to our car inserting example, and demonstrate the way of building the timed specifications by them. In the end, the simulations and corrections are implemented in the TimeSquare tool.

6.1 Context and problematic

In chapter 4 we discussed precedence and coincidence relations, in which the relation operators ("$\prec$" and "$=$") apply to all pairs of corresponding timed-action occurrences as shown in Fig 4.4 in page 62. The small kernel used in the chapters 4 and 5 keeps the definitions and proofs as small as possible. However, this way is not flexible when facing the case that the relations do not apply to all occurrences. Let us take the "Control" component in Fig. 4.5 in the page 64 as an example. After the action "$?Consensus(ExpRes)^{t_o}$" executes, the action "$LocExe^{t_x}$" can execute undetermined times before going to the next action "$!Finish^{t_f}$". In other words, the precedence relation between clock $C_{?Consensus(ExpRes)^{t_o}}$ and $C_{LocExe^{t_x}}$ does not apply to all corresponding occurrences. To solve the issue, we design the concept of clock partition that provides a way to split timed-action occurrences into groups. Then partition clocks and the relations on them are defined to help us flexibly set relations on those timed-action occurrences, and in the end provide us flexibility for system specifications.

In order to be able to specify the undetermined clock choices (e.g. branches) in the transition systems, we define a clock union operator ("$\dot{+}$") to compose two logical clocks (e.g. $C_\alpha \dot{+} C_\beta$) in the sense that either clock C_α or clock C_β ticks. We call it clock union because we can consider the two united clocks (e.g. $C_\alpha \dot{+} C_\beta$) as a new logical clock (e.g. C_γ) that is created by the union of the two clocks and C_γ ticks whenever C_α or C_β ticks. Let us take "Initial" component in Fig. 4.5 as an example. After clock "$C_{?R(b)^t_R}$" ticks, either clock $C_{!Cancel^{t_L}}$ ticks or clock $C_{\tau^{t_\tau}}$ ticks. In this case, we specify their relations as $C_{?R(b)^t_R} \prec C_{!Cancel^{t_L}} \dot{+} C_{\tau^{t_\tau}}$ or $C_{?R(b)^t_R} \prec C_\gamma$ if $C_\gamma \triangleq C_{!Cancel^{t_L}} \dot{+} C_{\tau^{t_\tau}}$. Notice that we can not simply specify the branch as the relation $C_{?R(b)^t_R} \prec C_{!Cancel^{t_L}}$ and $C_{?R(b)^t_R} \prec C_{\tau^{t_\tau}}$, because the two precedence relations do not cover the semantics that the clocks $C_{!Cancel^{t_L}}$ and $C_{\tau^{t_\tau}}$ are exclusive.

Then, we take the "Control" and "Initial" components in Fig.4.5 in the pager 64 as an example to represent how to specify the systems by using the partition clocks and clock union operators. In the end, we check the system safety properties and timed properties in the TimeSquare tool.

This chapter is organized as follows. In section 6.2 we introduce clock partition as well as formal definitions of precedence and coincidence relations on partition clocks. Then, clock union is defined in section 6.3. Examples and simulations are illustrated in the section 6.4. In the end, we conclude the chapter in section 6.5.

6.2 Clock Partition

We define a partition of clock C_α as a division of the occurrences of timed-action α. It is a sequence of subsequences of the occurrences of α such that every occurrence α_i is in exactly one of these subsequences.

Definition 29 (Clock Partition) Let $X = \{x_i\}$ $(x_i, i \in \mathbb{N}^+)$ be a sequence of natural numbers. The partition of clock C_α $(= \{\alpha\_1, \alpha\_2, \ldots, \alpha\_k, \alpha\_k', \ldots\}, k, k' \in \mathbb{N}, k' = k{+}1)$ is a sequence of subsequences $\mathbb{S} = \{S_i\} = (\{\alpha\_i_1, \alpha\_i_2, \ldots, \alpha\_i_k, \alpha\_i_{k'}, \ldots, \alpha\_i_{x_i}\},$ $i_{k'} = i_k + 1, i_k, i_{k'}, i_{x_i} \in \mathbb{N}^+)$ of the occurrences of the timed-action α in terms of X such that:

- The length of the i^{th} subsequence of $\mathbb{S}$ equals to x_i. $(|S_i| = x_i)$
- The union of the subsequences in $\mathbb{S}$ equals to the occurrences of the timed-action α. $(\bigcup_{S_i \in \mathbb{S}} S_i = C_\alpha)$
- The order of the subsequences reserves the original order of the occurrences in C_α (let $S_j = \{\alpha\_j_1, \alpha\_j_2, \ldots, \alpha\_j_k, \alpha\_j_{k'}, \ldots, \alpha\_j_{x_j}\}$, $j_{k'} = j_k + 1, j_k, j_{k'}, j_{x_j} \in \mathbb{N}^+$. $\forall i, j \in \mathbb{N}^+$, if $j = i + 1$, then $\alpha\_j_1 = \alpha\_i_{x_i} + 1$).
- The intersection of any two distinct subsequences in $\mathbb{S}$ is empty. (if S_i,$S_j \in \mathbb{S}$ and $S_i \neq S_j$ then $S_i \cap S_j = \emptyset$, $i, j \in \mathbb{N}^+$)

According to the clock partition, we define a new clock in which timed-action occurrences are grouped in terms of the partition schema X. Fox example, if $X = \{2, 3, 1, 5, \ldots\}$, then the new clock can be represented as $\{\{\alpha\_1, \alpha\_2\}, \{\alpha\_3, \alpha\_4, \alpha\_5\}, \{\alpha\_6\}, \{\alpha\_7, \alpha\_8, \alpha\_9, \alpha\_10, \alpha\_11\}, \ldots\}$.

In order to flexibly adjust the speed of the new clock, we introduce the concept of idle actions in the sense that these actions do not participate in any communication and task execution. In a consequence, we do not build the relations between the idle actions and other timed-actions, but they do have an effect on the clock speed. This point will be well explained after we give the definition of Idle Actions, and it also can be seen in the sections 6.2.1 and 6.2.2. Here we first give the definitions of Idle Actions, and then define Partition Clocks in which idle actions are used in a partitioned clock to adjust the clock speed.

Definition 30 (Idle Actions) Idle Actions are the actions that stay in a logical clock to slow down the speed of the clock.

For example, let ρ be a idle action. Given a clock $C_\alpha = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots\}$, the clock $C'_\alpha = \{\alpha_1, \alpha_2, \rho, \alpha_3, \alpha_4, \ldots\}$ is a new clock that is one step slower than C_α after the timed-action occurrence α_2. Similar, $C''_\alpha = \{\alpha_1, \alpha_2, \rho, \rho, \alpha_3, \alpha_4, \ldots\}$ is a new clock that is two steps slower than C_α.

The effect on the speed of a clock can be seen clearly when we compare it with another clock. For example, let $C_\beta = \{\beta_1, \beta_2, \beta_3, \beta_4, \ldots\}$ be a clock without idle actions. Assume $C_\alpha = C_\beta$ as shown in the Fig. 6.1 (1), we can see that the two clocks are coincident in the sense that both clocks increase the same number of steps at any stopwatch. However, after adding a idle action in C_α as shown in the right side of the Fig. 6.1 (1), we can see that when α_3 occurs, the clock C_β has reached step β_4. It tells us that the clock C_α is one step slower than clock C_β due to the idle action. The same effect also applies to precedence relations as shown in the Fig. 6.1 (2).

Definition 31 (Partition Clocks) Let $X = \{x_i\}$ $(x_i, i \in \mathbb{N})$ be a sequence of natural numbers. Let ρ be an idle action. The new clock that is generated by the clock partition on clock C_α in terms of X and idle actions is called a partition clock (denoted as $C_\alpha^{P\{X\}}$). The empty subsequences $(x_i = 0)$ are filled by idle actions.

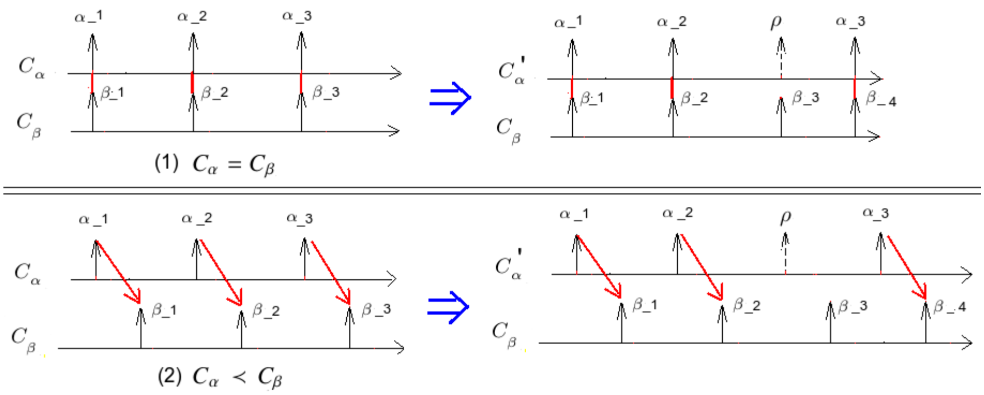

Fig. 6.1: Clock Relations with Idle Actions

Notice that in this definition, the assignment of x_i can be 0, which is looser than that is in the definition 29, so that the speed of the partition clocks is able to be adjusted. Fox example, if $X = \{2, 3, 0, 1, 5, \ldots\}$, then $C_\alpha^{P\{X\}} = \{\{\alpha\_1, \alpha\_2\}, \{\alpha\_3, \alpha\_4, \alpha\_5\}, \{\rho\}, \{\alpha\_6\}, \{\alpha\_7, \alpha\_8, \alpha\_9, \alpha\_10, \alpha\_11\}, \ldots\}$. It is slower than the clock $\{\{\alpha\_1, \alpha\_2\}, \{\alpha\_3, \alpha\_4, \alpha\_5\}, \{\alpha\_6\}, \{\alpha\_7, \alpha\_8, \alpha\_9, \alpha\_10, \alpha\_11\}, \ldots\}$.

6.2.1 Semantics of Precedence Relations on Partition Clocks

Here we introduce the semantics of precedence relations on partition clocks in three cases. In order to illustrate them, we take the same example for all cases. In this common example, we let $C_\alpha^{P\{X\}}$ be a partition clock with $X = \{x_i\} = \{2, 3, 0, 1, 5, \ldots\} (i \in \mathbb{N})$ and clock C_β be a normal clock that has not been partitioned.

- **[R1:]** $[\![C_\alpha^{P\{X\}} \prec C_\beta]\!] = \forall i$, if $x_i \neq 0$, then $\alpha\_(\sum_{j=1}^{i} x_j) \prec \beta\_i$
  Relation $R1$ applies to the case where a partition clock precedes a normal clock. The semantics of $R1$ tells that for each non empty subsequence on $C_\alpha^{P\{X\}}$, the last occurrence of the i^{th} subsequence in $C_\alpha^{P\{X\}}$ precedes the i^{th} occurrence of clock C_β. Fig.6.2 shows us a table in which we deduce the occurrence relations as well as a figure that

i	x_i	Relations $\alpha\_(\sum_{j=1}^{i} x_j) \prec \beta\_i$
$i = 1$	$x_i = 2$	$\alpha_2 \prec \beta_1$
$i = 2$	$x_i = 3$	$\alpha_5 \prec \beta_2$
$i = 3$	$x_i = 0$	$-----$
$i = 4$	$x_i = 1$	$\alpha_6 \prec \beta_4$
$i = 5$	$x_i = 5$	$\alpha_{11} \prec \beta_5$
...	...	...

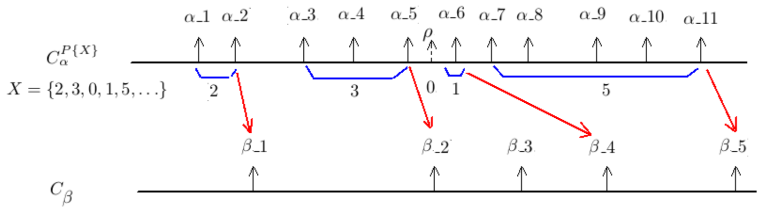

Fig. 6.2: Relation 1

demonstrates the relations.

- **[R2:]** $[\![C_\beta \prec C_\alpha^{P\{X\}}]\!]$ = Let $x_0 = 0$, $\forall i$, if $x_i \neq 0$, then $\beta\_i \prec \alpha\_(1 + \sum_{j=0}^{i-1} x_j)$
Relation $R2$ applies to the case where a normal clock precedes a partition clock. The semantics of $R2$ tells that for each non empty subsequence on $C_\alpha^{P\{X\}}$, the i^{th} occurrence of clock C_β precedes the first occurrence of the i^{th} subsequence in $C_\alpha^{P\{X\}}$. Fig.6.3 shows us a table in which we deduce the occurrence relations as well as a figure that demonstrates the relations.

- **[R3:]** Let $Y = \{y_i\}$ and $y_0 = 0$, $[\![C_\alpha^{P\{X\}} \prec C_\gamma^{P\{Y\}}]\!] = \forall i, j$, if $x_i \neq 0, y_i \neq 0$, then $\alpha\_(\sum_{j=1}^{i} x_j) \prec \gamma\_(1 + \sum_{k=0}^{i-1} y_k)$
Let $C_\gamma^{P\{Y\}}$ be a partition clock. Relation $R3$ illustrates the case of a precedence relation on two partition clocks. The semantics of $R3$ tells that for each non empty subsequence on $C_\alpha^{P\{X\}}$ and $C_\gamma^{P\{Y\}}$, the last occurrence of the i^{th} subsequence in clock $C_\alpha^{P\{X\}}$ precedes the first

i	x_i	Relations $\beta\_i \prec \alpha\_(1+\sum_{j=0}^{i-1} x_j)$
$i=1$	$x_i=2$	$\beta_1 \prec \alpha_1$
$i=2$	$x_i=3$	$\beta_2 \prec \alpha_3$
$i=3$	$x_i=0$	$-----$
$i=4$	$x_i=1$	$\beta_4 \prec \alpha_6$
$i=5$	$x_i=5$	$\beta_5 \prec \alpha_7$
$\ldots$	$\ldots$	$\ldots$

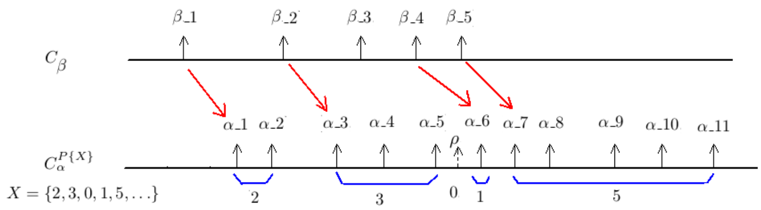

Fig. 6.3: Relation 2

occurrence of the i^{th} subsequence in $C_\alpha^{P\{X\}}$. Assume $Y = \{y_i\} = \{3,2,1,0,4,\ldots\}$. Fig.6.4 shows us a table in which we deduce the occurrence relations as well as a figure that demonstrates the relations.

Example 12 Let us take the "Control" component in Fig. 4.5 as an example. The way of partition depends on the guard "[ExpRes != CurData]". Assume the guard triggers 3 times self-loops in the first cycle, twice self-loops in the second cycle, and keep on triggering twice self-loops for the rest cycles. Then we can write a partition $X = \{x_i\} = \{3,2,2,\ldots\}$. The timed specification of the "Control" component can be written as: $C_{?Consensus} \prec C_{LocExe}^{P\{X\}} \prec C_{!Finish} \prec C_{?Consensus}^{\Delta(1)}$. According to the relations $R1$ and $R2$, we can draw the clock relations as shown in Fig.6.5.

6.2.2 Semantics of Coincidence Relations on Partition Clocks

This section represents the semantics of coincidence relations on partition clocks. We first define Occurrence Filter on a partition clock.

i	x_i	y_i	Relations $\alpha\_(\sum_{j=1}^{i} x_j) \prec \gamma\_(1+\sum_{k=0}^{i-1} y_k)$
$i=1$	$x_i=2$	$y_i=3$	$\alpha\_2 \prec \gamma\_1$
$i=2$	$x_i=3$	$y_i=2$	$\alpha\_5 \prec \gamma\_4$
$i=3$	$x_i=0$	$y_i=1$	$------$
$i=4$	$x_i=1$	$y_i=0$	$------$
$i=5$	$x_i=5$	$y_i=4$	$\alpha\_11 \prec \gamma\_7$
...	...	...	

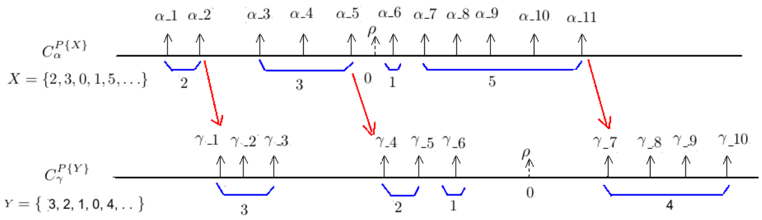

Fig. 6.4: Relation 3

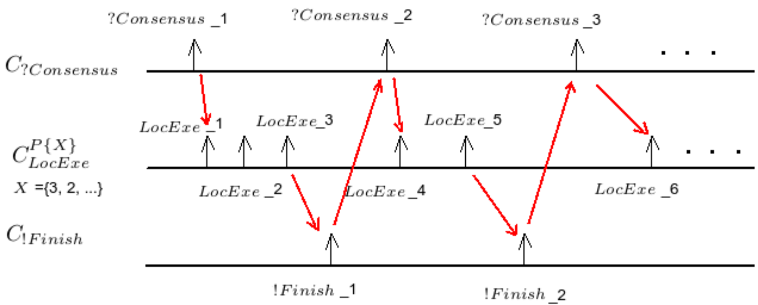

Fig. 6.5: One example of Control Component Clock Relations

Definition 32 (The k^{th} Filter of a partition clock) Let $C_\alpha^{P\{X\}} = \{\{\alpha\_1_1, \alpha\_1_2, \ldots, \alpha\_1_{x_1}\}, \{\alpha\_2_1, \alpha\_2_2, \ldots, \alpha\_2_{x_2}\}, \ldots, \{\alpha\_i_1, \alpha\_i_2, \ldots, \alpha\_i_{x_i}\}, \ldots\}$ be a partition clock $(X = \{x_i\})$. The k^{th} ($k \in \mathbb{N}$) filter of $C_\alpha^{P\{X\}}$ is $C_\alpha^{P\{X\}^{\triangleright k}} = \{\alpha\_1_k, \alpha\_2_k, \ldots, \alpha\_i_k, \ldots\}$ in which $\alpha\_i_k = \begin{cases} \rho & \text{for} \quad i_k > x_i \\ \alpha\_i_k & \text{for} \quad i_k \leq x_i \end{cases}$. The new clock filters out the k^{th} occurrence of each subsequence from the partition clock $C_\alpha^{P\{X\}}$.

For example, assume we have a partition clock $C_\alpha^{P\{X\}} = \{\{\alpha\_1, \alpha\_2\}, \{\alpha\_3, \alpha\_4, \alpha\_5\}, \{\rho\}, \{\alpha\_6\}, \{\alpha\_7, \alpha\_8, \alpha\_9, \alpha\_10, \alpha\_11\}, \ldots\}$. If we set k=2, then we have $C_\alpha^{P\{X\}^{\triangleright 2}} = \{\alpha\_2, \alpha\_4, \rho, \rho, \alpha\_8, \ldots\}$.

Two cases of relations are discussed: a partition clock coincides with a normal clock; a partition clock coincides with a family of normal clocks.

- **[R4:]** $[\![C_\alpha^{P\{X\}^{\triangleright k}} = C_\beta]\!] = [\![C_\beta = C_\alpha^{P\{X\}^{\triangleright k}}]\!] =$ Let $x_0 = 0$, $\forall i$, if $x_i \neq 0 \wedge k \leq x_i$, then $\alpha\_(k + \sum_{j=0}^{i-1} x_j) = \beta\_i$
  Relation $R4$ applies to the case where a filtered partition clock coincides a normal clock. The semantics of $R4$ tells that the k^{th} occurrence of the i^{th} subsequence in $C_\alpha^{P\{X\}}$ coincides the i^{th} occurrence of clock C_β. Fig.6.6 shows us an example with $k = 1$, in which a table represents the occurrence relations we deduced and a figure that demonstrates the relations.

- **[R5:]** $[\![C_\alpha^{P\{X\}} = C_\eta[n]]\!] =$ Let $x_0 = 0$, $\forall i, j \leq x_i$, if $x_i \neq 0$, then $\alpha\_(\sum_{r=0}^{i-1} x_r + j) = \eta_{[j]}\_k_j$ ($j \in [1, n]$), in which $k_j = N(\rho) + \sum_{r=0}^{i-1} x_r + j - (\sum_{j' \neq j, j'=1}^{n} k_{j'})$ ($N(\rho)$ is the sum of the occurrences of ρ till the i^{th} subsequence).
  Relation $R5$ applies to the case where a partition clock coincides with a family of normal clocks. Let $C_\eta[n]$ be a family of clock C_η, in which n is the length of the set $C_\eta[n] = \{C_{\eta[1]}, C_{\eta[2]}, \ldots, C_{\eta[n]}\}$. The semantics of $R5$ tells that for each non empty subsequence in the $C_\alpha^{P\{X\}}$, the j^{th}

i	x_i	Relations $\alpha\_(1 + \sum_{j=0}^{i-1} x_j) = \beta\_i$
$i = 1$	$x_i = 2$	$\beta_1 = \alpha_1$
$i = 2$	$x_i = 3$	$\beta_2 = \alpha_3$
$i = 3$	$x_i = 0$	$- - - - - -$
$i = 4$	$x_i = 1$	$\beta_4 = \alpha_6$
$i = 5$	$x_i = 5$	$\beta_5 = \alpha_7$
...	...	...

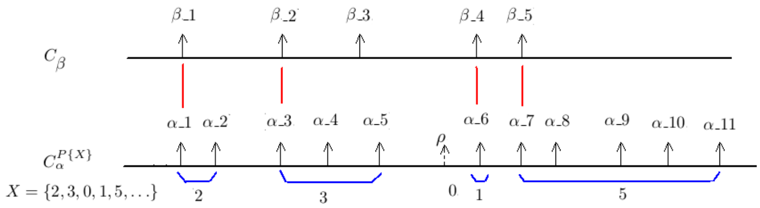

Fig. 6.6: Relation 4

occurrence of the i^{th} subsequence in the clock $C_\alpha^{P\{X\}}$ coincides with the k_j^{th} occurrence in the clock $C_{\eta[j]}(j \in [1, n])$.

The relation can be used to specify a flexible number of communications. In chapter 4, the use case 4.5 given in the page 64 has a fix number of cars ($car1$ and $car2$) that communicate with $car0$. However, in reality, the number of cars may change, for example, in the first cycle, there are 2 cars that communicate with $car0$; then in the second cycle, it may change to 3 cars that communicate with $car0$. It is very complicated to specify this situation without using the relations R5.

Fig.6.7 shows us an example with a table in which we deduce the occurrence relations and a figure that illustrates the relations. Let us take relation "$\alpha\_7 = \eta_{[1]}\_5$" as an example. At the 5^{th} cycle ($i = 5$), we have $N(\rho) = 1$, $x_5 = 5$. Since $j = 1$, we can calculate the index of α by the formula $\sum_{i=0}^{i-1} x_{(i-1)} + j = x_0 + x_1 + x_2 + x_3 + x_4 + j = 0 + 2 + 3 + 0 + 1 + 1 = 7$, and the index of $\eta_{[1]}$ by the formula

j	Relation	$\frac{i=1}{x_1=2}$	$\frac{i=2}{x_2=3}$	$\frac{i=3}{x_3=0}$	$\frac{i=4}{x_4=1}$	$\frac{i=5}{x_5=5}$	...
$j=1$	$\alpha\_(\sum_{i=0}^{i-1} x_i + 1) = \eta_{[1]}\_k_1$	$\alpha\_1 = \eta_{[1]}\_1$	$\alpha\_3 = \eta_{[1]}\_2$	$--$	$\alpha\_6 = \eta_{[1]}\_4$	$\alpha\_7 = \eta_{[1]}\_5$	...
$j=2$	$\alpha\_(\sum_{i=0}^{i-1} x_i + 2) = \eta_{[2]}\_k_2$	$\alpha\_2 = \eta_{[2]}\_1$	$\alpha\_4 = \eta_{[2]}\_2$	$--$	$--$	$\alpha\_8 = \eta_{[2]}\_3$	...
$j=3$	$\alpha\_(\sum_{i=0}^{i-1} x_i + 3) = \eta_{[3]}\_k_3$	$--$	$\alpha\_5 = \eta_{[3]}\_1$	$--$	$--$	$\alpha\_9 = \eta_{[3]}\_2$	...
$j=4$	$\alpha\_(\sum_{i=0}^{i-1} x_i + 4) = \eta_{[4]}\_k_4$	$--$	$--$	$--$	$--$	$\alpha\_10 = \eta_{[4]}\_1$	...
$j=5$	$\alpha\_(\sum_{i=0}^{i-1} x_i + 5) = \eta_{[5]}\_k_5$	$--$	$--$	$--$	$--$	$\alpha\_11 = \eta_{[5]}\_1$	...
...	...	...					

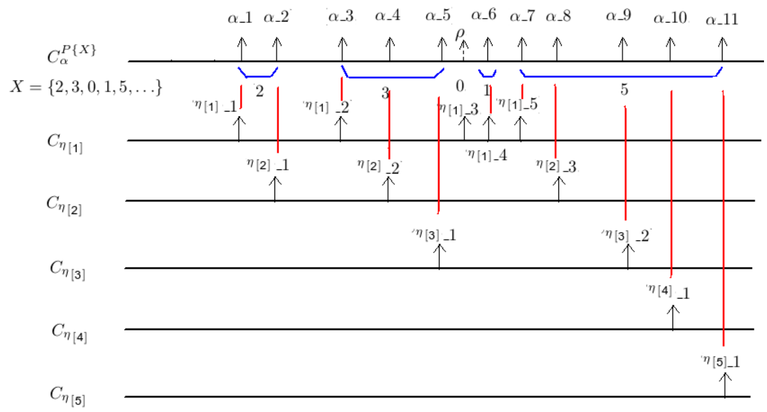

Fig. 6.7: Relation 5

$$k_j = N(\rho) + \sum_{i=0}^{i-1} x_{(i-1)} + j - (\sum_{j' \neq j, j'=1}^{n} k'_j) = 1 + 6 + 1 - (2 + 1) = 5.$$

6.2.3 Partition Clock Property

The design of clock partition helps us to flexibly group the timed-action occurrences. In consequence, we are able to flexibly build relations between the occurrences of two clocks. It alleviates our workload on constructing clock filters and results in an easier and more flexible way to build timed specifications. Hereafter, in the theorem 5, we prove that the relation on partition clocks can be expressed by a set of filtered clocks. This theorem tells that the precedence and coincidence relations can also apply to the partition clocks and keeps the correction of the relation properties in the section 4.2.2 of the chapter 4. Therefore, the theorem 1 from the chapter 4 also applies to the partition clocks.

Theorem 5 Let C_β be a normal clock that has not been partitioned and $C_\eta[n]$ be a set of normal clocks. Let $C_\alpha^{P\{X\}}$ (resp. $C_\gamma^{P\{Y\}}$) be a partition clock with $X = \{x_i\}$ ($i \in \mathbb{N}, x_0 = 0$) (resp. $Y = \{y_i\}$ ($y_0 = 0$)). Given a set of relations $\{C_\alpha^{P\{X\}} \prec C_\beta, C_\beta \prec C_\alpha^{P\{X\}}, C_\alpha^{P\{X\}} \prec C_\gamma^{P\{Y\}}, C_\alpha^{P\{X\} \triangleright k} = C_\beta, C_\alpha^{P\{X\}} = C_\eta[n]\}$. We say that these clock relations can be expressed by a set of filtered clocks as shown in the following cases:

- **Case1:** the relation $C_\alpha^{P\{X\}} \prec C_\beta$ can be substituted by $C_\alpha^{\{\sum_{j=1}^{i} x_j\}_{i\in\mathbb{N}}} \prec C_\beta^{\{i|x_i \neq 0\}_{i\in\mathbb{N}}}$;
- **Case2:** the relation $C_\beta \prec C_\alpha^{P\{X\}}$ can be substituted by $C_\beta^{\{i|x_i\neq 0\}_{i\in\mathbb{N}}} \prec C_\alpha^{\{1+\sum_{j=0}^{i-1} x_j\}_{i\in\mathbb{N}}}$;
- **Case3:** the relation $C_\alpha^{P\{X\}} \prec C_\gamma^{P\{Y\}}$ can be substituted by $C_\alpha^{\{\sum_{j=1}^{i} x_j | y_i \neq 0\}_{i\in\mathbb{N}}} \prec C_\gamma^{\{1+\sum_{j=0}^{i-1} y_j | x_i \neq 0\}_{i\in\mathbb{N}}}$;
- **Case4:** the relation $C_\alpha^{P\{X\}\triangleright k} = C_\beta$ can be substituted by $C_\alpha^{\{k+\sum_{j=0}^{i-1} x_j\}_{i\in\mathbb{N}}} = C_\beta^{\{i|x_i\neq 0 \wedge k \leq x_i\}}$;
- **Case5:** the relation $C_\alpha^{P\{X\}} = C_\eta[n]$ can be substituted by a set of relations $\{C_\alpha^{\{1+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}} = C_{\eta[1]}^{\{k_1|x_i\neq 0\wedge 1\leq x_i\}}, C_\alpha^{\{2+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}} = C_{\eta[2]}^{\{k_2|x_i\neq 0\wedge 2\leq x_i\}}, \ldots, C_\alpha^{\{n+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}} = C_{\eta[n]}^{\{k_n|x_i\neq 0\wedge n\leq x_i\}}\}$;

Proof. Let us analyze these clock relations case by case.

- For the **Case1**, the semantics of the relation $C_\alpha^{P\{X\}} \prec C_\beta$ is $[\![C_\alpha^{P\{X\}} \prec C_\beta]\!] = \forall i$, if $x_i \neq 0$, then $\alpha\_(\sum_{j=1}^{i} x_j) \prec \beta\_i$. According to the definitions of precedence and clock filtering, we can construct two filtered clocks

$C_\alpha^{\{\sum_{j=1}^{i} x_j\}_{i\in\mathbb{N}}}$ and $C_\beta^{\{i|x_i\neq 0\}_{i\in\mathbb{N}}}$ such that the semantics of $[\![C_\alpha^{\{\sum_{j=1}^{i} x_j\}_{i\in\mathbb{N}}} \prec C_\beta^{\{i|x_i\neq 0\}_{i\in\mathbb{N}}}]\!] = \forall i$, if $x_i \neq 0$, then $\alpha\_(\sum_{j=1}^{i} x_j) \prec \beta\_i$. So the relation $C_\alpha^{P\{X\}} \prec C_\beta$ can be substituted by a precedence relation on the two filtered clocks.

- For the **Case2**, the semantics of the relation $C_\beta \prec C_\alpha^{P\{X\}}$ is $[\![C_\beta \prec C_\alpha^{P\{X\}}]\!] = \forall i$, if $x_i \neq 0$, then $\beta\_i \prec \alpha\_(1 + \sum_{j=0}^{i-1} x_j)$. According to the definitions of precedence and clock filtering, we can construct two filtered clocks $C_\alpha^{\{1+\sum_{j=0}^{i-1} x_j\}_{i\in\mathbb{N}}}$ and $C_\beta^{\{i|x_i\neq 0\}_{i\in\mathbb{N}}}$ such that the semantics of $[\![C_\beta^{\{i|x_i\neq 0\}_{i\in\mathbb{N}}} \prec C_\alpha^{\{1+\sum_{j=0}^{i-1} x_j\}_{i\in\mathbb{N}}}]\!] = \forall i$, if $x_i \neq 0$, then $\beta\_i \prec \alpha\_(1 + \sum_{i=0}^{i-1} x_{(i-1)})$. So the relation $C_\beta \prec C_\alpha^{P\{X\}}$ can be substituted by a precedence relation on the two filtered clocks.

- For the **Case3**, the semantics of the relation $C_\alpha^{P\{X\}} \prec C_\gamma^{P\{Y\}}$ is $[\![C_\alpha^{P\{X\}} \prec C_\gamma^{P\{Y\}}]\!] = \forall i$, if $x_i \neq 0, y_i \neq 0$, then $\alpha\_(\sum_{j=1}^{i} x_j) \prec \gamma\_(1 + \sum_{j=0}^{i-1} y_j)$. We can construct two filtered clocks $C_\alpha^{\{\sum_{j=1}^{i} x_j|y_i\neq 0\}_{i\in\mathbb{N}}}$ and $C_\gamma^{\{1+\sum_{j=0}^{i-1} y_j|x_i\neq 0\}_{i\in\mathbb{N}}}$ such that $C_\alpha^{\{\sum_{j=1}^{i} x_j|y_i\neq 0\}_{i\in\mathbb{N}}} \prec C_\gamma^{\{1+\sum_{j=0}^{i-1} y_j|x_i\neq 0\}_{i\in\mathbb{N}}}$ has the same semantics as $C_\alpha^{P\{X\}} \prec C_\gamma^{P\{Y\}}$.

- For the **Case4**, the semantics of the relation $C_\alpha^{P\{X\}\rhd k} = C_\beta$ is $[\![C_\alpha^{P\{X\}\rhd k} = C_\beta]\!] = [\![C_\beta = C_\alpha^{P\{X\}\rhd k}]\!] = \forall i$, if $x_i \neq 0 \wedge k \leq x_i$, then $\alpha\_(k+\sum_{j=0}^{i-1} x_j) = \beta\_i$ (k is a fixed natural number). We can construct two filtered clocks $C_\alpha^{\{k+\sum_{j=0}^{i-1} x_j\}_{i\in\mathbb{N}}}$ and $C_\beta^{\{i|x_i\neq 0\wedge k\leq x_i\}}$ such that the semantics of clock relation $[\![C_\alpha^{\{k+\sum_{j=0}^{i-1} x_j\}_{i\in\mathbb{N}}} = C_\beta^{\{i|x_i\neq 0\wedge k\leq x_i\}}]\!] = \forall i$, if $x_i \neq 0 \wedge k \leq x_i$, then $\alpha\_(k + \sum_{j=0}^{i-1} x_j) = \beta\_i$.

- For the **Case5**, the semantics of the relation $C_\alpha^{P\{X\}} = C_\eta[n]$ is $[\![C_\alpha^{P\{X\}} = C_\eta[n]]\!] = \forall i, j \leq x_i$, if $x_i \neq 0$, then $\alpha\_(\sum_{r=0}^{i-1} x_r + j) = \eta_{[j]}\_k_j$ $(j \in [1, n])$.
  We can construct a set of filtered clocks $\{C_\alpha^{\{1+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}}, C_\alpha^{\{2+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}}, \dots, C_\alpha^{\{n+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}}\}$ and $\{C_{\eta[1]}^{\{k_1|x_i\neq 0\wedge 1\leq x_i\}}, C_{\eta[2]}^{\{k_2|x_i\neq 0\wedge 2\leq x_i\}}, \dots, C_{\eta[n]}^{\{k_n|x_i\neq 0\wedge n\leq x_i\}}\}$. Then we can build a set of coincidence relations on these filtered clocks as follows:
  $[\![C_\alpha^{\{1+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}} = C_{\eta[1]}^{\{k_1|x_i\neq 0\wedge 1\leq x_i\}}]\!] = \forall i$, if $x_i \neq 0 \wedge 1 \leq x_i$, then $\alpha\_(\sum_{r=0}^{i-1} x_r + 1) = \eta_{[1]}\_k_1$,
  $[\![C_\alpha^{\{2+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}} = C_{\eta[2]}^{\{k_2|x_i\neq 0\wedge 2\leq x_i\}}]\!] = \forall i$, if $x_i \neq 0 \wedge 2 \leq x_i$, then $\alpha\_(\sum_{r=0}^{i-1} x_r + 2) = \eta_{[2]}\_k_2$
  ...
  $[\![C_\alpha^{\{n+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}} = C_{\eta[n]}^{\{k_n|x_i\neq 0\wedge n\leq x_i\}}]\!] = \forall i$, if $x_i \neq 0 \wedge n \leq x_i$, then $\alpha\_(\sum_{r=0}^{i-1} x_r + n) = \eta_{[n]}\_k_n$
  under the condition $\sum_{j=1}^{n} k_j = N(\rho) + \sum_{r=0}^{i-1} x_r + j$. From the set of semantics we can see that the set of coincidence filtered clocks has the same semantics as the relation $C_\alpha^{P\{X\}\triangleright k} = C_\beta$. Therefore, the relation can be substituted by a set of relations $\{C_\alpha^{\{1+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}} = C_{\eta[1]}^{\{k_1|x_i\neq 0\wedge 1\leq x_i\}}$, $C_\alpha^{\{2+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}} = C_{\eta[2]}^{\{k_2|x_i\neq 0\wedge 2\leq x_i\}}, \dots, C_\alpha^{\{n+\sum_{r=0}^{i-1} x_r\}_{i\in\mathbb{N}}} = C_{\eta[n]}^{\{k_n|x_i\neq 0\wedge n\leq x_i\}}\}$.

□

6.3 Clock Union

Here, we define a clock union operator "$\dot{+}$" that is able to create a new clock from two different clocks. In CCSL [7], a simple version of clock union is defined as $[\![c_1 + c_2]\!] = (c_1 \vee c_2)$ (c_1 and c_2 are two logical clocks) in the sense that the union clock expression $c_1 + c_2$ ticks whenever c1 or c2

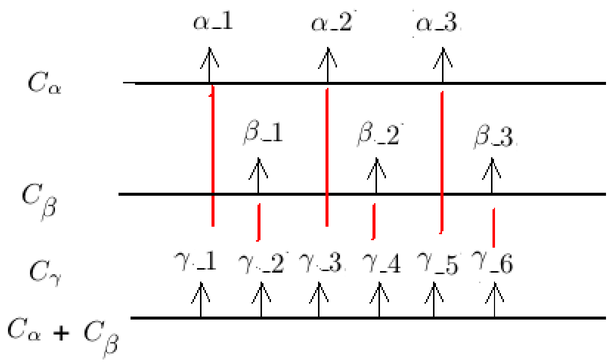

Fig. 6.8: clock union

ticks. In our thesis, we extend the clock union of CCSL by adding exclusion constraints between clock c_1 and c_2. Therefore, our clock union is typical for uniting the clocks that are forbidden to coincide. We mainly use the operator for specifying the branches in the transition systems in order to simplify the expressions of timed specifications. We give the definition of the Clock Union operator as follows.

Definition 33 (Clock Union on two clocks) Let C_α and C_β be two logical clocks. Clock C_γ is the union of the two clocks (denoted as $C_\alpha \dotplus C_\beta$). We say that clock C_γ can tick if

- the clocks C_α and C_β are exclusive ($C_\alpha \sharp C_\beta$);
- either C_α or C_β ticks.

Formally, $[\![C_\gamma]\!] = [\![C_\alpha \dotplus C_\beta]\!] = (\gamma\_k = \alpha\_i \vee \beta\_j) \wedge (k = i + j)(i, j, k \in \mathbb{N})$.

From the definition, we know that the clocks C_α and C_β cannot tick at the same time. In other words, only one clock can tick each time. Fig. 6.8 demonstrates a simple example of the clock union. Furthermore, we can build the union of a set of clocks $\mathcal{C}$. The definition below defines a clock union on n clocks.

Definition 34 (Clock Union on a set of clocks) Let clock C_γ be a new clock generated by the union of a finite set of logical clocks $\{C_i\}(i \in [1, n], n \in \mathbb{N})$. (denoted as $\dotplus\{C_i\}$). We say that clock C_γ can tick if

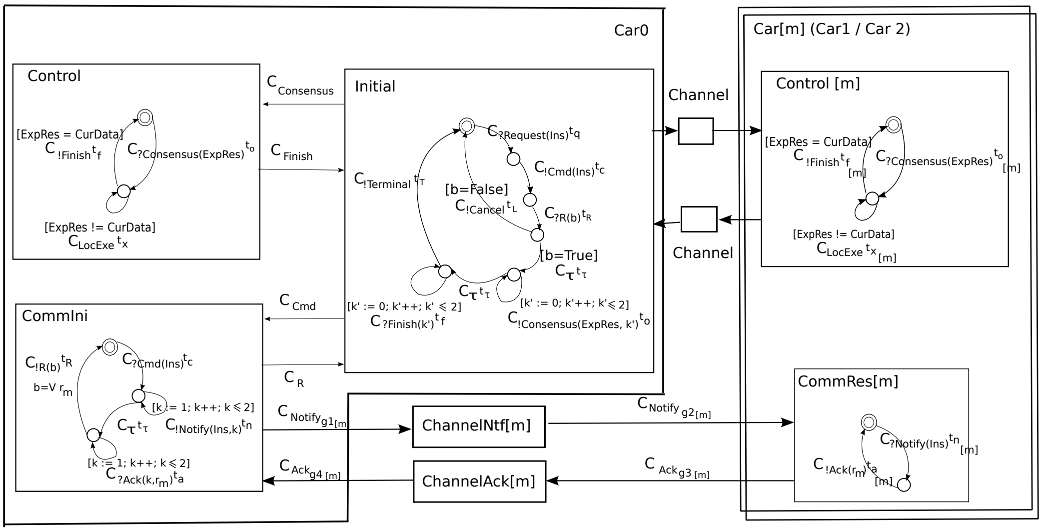

Fig. 6.9: Timed-pNets: Communication Behaviour Model of Cars Insertion Scenario

- $\forall i$, the clocks C_i are exclusive among them ($C_1 \sharp C_2 \sharp \ldots \sharp C_i \sharp \ldots \sharp C_n$);
- whenever C_i ticks.

Formally, $[\![C_\gamma]\!] = [\![\dotplus\{C_i\}]\!] = (C_\gamma[k] = C_1[j_1] \vee \ldots \vee C_i[j_i] \vee \ldots \vee C_n[j_n]) \wedge (k = \sum_{i=1}^{n} j_i)(k, j_i \in \mathbb{N})$

The operator is commutative and associative. Furthermore, since the operator is used to build a new logical clock by means of building a union of clocks, we actually do not change the definition of a logical clock. Therefore, the precedence and coincidence relations can still apply to the new generated clocks (built by clock union operators) and the properties in the section 4.2.2 in the page 63 are still hold for these new clocks.

6.4 Examples and Simulations

In this section, we take the components "Control" and "Initial" from the Fig. 4.5 in the page 64 as examples to represent how to specify them by using partition clocks and union operators. Here, we copy the figure to this section as shown in the Fig. 6.9.

Let C_f be a reference clock chosen for our simulation. For simplification, we set the C_f as a logical clock in which the occurrences appear periodically. we assume the delay bounds of all clocks are [1,2] (based on C_f), and require that the execution time of car0 moving to another lane must less than 5 steps of C_f after the clock $C_{?Consensus(ExpRes)^{t_o}}$ ticks. Here, we define two properties to be checked in our simulation:

- Safety Property: no clock relation conflict exists.
- Time Property: the clock $C_{!Finish^{t_f}}$ must tick within 5 steps (based on C_f) after the clock $C_{?Consensus(ExpRes)^{t_o}}$ ticks (formally, $C_{?Consensus(ExpRes)^{t_o}} \prec_{[1,5]} C_{!Finish^{t_f}}$).

We then input the timed specifications of these components into the TimeSquare tool to check the safety and time properties.

6.4.1 The Timed Specification of "Control" Component

From the Fig. 6.9 we can see that the execution of clock $C_{LocExe^{t_x}}$ depends on the guard "[ExpRes != CurData]". The system keeps on checking the guard. If it is satisfied, it triggers the clock $C_{LocExe^{t_x}}$ to tick once. Then the system checks the guard again. If it is still satisfied, the system keeps on triggering the clock to tick until the guard is not satisfied. So the clock $C_{LocExe^{t_x}}$ can be triggered many timed before the system transits to the next state. We use the partition clock $C^{P\{X\}}_{LocExe^{t_x}}$ to specify the situation. And the way of partition $X = \{x_i\}$ is built from the function below: $x_i = \sum_{j=u}^{v} a_j$, in which $u = i + \sum_{k=1}^{i-1} x_k$, $v = i + \sum_{k=1}^{i-1} x_k + x_i$, and $a_j = \begin{cases} 1 & \text{for} \quad ExpRes! = CurData \\ 0 & \text{for} \quad ExpRes = CurData \end{cases}$.

For example, if we get a sequence of $a_j = 1,1,1,0,1,1,0,0,1,1,0$, then we can calculate $x_1 = a_1 + a_2 + a_3 + a_4 = 3$. It stops at a_4 because $a_4 = 0$. Then $x_2 = a_5 + a_6 + a_7 = 2$ ($a_7 = 0$). Then $x_3 = a_8 = 0$. Then

6.1 : Calculate the Way of Partition X

j	1	2	3	4	5	6	7	8	9	10	11
a_j	1	1	1	0	1	1	0	0	1	1	0
i	1				2			3	4		
x_i	3				2			0	2		

$x_4 = a_9 + a_{10} + a_{11} = 2$ ($a_{11} = 0$). The result is shown in the table 6.1. In the end we have $X = \{x_i\} = \{3, 2, 0, 2\}$. Furthermore, according to the Fig.4.5, we can get the timed specification of the "Control "component as: $C_{?Consensus(ExpRes)^{t_o}} \prec C^{P\{X\}}_{LocExe^{t_x}} \prec C_{!Finish^{t_f}} \prec C^{\Delta(1)}_{?Consensus(ExpRes)^{t_o}}$ in which $X = \{x_i\}(x_i \in \mathbb{N})$.

6.4.2 Timed Specification of "Initial" Component

The "Initial" component in the Fig. 4.5 includes a branch. It tells that after clock $C_{?R^{t_R}}$, either clock $C_{\tau^{t_\tau}}$ or clock $C_{!Cancel^{t_L}}$ ticks. In this case we use the clock union operator to specify the relation as $C_{?R(b)^{t_R}} \prec C_{\tau^{t_\tau}} \dotplus C_{!Cancel^{t_L}}$. Similarly, when finishing a cycle, a clock union $C_{!Terminal^{t_T}} \dotplus C_{!Cancel^{t_L}}$ precedes $C^{\Delta(1)}_{?Request(Ins)^{t_q}}$. Besides, from the guard "$[k' = 0, k' + +, k' \leq 2]$" we can see that the selfloop on clock $C_{!Consensus(ExpRes,k')^{t_o}}$ and $C_{?Finish(k')^{t_f}}$ execute 3 times. So we have the relation $C^{\{3s-2\}_{s\in\mathbb{N}}}_{!Consensus(ExpRes,k')^{t_o}} \prec C^{\{3s-1\}_{s\in\mathbb{N}}}_{!Consensus(ExpRes,k')^{t_o}} \prec C^{\{3s\}_{s\in\mathbb{N}}}_{!Consensus(ExpRes,k')^{t_o}}$ and $C^{\{3s-2\}_{s\in\mathbb{N}}}_{?Finish(k')^{t_f}} \prec C^{\{3s-1\}_{s\in\mathbb{N}}}_{?Finish(k')^{t_f}} \prec C^{\{3s\}_{s\in\mathbb{N}}}_{?Finish(k')^{t_f}}$. The timed specification of this component is as follows:

TS of "Initial" Component:

$$C_{?Request(Ins)^{t_q}} \prec C_{!Cmd(Ins)^{t_c}} \prec C_{?R(b)^{t_R}} \prec C_{\tau^{t_\tau}} \dotplus C_{!Cancel^{t_L}} \prec C_{\tau^{t_\tau}} \prec C^{\{3s-2\}_{s\in\mathbb{N}}}_{!Consensus(ExpRes,k')^{t_o}} \prec C^{\{3s-1\}_{s\in\mathbb{N}}}_{!Consensus(ExpRes,k')^{t_o}} \prec C^{\{3s\}_{s\in\mathbb{N}}}_{!Consensus(ExpRes,k')^{t_o}} \prec C'_{\tau^{t_\tau}} \prec C^{\{3s-2\}_{s\in\mathbb{N}}}_{?Finish(k')^{t_f}} \prec C^{\{3s-1\}_{s\in\mathbb{N}}}_{?Finish(k')^{t_f}} \prec C^{\{3s\}_{s\in\mathbb{N}}}_{?Finish(k')^{t_f}} \prec C_{!Terminal^{t_T}};$$

$$C_{!Terminal^{t_T}} \dotplus C_{!Cancel^{t_L}} \prec C^{\Delta(1)}_{?Request(Ins)^{t_q}}$$

Actually, we can use a partition clock $C^{P\{Y\}}_{!Consensus(ExpRes,k')^{t_o}}$ and $C^{P\{Z\}}_{?Finish(k')^{t_o}}$

($Y = Z = \{3, 3, 3, \ldots\}$) to substitute a set of clocks for the two loops. In the end, the timed specification can be simplified as:

The simple version of "Initial" Component:

$C_{?Request(Ins)^{t_q}} \prec C_{!Cmd(Ins)^{t_c}} \prec C_{?R(b)^{t_R}} \prec C_{\tau^{t_\tau}} \dotplus C_{!Cancel^{t_L}} \prec C_{\tau^{t_\tau}} \prec C^{P\{Y\}}_{!Consensus(ExpRes,k')^{t_o}} \prec C'_{\tau^{t_\tau}} \prec C^{P\{Z\}}_{?Finish(k')^{t_o}} \prec C_{!Terminal^{t_T}}$;
$C_{!Terminal^{t_T}} \dotplus C_{!Cancel^{t_L}} \prec C^{\Delta(1)}_{?Request(Ins)^{t_q}}$

6.4.3 Simulate the "Control" component

Let us take "Control" component as an example. When doing simulation, assume $X = \{x_i\} = \{3, 2, 0, 2, \ldots\}$.[1] By the tool TimeSquare, we find that there is no clock relation conflict. However, when we check the time property, we found a time constraint conflict.

The reason for this conflict is that the action "LocExe" may repeat 3 times. Since we assumed that the delay of each clock is among [1,2], the delay between Clock $C_{?Consensus(ExpRes)^{t_o}}$ and $C_{!Finish^{t_f}}$ falls among [3, 6] $\not\subset$ [1,5]. So we need refine the specification to remove the conflict.

One solution is to design a new guard $[x_i \geq 3]$ and a new clock $C_{!Abortion^{t_b}}$ that communicates with "Initial" component. When the guard is satisfied (it means that the action "LocExe" executes more than 3 times), the clock $C_{!Abortion^{t_b}}$ ticks (see Fig. 6.10). We update the timed specification of the "Control" component as follows:

TS of updated Control component:

$C_{?Consensus(ExpRes)^{t_o}} \prec C^{P\{X\}}_{LocExe^{t_x}}$;
$C^{P\{X\}}_{LocExe^{t_x}} \prec [x_i < 3]C_{!Finish^{t_f}} \dotplus [x_i \geq 3]C_{!Abortion^{t_b}}$;
$C_{!Abortion^{t_b}} \dotplus C_{!Finish^{t_f}} \prec C^{\Delta(1)}_{?Consensus(ExpRes)^{t_o}}$

[1] the dots "..." mean that the number 2 is repeated. If we need repeat a set of numbers, we can use brace simbols. For example, {3, 2, 0, (2, 3), ...} means the set numbers (2,3) are repeated. So it has the same meaning as {3, 2, 0, 2, 3, 2, 3, (2, 3), ...}.

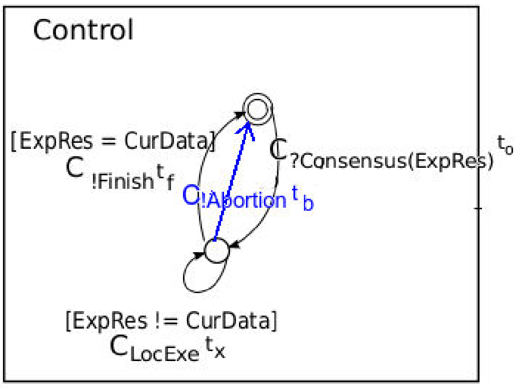

Fig. 6.10: Control Component Update

Notice that we do not put the guard [ExpRes != ExpData] as a guard in the timed specification. Because as we explained before, when we generate the partition clock $C^{P\{X\}}_{LocExe^{t_x}}$, we already use the guard [ExpRes != ExpData] to get the partiton X.

6.4.4 Simulate the "Initial" component

Since the "Abortion" signal will be sent from the "Control" component to the "Initial" component, we add a clock $C_{?Abortion^{t_b}}$ in the "Initial" component. The new timed-pLTS of the component is shown in Fig.6.11. We also update its timed specification as follows:

TS of updated Initial component:

$C_{\tau^{t_\tau}} \prec C^{P\{Y\}}_{!Consensus(ExpRes,k')^{t_o}}$; $(Y = \{3,3,3,\dots\})$

$C^{P\{Y\}}_{!Consensus(ExpRes,k')^{t_o}} \prec C^{P\{Z\}_{z_i \in [0,3]}}_{?Finish(k')}$; $(Y = Z = \{3,3,3,\dots\})$

$C^{P\{Z\}_{z_i \in [0,3]}}_{?Finish(k')} \prec [z_i = 3]C_{!Terminal^{t_T}} \dotplus [z_i < 3]C_{?Abortion^{t_b}}$;

$C_{?Abortion^{t_b}} \prec C_{!Abortion^{t_b}}$;

$C_{!Terminal^{t_T}} \dotplus C_{!Abortion^{t_b}} \dotplus C_{!Cancel^{t_L}} \prec C^{\Delta(1)}_{?Request(Ins)^{t_q}}$;

After simulating again the corrected component, we found both proper-

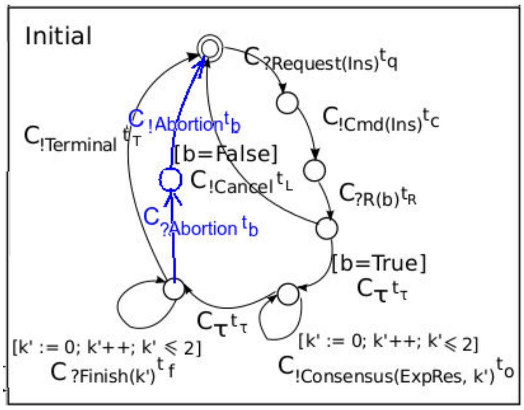

Fig. 6.11: Initial Component Update

ties are satisfied.

6.5 Conclusion

In this chapter, we defined clock partition and clock union to easy the building of timed specifications. The simulation of applying them to the "Control" and "Initial" components illustrated the advantages: making the timed specifications easier to understand and providing users a flexible way to specify complicated situations. The extensions are conservation in the sense that they preserve the theorem 1 and properties in the section 4.2.2.

In the next chapter, we will model the full "car inserting" use case and represent how we build a hierarchical timed-pNets. System properties (e.g. safety, latency properties) will be designed for the system. We will use the TimeSquare tool to check these properties in each layer. Besides, corrections of the use case model will be discussed when these properties are not satisfied.

Chapter 7 Full Use Case

In this chapter, a full use case is represented to demonstrate how we build a timed-pNets model and check its safety and time properties. We start with a full scenario of the car inserting use case in the section 1.5.2 of the page 23. Then we design five properties that we are interested in. We represent the procedure of building time-pNets model including the structure designing of the model. Since timed-pNets have a hierarchical structure, we build and simulate the model from bottom to top. In each layer we use the TimeSquare tool to check the properties that are related to this layer. Refinements are proposed if the properties are not satisfied. Furthermore, we design some advanced simulations like communicating with undetermined number of cars. In the end, we conclude our works.

7.1 Use Case

7.1.1 Background of ITS

An Intelligent Transportation System (ITS) is an application integrated the technologies of communication, control and information processing. All elements of the transportation system, including the vehicles, the infrastructures, and the drivers or users, interact dynamically among them. The aim of ITS is to improve real time decision making, thereby improving the efficiency of the entire transport system. In ITS, vehicles and infrastructures are equipped with sensors and actuators. They communicate with each other to update physical information and accomplish remote controlling. Currently, Research and Innovative Technology Administration (RITA) [91] in U.S. Department of Transportation has started research work on it to achieve a vision of national transportation by feature a connected transportation environment among vehicles, infrastructures and passengers' portable devices. It raises the importance of real-time communications among these distributed nodes since the data out of date would make big mistakes even sometimes could lead to a car accidence. For example, the late delivery of global traffic information to cars may result to a wrong guiding for cars to choose their best way. Moreover, the late information exchange among cars may cause a car accidence especially when they cannot see each other at cross.

Two communication safety applications are considered in ITS: vehicle to infrastructure (V2I) communications and vehicle to vehicle (V2V) communications. In the two applications, the vehicles are allowed to access network resources (e.g. MB-Portal, A-Class-Online, smart webmove, ...), and the back-end infrastructures are able to retrieve information (e.g. diagnostics data) from the vehicles. Vehicles and infrastructures share and exchange information and sensor data among each other. We took a use case mainly from vehicle to vehicle application to build a timed-pNets model and analyze its properties. We call the use case as Car Inserting. It describes that $car0$ can change its current lane and insert between other two cars after getting

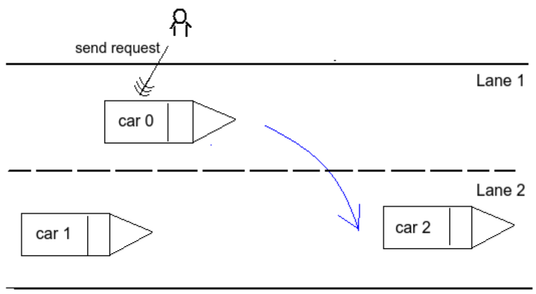

Fig. 7.1: Car Insertion

an "insert" request from human beings or other smart devices, as shown in the Fig. 7.1. Before inserting, $car0$ sends a request to other two cars to ask if it can execute the inserting action. If one of the two cars does not agree, the inserting action will be aborted. If both cars agree to let $car0$ insert between them, then $car0$ starts to change its lane and to insert between the two cars. The next section represents us the scenarios and requirements.

7.1.2 Car Inserting Use Case Scenario

In this section, we list detailed scenarios and timed requirements. These requirements are used to check the time constraint conflicts and latency properties. We import into our model a reference clock C_f in which the timed-action occurs periodically. All the timed requirements are designed based on the reference clock. We separate the use case into two phases: agreement phase and execution phase. For the agreement phase, the scenario is as follows:

- $car0$ gets a change-lane request (e.g. from a human user);
- $car0$ sends "notify" requests to $car1$ and $car2$ to get an agreement, and the time delay from getting a change-lane request to sending a "notify" is no more than 3 time units(based on C_f);
- $car1$ (resp. $car2$) acknowledges $car0$ "yes" or "no" within 10 time units(based on C_f);

- after sending "notify", $car0$ collects all results from $car1$ and $car2$;
- If both $car1$ and $car2$ answer "yes", $car0$ signals the consensus to $car1$ and $car2$ and then goes to the execution phase scenario,
- otherwise $car0$ aborts the procedure.

The scenario of the execution phase is:

- $car1$ slows down and/or $car2$ speeds up to leave more space between them for $car0$, and this execution must finish within 5 units (based on C_f);
- Meanwhile, $car0$ changes its direction and moves to lane2;
- $car0$ notifies the end of the procedure with a "finish" signal.

In the use case, for simplification, the delay bounds of those clocks that we do not specified in the scenario are set as $[1, 2]$.

7.1.3 Properties

Here we design some properties in which the time units of these properties are based on the reference clock C_f. The properties P1 and P2 are meta-properties in the sense that they do not need to be encoded and can be checked directly by the TimeSquare tool. The other three properties need to be encoded with CCSL format (detailed information can be found in the section 7.3).

(P1.) Safety Property: no logical clock relation conflict exists.

(P2.) Safety Property: system clock relations satisfy the following relation requirements: 1) the change-lane requests happen before sending notifications; 2) the sending notifications happen before getting acknowledgements; 3) the changing lane execution actions happen before sending the "finish" signal.

(P3.) Safety Property: no time constraint conflicts.

(P4.) Latency Property: assume that the network communication delay is less than 10 time units, then the latency from sending a notification to finishing collecting all acknowledgements is no more than 30 time units.

(P5.) Latency Property: the latency from $car0$ getting change-lane requests to sending "Terminal" signals is no more than 55 time units.

7.2 Build Timed-pNets Model

Here we represent the procedure of building timed-pNets as following steps:

(1.) According to the scenario of the use case, we design a component-based structure that includes timed-pNets holes and communications between holes;

(2.) Fill holes with timed-pLTSs and then transform to timed specifications.

7.2.1 System Structure

We use the component-based modelling approach to design the system structure. Since the communications between cars are asynchronous, we design channel components to build communications between cars. As shown in the Fig. 7.2, the top level (level 2) represents a coarse design of our system. Then we refine the system as shown in the level 1 in the Fig. 7.2. Since the communications in this level is synchronous, we directly build the communications by using synchronous vectors. In the leaf level (level 0, as shown with green circles), we represent the timed specifications of those components.

For simplification, we directly represent the structure with all levels as shown in the Fig. 7.3. In this structure, on-board car systems are modeled by

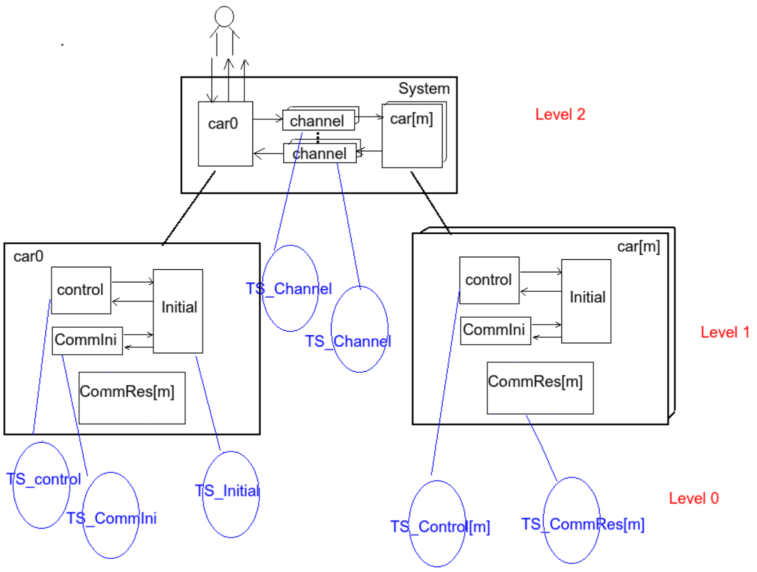

Fig. 7.2: Tree Structure of Use Case

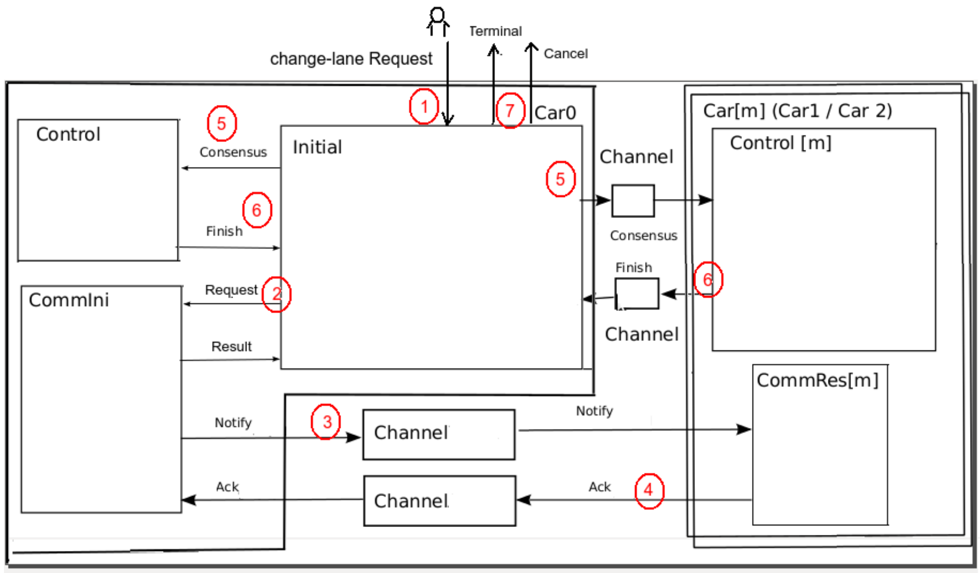

Fig. 7.3: The Component-based Structure of Car Inserting Use Case

several components including "Initial", "CommIni", "CommRes", "Control", etc. In the figure we only show the components that participate in the protocol. The change-lane requests are received by the "Initial" component. Then the request triggers "CommIni" component that takes charge of the communication part. The component "CommIni" sends "Notify" signals to the component "CommRes" of other cars and waits for the "ack" signals from them. Then the "CommIni" transmits a communication result to the "Initial" component. According to the result, the component decides whether or not to send "Consensus" signals to other cars to execute their movements. In the end, "Initial" component sends "Terminal" or "Cancel" signals to users or drivers. The red numbers give the order of these actions.

7.2.2 Fill Holes

Then these holes in the Fig.7.3 are filled with timed-pLTSs as shown in the Fig. 7.4. These timed-pLTSs are then translated to the timed specifications before inputting to the TimeSquare to check the properties. Since we already discussed the timed specifications of these holes in the chapters 4 and 6. For simplification, here we directly represent the holes with their timed specification as shown in the Fig.7.5. Notice that the timed specifications are not the final version. They would be modified later if the properties we required are not satisfied.

7.3 Simulation

We use the TimeSquare tool to simulate and check our model. Since the timed-pNets have a tree structure, and the timed-pNets we designed for the "Car Inserting" use case has three levels, by analysing the properties designed in the section 7.1, we locate these properties in the levels where they would be checked. As shown in the Table 7.1, the symbol $\surd$ says that the properties of the columns should be checked in the level of their crossed rows. We start our simulation from the bottom level and check the properties that located

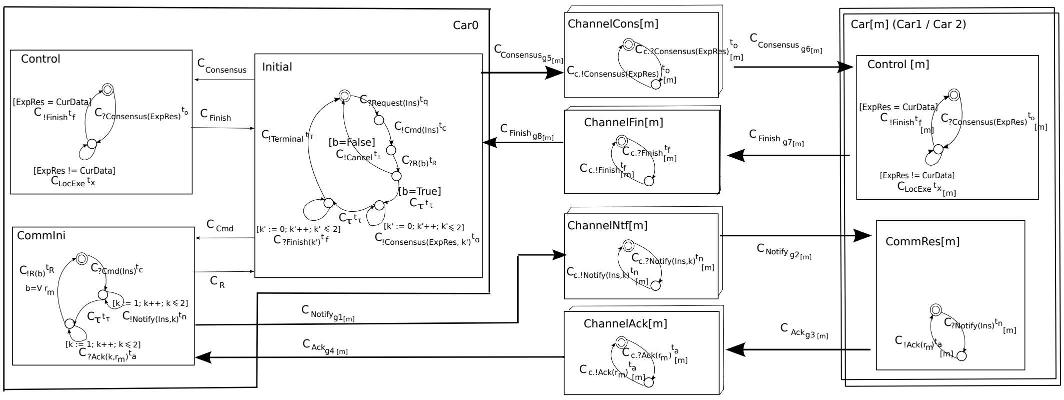

Fig. 7.4: Fill Timed-pLTS into Holes

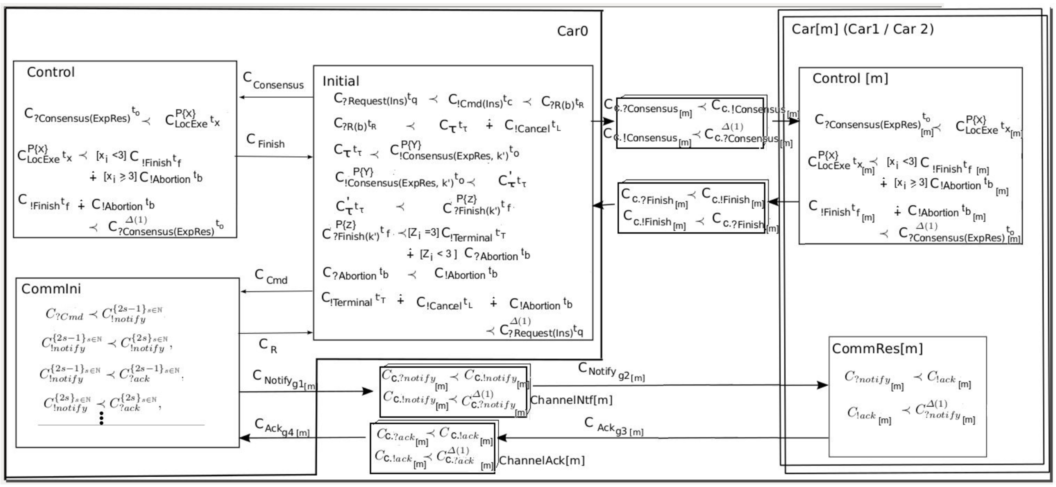

Fig. 7.5: Put Timed Specification into Holes

7.1 : Levels and properties

	P1	P2	P3	P4	P5
Level 0	√	-	√	-	-
Level 1	√	√	√	-	-
Level 2	√	√	√	√	√

in this level. Then we build an upper level and check the properties located in this level till we finish all the levels. Here we represent the procedure of simulating the model and checking the properties as follows:

(1.) simulate the leaf nodes of Fig. 7.2 and check if property P1 and P3 are satisfied;

(2.) build the middle level of Fig. 7.2 by composing these components into timed-pNets nodes, and check the properties P1, P2 and P3;

(3.) build the top level of Fig. 7.2 and check the properties P1 to P5.

Since the properties P1 and P3 are meta-properties, they should be checked in all the levels. Among these properties, some need to be translated to the form that can be accepted by the TimeSquare tool, some do not need. For example, usually the property P1 does not need to be translated, because the property can be check directly by the TimeSquare. Take another example, the property P3, needs to be translated. Actually it includes a set of properties. According to the scenario from the section 7.1, we list its sub properties as follows:

- (P3.1) $C_{?Consensus(ExpRes)^{t_o}} \prec_{[1,5]} C_{!Finish^{t_f}}$, this property is located in the "Control" component of the level 0;

- (P3.2) $Car0.C_{?Request(Ins)^{t_q}} \prec_{[1,3]} Car0.C_{!Notify(Ins,k)^{t_n}}$, this property is located in the "Car0" component of the level 1.

- (P3.3)$Car0.C_{!Notify(Ins,k)^{t_n}} \prec_{[1,10]} Car[m].C_{!Ack(r_m)^{t_a}}$, this property is located in the top level (level 2) between the components "Car0" and "Car[m]".

Usually a property can be represented as different forms when they are located in different levels. Let us take the property P3.1 as an example, when we discuss it in the level 0, the property is presented as $C_{?Consensus(ExpRes)^{t_o}} \prec_{[1,5]}$ $C_{!Finish^{t_f}}$. When we discus it in the level 1, it is presented as $Car0.C_{Consensus(ExpRes)^{t_o}} \prec_{[1,5]}$

$Car0.C_{Finish^{t_f}}$. Furthermore, when we discuss it in the level 2, it is presented as $C_{Consensus(ExpRes)^{t_o}}{}_{g6_{[m]}} \prec C_{Finish^{t_f}}{}_{g7_{[m]}}$. In other words, the property does not change, but the clocks related to the property may be changed in terms of the level they are located. For the other properties P2, P4 and P5, they also need to be encoded with the different clocks when we discuss them in different levels. The detailed difference will be represented in the sections 7.3.2 and 7.3.3.

7.3.1 Simulate the leaf level

In the leaf level, we check the components "Control", "Initial", "CommIni", "CommRes" and the channels ("ChannelNtf", "ChannelAck", etc.). We import the timed specifications (see the Fig.7.5) of these components into TimeSquare.

Translate properties

We encode the timed specifications of these leaves into the TimeSquare tool [41]. Then we check if the clock relation conflict (P1) and the time constraint conflict (P3) exist. If a conflict exists, we need to correct our model. We do not need to translate property P1. In the leaf level, only the property P3.1 is located in this level and it is translated to $C_{?Consensus(ExpRes)^{t_o}} \prec_{[1,5]} C_{!Finish^{t_f}}$.

Simulation Result

Since we already discussed the timed specifications of those leaves in the chapters 4 and 6. Here we directly give the results. According to the two chapters, to satisfy the property P1 and P3.1, we refined the system as shown in the Fig. 7.6.

After encoding these timed specifications of the refined system to the TimeSquare, we find that there is no clock relations conflict. And the latency property P3.1 is also satisfied.

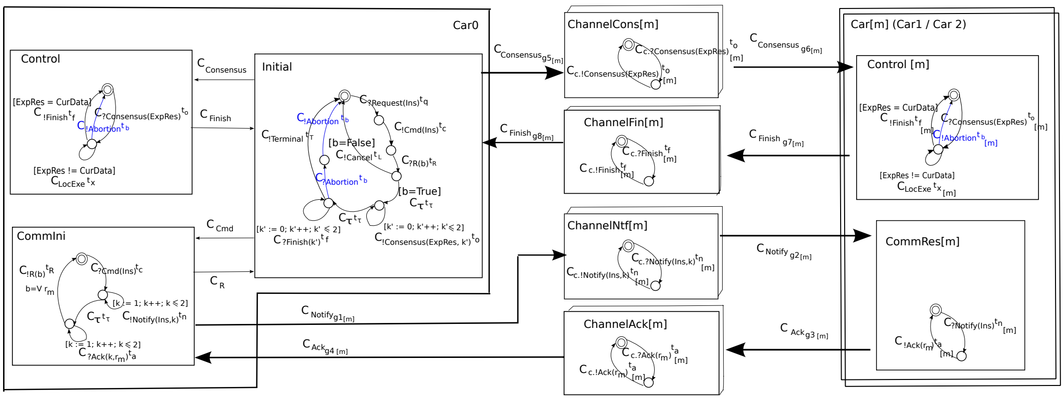

Fig. 7.6: The first Refinement

7.3.2 Simulate the middle level

Then we compose these well designed components and build the upper level timed-pNet nodes. From the Fig.7.2, we can see that car0 and car1 (resp.car2) are located in the middle level. These cars are composed by the components “Initial”, “Control”, “CommIni” and “CommRes”. Let us take car0 as an example to represent how to build its timed specification. Other timed specifications in this layer can also be built by the same way.

Timed-pNets formalization of car0

Fig.7.7 represents the timed-pNet node of car0 including the communications between its local components. Even though the component “CommRes” in car0 does not participate to the local communications, we still keep it here to represent a complete car model.

We formalize the node as follows:

- $P = \{k, Ins, m, r_m, b\}(m := 1, 2)$,
- $C_{GCar0} = \{Car0.C_{?Request(Ins)^{t_q}}, Car0.C_{Cmd(Ins)^{t_C}}, Car0.C_{R(b)^{t_R}}, Car0.C_{Consensus(ExpRes)^{t_o}}, Car0.C_{Finish^{t_f}}, Car0.C_{Abortion^{t_b}}, Car0.C_{!Notify(Ins,k)^{t_n}\,[m]}, Car0.C_{?Ack(k,r_m)^{t_a}\,[m]}, Car0.C_{!Consensus(ExpRes)^{t_o}}, Car0.C_{?Finish^{t_f}}, Car0.C_{?Abortion^{t_b}}, Car0.C_{!Terminal^{t_T}}, Car0.C_{!Cancel^{t_L}}, Car0.C_{!Abortion^{t_b}}\}$
- $J = \{CommIni, Control, Initial\}$

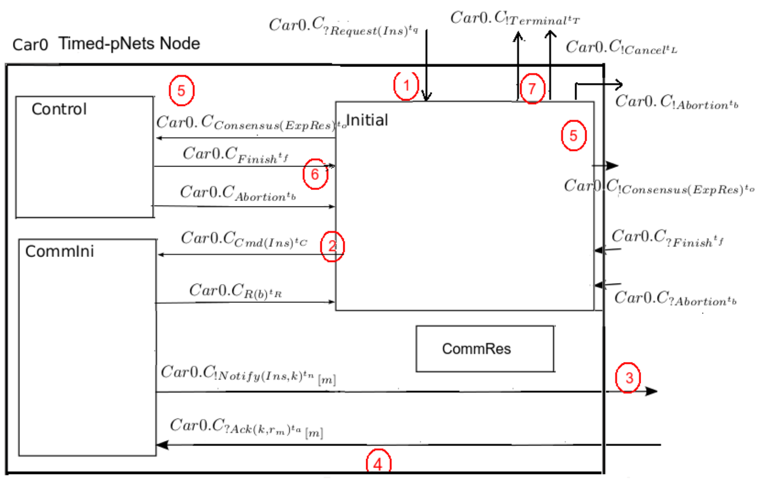

Fig. 7.7: Timed-pNets node of Car0

In the Fig.7.7, the global clocks of car0 are generated by a set of synchronous vectors as follows:

$V1 :< Initial.C_{!Cmd(Ins)^{t_C}}, -, CommIni.C_{?Cmd(Ins)^{t_C}} > \to Car0.C_{Cmd(Ins)^{t_C}}$

$V2 :< Initial.C_{?R(b)^{t_R}}, -CommIni.C_{!R(b)^{t_R}} > \to Car0.C_{R(b)^{t_R}}$

$V3 :< Initial.C^{P\{Y\}\triangleright 1}_{!Consensus(ExpRes)^{t_o}}, Control.C_{?Consensus(ExpRes)^{t_o}} \to$
$Car0.C^{P\{Y\}\triangleright 1}_{Consensus(ExpRes)^{t_o}}$

$V4 :< Initial.C^{P\{Z\}\triangleright 1}_{?Finish^{t_f}}, Control.C_{!Finish^{t_f}}, - > \to Car0.C^{P\{Z\}\triangleright 1}_{Finish^{t_f}}$

$V5 :< Initial.C_{?Abortion^{t_b}}, Control.C_{!Abortion^{t_b}}, - > \to Car0.C_{Abortion^{t_b}}$

$V6 :< -, -, CommIni.C_{!Notify(Ins,k)^{t_n}} > \to Car0.C_{!Notify(Ins,k)^{t_n}}$

$V7 :< -, -, CommIni.C_{?Ack(k,r_m)^{t_a}} > \to Car0.C_{?Ack(k,r_m)^{t_a}}$

$V8 :< Initial.C^{P\{Y\}\triangleright\{2,3\}}_{!Consensus(ExpRes)^{t_o}}, -, - > \to Car0.C^{P\{Y\}\triangleright\{2,3\}}_{!Consensus(ExpRes)^{t_o}}$

$V9 :< Initial.C^{P\{Z\}\triangleright\{2,3\}}_{?Finish^{t_f}}, -, - > \to Car0.C^{P\{Z\}\triangleright\{2,3\}}_{?Finish^{t_f}}$

$V10 :< Initial.C_{?Abortion^{t_b}}, -, - > \to Car0.C_{?Abortion^{t_b}}$

$V11 :< Initial.C_{!Terminal^{t_T}} > \to Car0.C_{!Terminal^{t_T}}$

$V12 :< Initial.C_{!Cancel^{t_L}} > \to Car0.C_{!Cancel^{t_L}}$

$V13 :< Initial.C_{!Abortion^{t_b}} > \to Car0.C_{!Abortion^{t_b}}$

$V14 :< Initial.C_{?Request(Ins)^{t_q}} > \to Car0.C_{?Request(Ins)^{t_q}}.$

Translate properties

To check the property P2, we need translate the property to the form that accepted by the TimeSquare, and then we run the tool to see if conflicts exist. The P2 is translated to:

- (1) $Car0.C_{?Request(Ins)^{tq}} \prec Car0.C_{!Notify(Ins,k)^{tn}}$
- (2) $Car0.C_{!Notify(Ins,k)^{tn}} \prec Car0.C_{?Ack(k,r_m)^{ta}}$
- (3) $Car0.C_{?Request(Ins)^{tq}} \prec Car0.C_{!Terminal^{t_T}}$.

For the property P3, we translate the P3.1 to $Car0.C_{Consensus(ExpRes)^{to}} \prec_{[1,5]} Car0.C_{Finish^{tf}}$ that is the relations based on the global clocks in the level 1. This property comes from the scenario in the section 7.1.2 which requires that the three cars would not take more than 5 time units (based on C_f) to finish moving themselves to their expect positions after receiving the consensus signals. Another property in the level 1 is P3.2. The property requires that the delay from getting a change-lane request to sending a "notify" is no more than 3 timed unites (based on C_f). Formally, it is written as $Car0.C_{?Request(Ins)^{tq}} \prec_{[1,3]} Car0.C_{!Notify(Ins,k)^{tn}}$.

Simulation result

We encode the timed specifications of those components and the synchronous vectors into TimeSquare to check the properties P1, P2, P3.1 and P3.2. For the property P1 and P2, the tool does not report any error. However, when we check P3.1, an error is reported.

The main reason is that the system cannot get "Finish" signals before finishing sending all "Consensus" signals. According to the assumption in the section 7.1.2 , the remote communication time (from sending "Consensus" to getting "Finish" signals) may take 20 time units (based on C_f). By waiting for the remote "Finish" signals from other cars, the delay between the clocks $C_{Consensus(ExpRes)^{to}}$ and $C_{Finish^{tf}}$ in car0 may be more than 5 units. It results to the failure of the property P3.1.

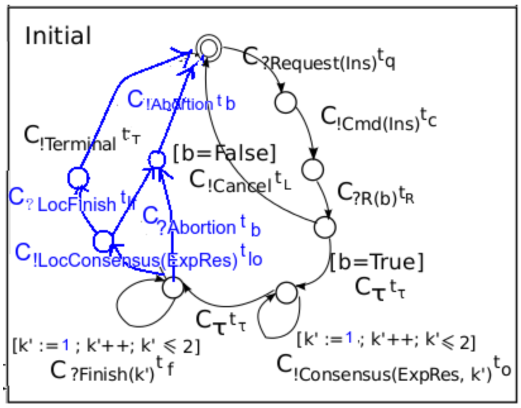

Fig. 7.8: Refined Initial Component

One solution is to let the local "Consensus" directly precede local "Finish" so that it can avoid waiting for the responses from other cars. Fig.7.8 represents the modified "Initial" component, in which we put the "local consensus" and "local finish" signals behind the remote ones. It guarantees that car1 and car2 have left enough space for car0 before it starts changing its lane. The timed specifications of the modified "Initial" component are listed as follows:

TS of refined Initial component:

$C_{\tau^{t_\tau}} \prec C^{P\{Y=\{2,2,2...\}\}}_{!Consensus(ExpRes,k')^{t_o}};$

$C^{P\{Y=\{2,2,2...\}\}}_{!Consensus(ExpRes,k')^{t_o}} \prec C^{P\{Z=\{z_i\}\}_{z_i\in[0,2]}}_{?Finish(k')};$

$C^{P\{Z=\{z_i\}\}_{z_i\in[0,2]}}_{?Finish(k')} \prec ([z_i = 2]C_{!LocConsensus(ExpRes)^{t_{lo}}} \dotplus [z_i < 2]C_{?Abortion^{t_b}});$

$C_{!LocConsensus(ExpRes)^{t_{lo}}} \prec C_{?LocFinish^{t_{lf}}} \dotplus C_{?Abortion^{t_b}};$

$C_{?Abortion^{t_b}} \prec C_{!Abortion^{t_b}}$

$C_{?LocFinish^{t_{lf}}} \prec C_{!Terminal^{t_T}};$

$C_{!Terminal^{t_T}} \dotplus C_{!Abortion^{t_b}} \dotplus C_{!Cancel^{t_L}} \prec C^{\Delta(1)}_{?Request(Ins)^{t_q}};$

Meanwhile, the synchronous vectors $V3$, $V4$, $V8$ and $V9$ should be updated as follows:

$V3 :< Initial.C_{!LocConsensus(ExpRes)^{t_o}}, Control.C_{?Consensus(ExpRes)^{t_o}}, - >\rightarrow$

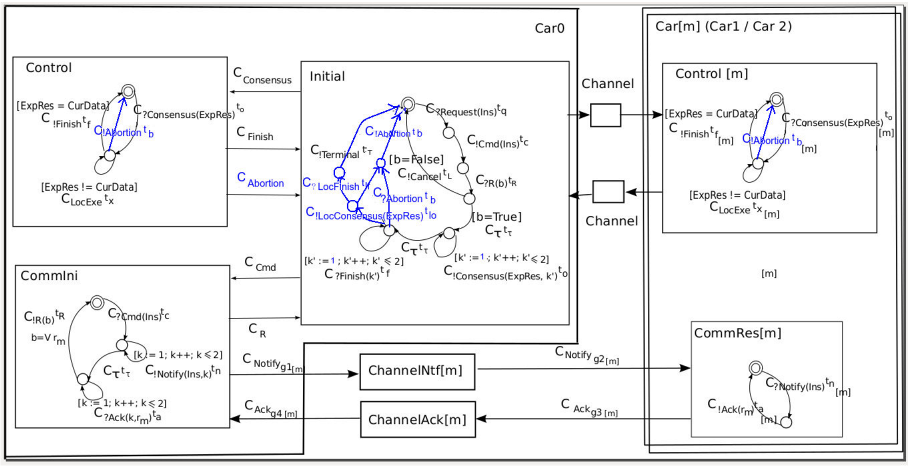

Fig. 7.9: Refined Version of Car Inserting Use Case

$$Car0.C_{LocConsensus(ExpRes)^{t_o}}$$
$$V4 :< Initial.C_{?LocFinish^{t_f}}, Control.C_{!Finish^{t_f}}, - >\rightarrow Car0.C_{LocFinish^{t_f}}$$
$$V8 :< Initial.C^{P\{Y\}\triangleright\{1,2\}}_{!Consensus(ExpRes)^{t_o}}, -, - >\rightarrow Car0.C^{P\{Y\}\triangleright\{1,2\}}_{!Consensus(ExpRes)^{t_o}}$$
$$V9 :< Initial.C^{P\{Z\}\triangleright\{1,2\}}_{?Finish^{t_f}}, -, - >\rightarrow Car0.C^{P\{Z\}\triangleright\{1,2\}}_{?Finish^{t_f}}$$

After updating the TSs of the Initial component and the synchronous vectors into the TimeSquare, we recheck the property P3.1 and it is satisfied. The Fig.7.9 demonstrates the refined version of our car inserting use case.

The global TS of Car0

According to the Theorem 1 in the page 87, we can generate the global timed specifications of car0. These logical clocks and clock relations in the global timed specifications can be observed from the top level. Furthermore, they are used to build the timed specifications of the top level. The global timed specifications TS_{car0} are generated as follows:

Global TS of car0:
$Car0.C_{?Request(Ins)^{t_q}} \prec Car0.C_{Cmd(Ins)^{t_c}} \prec Car0.C_{!Notify^{t_n}[m]} \prec Car0.C_{?Ack^{t_a}[m]} \prec Car0.C_R(b)^{t_R};$

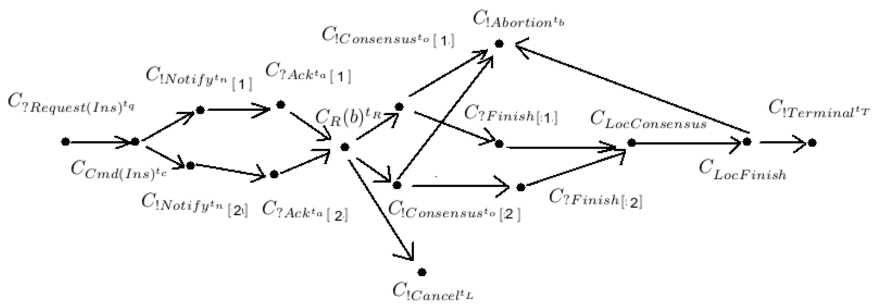

Fig. 7.10: Global Timed Specification Graph of car0

$$Car0.C_R(b)^{t_R} \prec Car0.C_{!Cancel^{t_L}} \dotplus Car0.C_{!Consensus^{t_o}[m]};$$
$$Car0.C_{!Consensus^{t_o}[m]} \prec Car0.C_{?Finish[m]} \dotplus Car0.C_{!Abortion^{t_b}};$$
$$Car0.C_{LocConsensus(ExpRes)^{t_{l^o}}} \prec Car0.C_{LocFinish^{t_{lf}}} \dotplus Car0.C_{!Abortion^{t_b}};$$
$$Car0.C_{?Finish[m]} \prec Car0.C_{LocConsensus(ExpRes)^{t_{l^o}}};$$
$$Car0.C_{LocFinish^{t_{lf}}} \prec Car0.C_{!Terminal^{t_T}};$$

The Fig.7.10 demonstrates the precedence relations of these global clocks of car0, in which the black points are clocks, and the arrow lines illustrate the relations. In the figure, for simplification, we omit the prefix "Car0".

7.3.3 Simulate the top level

The Formalization of the Node in the Top Level

In the end we build the node in the top level by composing the components car0, car[m](m=1,2) and channels as shown in the Fig. 7.11. The formalization of the node in the top level is as follows:

- $P = \{Ins, m, ExpRes\}(m := 1, 2)$,
- $C_G\text{-}\{top\} = \{C_{?Request(Ins)^{t_q}}{}_{g11}, C_{Notify^{t_n}}{}_{g1_{[m]}}, C_{Notify^{t_n}}{}_{g2_{[m]}}, C_{Ack^{t_a}}{}_{g3_{[m]}}, C_{Ack^{t_a}}{}_{g4_{[m]}}, C_{Consensus(ExpRes)^{t_o}}{}_{g5_{[m]}}, C_{Consensus(ExpRes)^{t_o}}{}_{g6_{[m]}}, C_{Finish^{t_f}}{}_{g7_{[m]}}, C_{Finish^{t_f}}{}_{g8_{[m]}}, C_{Abortion^{t_b}}{}_{g9_{[m]}}, C_{Abortion^{t_b}}{}_{g10_{[m]}}, C_{!Terminal^{t_T}}{}_{g12}, C_{!Cancel^{t_L}}{}_{g13}, C_{!Abortion^{t_b}}{}_{g14}\}$

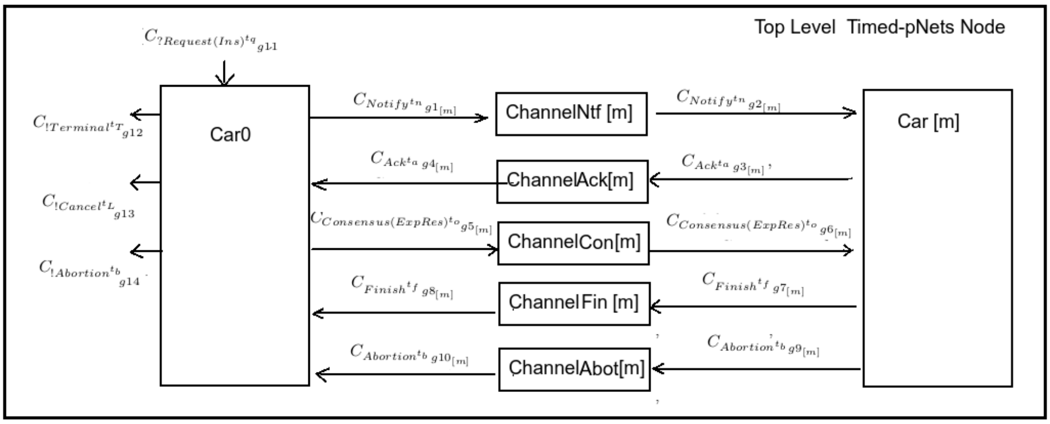

Fig. 7.11: Top Level Timed-pNets Node

- $J = \{Car0, Car[m], ChannelNtf[m], ChannelAck[m], ChannelCon[m], ChannelFin[m], ChannelAbot[m]\}$

We list part of the synchronous vectors that build the communications between those components as follows. These synchronous vectors generate the system global clocks $C_G\text{-}\{top\}$.

$V1_g :< Car0.C_{?Request(Ins)^{tq}}, -, -, -, -, -, - >\rightarrow C_{?Request(Ins)^{tq}}{}_{g11}$

$V2_g :< Car0.C^{\{2s-1\}}_{!Notify(Ins,k)^{tn}}, ChannelNtf[1].C_{?Notify(Ins,k)^{tn}}{}_{[1]}, -, -, -, -, - >\rightarrow C_{Notify^{tn}}{}_{g1_{[1]}}$

$V3_g :< Car0.C^{\{2s\}}_{!Notify(Ins,k)^{tn}}, ChannelNtf[2].C_{?Notify(Ins,k)^{tn}}{}_{[1]}, -, -, -, -, - >\rightarrow C_{Notify^{tn}}{}_{g1_{[2]}}$

$V4_g :< -, ChannelNtf[m].C_{!Notify(Ins,k)^{tn}}{}_{[m]}, -, -, -, -, Car[m].C_{?Notify(Ins,k)^{tn}}{}_{[m]} >\rightarrow C_{Notify^{tn}}{}_{g2_{[m]}}$

$V5_g :< Car0.C^{\{2s-1\}}_{?Ack(k,r_m)^{ta}}, -, ChannelAck[1].C_{!Ack(k,r_m)^{ta}}{}_{[1]}, -, -, -, - >\rightarrow C_{Ack^{ta}}{}_{g3_{[1]}}$

$V6_g :< Car0.C^{\{2s\}}_{?Ack(k,r_m)^{ta}}, -, ChannelAck[2].C_{!Ack(k,r_m)^{ta}}{}_{[2]}, -, -, -, - >\rightarrow C_{Ack^{ta}}{}_{g3_{[2]}}$

$V7_g :< Car0.C^{P\{X\}\triangleright\{1,2\}}_{!Consensus(ExpRes)^{to}}, -, -, ChannelCon[m].C_{?Consensus(ExpRes)^{to}}{}_{[m]}, -, -, - >\rightarrow C_{Consensus(ExpRes)^{to}}{}_{g5_{[m]}};$

...

Translate properties

In the top level, property P1 does not need to be translated. The other properties are translated with the global clocks of this level. For property P2, we translate it to:

- (1) $C_{?Request(Ins)^{tq}}{}_{g11} \prec C_{Notify^{tn}}{}_{g1_{[m]}}$
- (2) $C_{Notify^{tn}}{}_{g1_{[m]}} \prec C_{Ack^{ta}}{}_{g3_{[m]}}$
- (3) $C_{?Request(Ins)^{tq}}{}_{g11} \prec C_{!Terminal^{tT}}{}_{g12}$.

The property P3.1, P3.2, P3.3, P4 and P5 are translated to:

P3.1: $C_{Consensus(ExpRes)^{to}}{}_{g6_{[m]}} \prec_{[1,5]} C_{Finish^{tf}}{}_{g7_{[m]}}$

P3.2: $C_{?Request(Ins)^{tq}}{}_{g11} \prec_{[1,3]} C_{N}otify^{tn}{}_{g1_{[m]}}$

P3.3: $C_{N}otify^{tn}{}_{g1_{[m]}} \prec_{[1,10]} C_{Ack^{ta}}{}_{g3_{[m]}}$

P4: $C_{Notify^{tn}}{}_{g1_{[m]}} \prec_{[1,30]} C_{Ack^{ta}}{}_{g4_{[m]}}$

P5: $C_{?Request(Ins)^{tq}}{}_{g11} \prec_{[1,55]} C_{!Terminal^{tT}}{}_{g12}$

Simulation result

We encode the timed specifications of car0, car1, car2 and the channels into the TimeSquare to check the five properties from P1 to P5. Properties P1 to P4 are all satisfied. An error is reported when checking the property P5. It says that the whole procedure from receiving a request to finish changing the lane cannot be finished in 55 time units. To solve the issue, we either reduce the communication latency between cars or relax the real-time requirement. By analysing, when we reduce the communication latency to 5 time units, the property P5 can be satisfied.

7.4 Other Simulations

In this section, we discuss a more complex situation that is car0 communicates with more than two cars. Then we investigate if these properties still can be satisfied.

7.4.1 Car0 communicates with m cars ($m > 2$)

In this experiment, our aim is to check at most how many cars the system can afford such that these properties are still satisfied.

Simplify Specification

In order to simplify our simulation, we use the partition clocks $C^{P(U)}_{!Notify}$ and $C^{P(W)}_{?Ack}$ instead of $C_{!Notify}$ and $C_{?Ack}$ in the "CommIni" component. By modifying the assignment of U and W, we can easily change the number of cars that communicate with car0. For example, if we set $m = 3$, $U = \{3, 3, 3, \ldots\}$ and $W = \{3, 3, 3, \ldots\}$, then we can simulate the situation of communicating with 3 cars. Moreover, since we use partition clocks, we need to modify the synchronous vectors. In our use case, we combine $V2_g$ and $V3_g$ to $V2'_g$ as follows.

Original synchronous vectors:

$V2_g :< Car0.C^{\{2s-1\}}_{!Notify(Ins,k)^{tn}}, ChannelNtf[1].C_{?Notify(Ins,k)^{tn}}{}_{[1]},$
$-, -, -, -, - >\rightarrow C_{Notify^{tn}}{}_{g1_{[1]}}$
$V3_g :< Car0.C^{\{2s\}}_{!Notify(Ins,k)^{tn}}, ChannelNtf[2].C_{?Notify(Ins,k)^{tn}}{}_{[1]},$
$-, -, -, -, - >\rightarrow C_{Notify^{tn}}{}_{g1_{[2]}}$

To simplified vectors:

$V2'_g :< Car0.C^{P(U)}_{!Notify(Ins,k)^{tn}}, ChannelNtf[m].C_{?Notify(Ins,k)^{tn}}{}_{[m]},$

$-,-,-,-,->\rightarrow C_{Notify^{tn}} g1_{[m]}$

Similarly, we combine $V5_g$ and $V6_g$ to $V5'_g$ as follows.

Original synchronous vectors:

$V5_g :< Car0.C^{\{2s-1\}}_{?Ack(k,r_m)^{ta}}, -, ChannelAck[1].C_{!Ack(k,r_m)^{ta}}[1], -,-,-,->\rightarrow C_{Ack^{ta}} g3_{[1]}$

$V6_g :< Car0.C^{\{2s\}}_{?Ack(k,r_m)^{ta}}, -, ChannelAck[2].C_{!Ack(k,r_m)^{ta}}[2], -,-,-,->\rightarrow C_{Ack^{ta}} g3_{[2]}$

To simplified vectors:

$V5'_g :< Car0.C^{P(W)}_{?Ack(k,r_m)^{ta}}, -, ChannelAck[m].C_{!Ack(k,r_m)^{ta}}[m], -,-,-,->\rightarrow C_{Ack^{ta}} g3_{[m]}$

.

Simulation Result

We increase the number of cars one by one. First, we let car0 communicate with 2 cars. We found out that all our properties are satisfied. Then we increase one more car that communicates with car0 by setting $m = 3$, $U = \{3, 3, 3, \ldots\}$ and $W = \{3, 3, 3, \ldots\}$. We found out the property P5 cannot be satisfied. Then we increase one car more by setting $m = 4$, $U = \{4, 4, 4, \ldots\}$ and $W = \{4, 4, 4, \ldots\}$. We found out both P4 and P5 cannot be satisfied. We keep on increasing the number of cars. The table 7.2 shows us the results. From this table we can see that with the increasing number of cars, the safety property P1 and P2 can be satisfied. But the latency properties may not be satisfied.

7.2 : Simulation with Flexible Number of Cars

	P1	P2	P3	P4	P5
$m = 2$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
$m = 3$	$\surd$	$\surd$	$\surd$	$\surd$	$\times$
$m = 4$	$\surd$	$\surd$	$\surd$	$\times$	$\times$
$m = 5$	$\surd$	$\surd$	$\times$	$\times$	$\times$
$m = 50$	$\surd$	$\surd$	$\times$	$\times$	$\times$

7.5 Conclusion

In this chapter, we represented a full use case taken from ITS and represented how to build a timed-pNets semantic model for it. Our simulations were done layer by layer from bottom to top. In each layer, we checked its safety properties and time properties. The TimeSquare tool was used to check these properties. And we represented the detailed corrections when the properties were not satisfied. Besides, we have done the simulation when increasing the number of cars. From these simulations, we can see that our timed-pNets are flexible to compose components. By modeling components with timed specifications, we can take advantage of the TimeSquare tool to detect the system logical conflicts and check its latency properties.

Chapter 8 Conclusion

In this chapter, we present a summary of the thesis contributions, as well as corresponding limitations. Finally, we conclude the thesis work with a discussion of interesting directions for future work.

8.1 Summary and Conclusions

In this thesis, we have mainly focused on designing a semantic model that is able to specify timed-related communication behaviours of distributed systems with the requirements of addressing the following two goals:

- The timed model does not rely on a common global physical clock;
- Both synchronous and asynchronous communications are able to be specified.

To achieve the two goals, we have designed a novel timed model called Time-pNets that is able to specify and verify the time constrained communication behaviours of heterogeneous distributed systems. By taking advantage of the logical clock concept, the model can specify the relations (happen before or happen at the same time) of system behaviours without relying on a common physical clock.

The design of timed specifications helps us to flexibly specify the synchronous and asynchronous communications, as well as composing different components and building a hierarchical structure in a flexible and simple way. Thanks to the timed specification that paves the way to transform our system to CCSL, the TimeSquare tool can be used to check the system safety and time properties of timed-pNets systems.

The compatibility issues have been discussed in our thesis, which guarantee the correctness of refined timed specifications. We also designed algorithms to generate timed specifications from timed-pLTSs for building a hierarchical structure in a unify way. By introducing the concepts of reference clocks and visual timestamps, timed-pNets have the capability of measuring the delays between logical clocks, which allows for the time bound analysis, as well as the specification and verification of latency properties. Furthermore, in order to specify some complex situations, we designed partition clocks and clock union operators, by which a system can be specified in a simpler and more flexible way. Examples have been illustrated to show us the advantages:

to be adaptable to specify many complex cases, from undetermined numbers of cycles to unfixed car communications.

Finally, we have also validated the research results by using a typical use case taken from ITS. We described its timed-pNets model and checked by simulations the safety and correctness properties by using the TimeSquare tool.

In a conclusion, our model provides a simple and flexible way to model communications behaviours (synchronous and asynchronous) with time constraints without relying on physical clocks. This is one of the main advantages comparing to other current timed models. Moreover, our model is able to check the logical correctness and verify time properties of distributed systems. We believe that the timed-pNets are helpful for analyzing the time-related behaviours of distributed systems including cyber physical systems.

8.2 Future Work

As future work, there are several interesting directions.

- First, we can extend the current timed-pNets to durtion-pNets that are able to specify the system behaviours whose execution takes time.

  To realize this, we plan to define duration-events that are an extension of timed-actions by introducing execution time variables. Duration-events are expressed with the combinations of two instantaneous actions (a start action and an end action) with a precedence relation between them. For example, a car brake event can be described by a combination of 'start car brake' and 'end car brake' actions. In the duration-event, the start point action happens earlier than the end point action, except the case that the execution time equals to zero, which tells that the start point and the end points coincide. Thus we can say that actually timed-pNets is a specific case of duration-pNets in which the execution time of actions are zero.

Using the preliminary notions of duration-events, we then plan to build duration-clocks and classify various types of timing constraints on these logical duration-clocks. In these duration-clocks, the timed-action occurrences include execution time. It means that these clock occurrences are not just ticks. They are a sequence of intervals. Therefore, we will redefine the relations on these duration-clocks.

Similar to the timed-pNets, in order to build a hierarchical structure of duration-pNets, we will investigate the timed specifications and discuss the time bounds on these duration clocks.

- Second, the possibility of using model checking tools to verify the timed-pNets models is another interesting research direction.

  In the thesis, we use simulation to check the properties. As we know that simulation is an automated analysis technique that is being used extensively and effectively in industry. However, simulation is usually non-exhaustive. It means that not all possible behaviours are checked for conformance with the requirements. In other words, it can expose erroneous behaviour, but the absence of bad behaviours cannot be guaranteed. Compared to simulation, formal verification is a technique that aims to cover all the behaviours of a system. If the timed-pNets model is able to translate to timed-automata, we are able to use the model checking tool UPPAAL [16] to verify our model. As we know form the paper [82], a technique of transforming MARTE/CCSL behaviours into timed-automata has been proposed. It helps to address the issue of verify CCSL-based behaviours in the UPPAAL tool. Since our model is based on timed specification in which the clock relations are mainly taken from CCSL, it is possible for us to transform basic logical clocks and relations in our model into timed automata by using the technique in the paper [82]. However, for the partition clocks we should define a clear way to transform to timed-automata. Till now, this point is still not clear and be worthy to investigate.

- Third, the specification formalism are an important aspect for supporting the model checking in the future work.

  To be able to use model checking tool, it requires a well-defined semantics for our timed-pNets model. In this thesis, we are able to specify systems by timed specifications that are initially generated from the timed-pLTSs. Even though we have developed algorithms to generate timed specifications from timed-pLTSs, we do not have tools to automatically generate these timed specifications and it is not clear how many situations cannot be covered. Therefore, developing a tool to automatically generate the timed specifications and proposing a schema to cover all possible situations are good direction to reinforce our results.

- Fourth, the system refinement and compatibility are an interesting point for the future work.

  The compatibility should always be conserved in timed-pNets. We did some work on proposing the definition of compatibility and checking it by using TimeSquare tools. However, we discussed little about model refinement and compatibility that is conserved in the refinements. In the further work, the system refinement should be discussed and proper methods (e.g. model checking) should be proposed to verify the compatibility.

- Fifth, developing a tool to automatically generate TimeSquare input files is worthy point for the future work.

  For simulation, we encode our timed specifications and properties into TimeSquare tool after translating to the form that the tool accepted. We have no tool to generate them automatically and no schema to check if the translation is correct. For a small use case as we proposed in the thesis it is easy to be checked, but for a big use case, the tool is necessary.

- Last but not least, to apply our model on a large use case to investigate the scalability of our model is an important task in the future work.

  A natural question would arise about the scalability and the efficacy of the proposed analysis approach on larger case studies. Currently, our use case covers three layers structure with around fifty logical clocks. In the future work, the model should be applied to larger case studies and the scalability of timed-pNets model should be checked.

Chapter 9 附录：论文综述（中文版）

随着网络技术的不断发展，物联网/物理信息融合系统成为目前研究和发展的热点。 一个典型的例子是智能交通系统（ITS)。 通信作为信息交换的媒介，已成为物联网研究的核心问题之一。 在智能交通系统中，车辆可以与服务中心沟通（V2I），告知其他车辆他们的存在以便于车辆的安全监控和安全驾驶; 另外车辆和车辆之间也可以通信（V2V），从而提高交通的安全性，避免恶性交通事故的发生。

我们面临的科学问题是如何在这些分布的车辆间建立通信模型，研究通信的时实性。 这就需要我们显式的处理时间信息，描述分布式系统行为的时间特性并讨论其时间属性。 然而分布式系统中所有的处理器都各自的执行自己的任务，并且系统中没有一个公共的物理时间基准来核定这些处理器的时间变量。 因此该系统所执行的事件的逻辑顺利可能与各个事件在各自的处理器中所排列的时钟顺序不同。 例如，我们所期望的系统的逻辑顺序应该是发送信息的事件发生在接受信息的事件之前。 然而，如果发送器和接收器的时钟不同步，则有可能会存在接受信息的事件的时刻比发送信息的事件的时刻早的情况。

为了解决这一问题，我们充分利用分布式系统中事件发生的的逻辑顺序不会改变这一性质来为分布式系统建立时间约束模型。 我们用逻辑时钟关系来描述系统行为的因果依赖关系，并给系统的行为赋予不同的逻辑时间值， 从而来推断行为之间的因果关系或排除一些不可能的情况， 比如，一个“后期”的动作不能影响一个“早期”的行动。

逻辑时钟最早由Leslie Lamport在1978年提出用于描述分布式系统的执行情况。 逻辑时钟已经被证明在为并发系统建立各种不同抽象层次的

模型起着非常大的作用。 时钟约束规范语言(CCSL)使用逻辑时钟作为第一元素并支持一组(逻辑)时钟来描述系统的时间行为。 在CCSL中，逻辑时钟定义为一个动作重复出现的序列。 一个逻辑的时钟的每次”滴答“不像一个真实时钟那样等距，而是用于记录动作发生的先后顺序的值。 根据CCSL模型的启发（CCSL的具体的技术背景知识在论文第15页中给予详细的介绍),我们定义时钟关系来约束指定的时钟之间的逻辑限制。

我们尝试把逻辑时钟引入一个没有时间变量的模型pNets（具体定义见论文第18页）中来建立一个新的时间模型timed-pNets。 pNets是一个用于为分布式系统建模并进行验证的形式化模型。 它利用标记转换系统（LTS）描述系统的通信行为，并给LTS引入了参数用于更加洁的描述动态拓扑结构。 pNets支持种类繁多的通信机制，以至于能够足够灵活的处理大量的分布式编程。 参数化和层次结构也使得pNets结构紧凑，和程序结构相近，因此容易用组合的方式建模。 该参数化模型已经成功地用于为ProActive建模，并已被证明适合作为分布式系统的规范语言。

在该论文中，我们首先引入时间化动作（Timed-Actions）的概念(具体定义见论文第48页)。 然后，我们在时间化动作的基础上定义逻辑时钟(具体定义见论文第49页)。 每个逻辑时钟都是一个时间化动作的一组出现。逻辑时钟的一次“滴答”就表示时间化动作的一个出现（或执行）。

由于系统中的通信行为要么是同步或是异步的， 因此我们可以选择CCSL中的基本的时钟关系，例如”同时发生（e.g. $C_\alpha = C_\beta$）”和 ”优先关系（e.g. $C_\alpha \prec C_\beta$）”来定义逻辑时钟的同步和异步通信。 在此基础上，我们提出了系统的时间规范Timed Specification（具体定义见论文第61页），并用它来建立分布式系统的时间模型。

我们把时间规范（Timed Specification）引入pNets（参数化网络同步自动机） 来建立一个具有树型层次结构的时间规范框架结构如图9.1所示。其叶子节点是通过timed-pLTSs (具体定义9见论文第66页)表示的。 其非叶节点（称为timed-pNets节点，具体定义11见论文第68页） 是具有同步子网的行为的功能的节点（子网可以是叶子或非叶子节点）。 我们让所有的节点（叶子或非叶子节点）与时间规范（timed specification）相关联。 在这个框架结构中，上层的时间规范是它的下层子系统的抽象。 并且上层的时间规范可以由下层的子网中的时间规范推导而来。

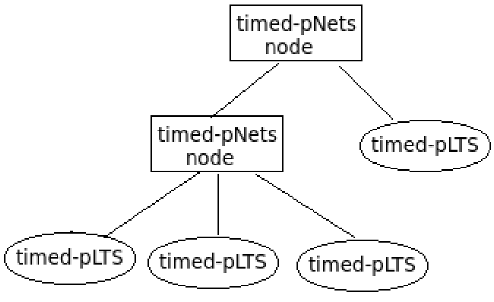

Fig. 9.1: Timed-pNets tree structure

Timed-pNets是用于描述和验证分布式系统中具有时间约束的通信行为的语义模型。 它不仅保持pNets模型的优点如层次结构，能够灵活的适应不同的编程结构和通信模式，而且它还具有其自身的新的优势。

通过引入时间规范，timed-pNets模型可以在不依赖公共的物理时钟的情况下描述系统的逻辑时间约束。 我们可以把timed-pLTSs和timed-pNets转换成时间规范（timed specification）， 并通过分析层次化的时间规范来分析我们的timed-pNets模型。在这种新的模型下， 低层次（同步的）组件中的逻辑时钟的关系能够由标签转换系统（LTS）来推导出。 通常的逻辑时钟是一个先验独立的。当不同的时钟建立了关系（例如同时或是优先关系）时候， 这些时钟变得相互依赖互相制约。一个时钟的关系描述了许多（可能是无穷的）时间实例的关系。 通过对多个时钟建立关系，这些时钟不再独立并且他们的时间实例也存在着偏序的关系。 这些偏序的时间实例构成了我们系统的时间规范（TSs）。

时间规范的引入也使得我们的模型能够灵活的设计系统。时间规范有其逻辑特性，要么由应用的设计者提供，要么从模型中推导出来。 这使得我们能够任意的使用至下而上的设计模式：详细的设计时间化的pLTSs并把它们以一个兼容的方式组装起来； 或采用自上而下的设计方式，构建抽象的timed-pNets，假设一些以孔的形式表现的时间规范，然后提供一些相容的实现来填充这些孔。 我们提出了相应的理论和方法（具体定理1, 定理2, 定理3分别见论文第87页，90页和93页）来检查系统的兼容性，这帮助我们验证了系统的正确性和安全性。

由于该模型不依赖于共同的物理时钟，因此来自不同的子网的时钟延迟是不可比较的。这给我们建立多层模型带来了困难。 为了解决这个问题，我们引入了参考时钟和虚拟时间戳的概念。参考时钟是用户指定的时钟， 可以为一般的精密计时时钟，也可以为逻辑时钟。一旦用户指定了一个参考时钟后， 所有其他的时钟依据该参考时钟得到其相应的虚拟时间戳，这将帮助我们比较来自不同子网的时钟延迟并计算上层的时钟延迟。该论文讨论了如何从子网的延迟推导出上层抽象节点的延迟，并帮助我们分析时间约束冲突（具体定理见论文第111页）， 用于验证模型的延迟属性(例如截止时间，等待时间等)。这对于一个时间模型来说是非常重要的方面。

时间规范的引入也给我们利用TimeSquare工具来检查系统的时间约束冲突铺平了道路。 由于使用时间规范，timed-pNets能够描述分布式系统的时间化的行为。 为了能够深入了解我们的模型，我们选择TimeSquare工具做模拟实验。 TimeSquare是用于分析时间模型的软件环境。 它能够根据时间规范显示由标准的VCD格式生成时间波形图(TimeSquare的更多信息介绍在论文的第17页加以详细介绍)。 该工具可以在冲突发现后产生错误报告。

简而言之，Timed-pNets模型提供了一种简单而灵活的方式来建立不依赖物理时钟的时间化的通信行为（同步和异步）模型。 该模型通过引入逻辑时钟巧妙的避开了使用系统的物理时钟，解决了分布式系统研究通信实时性的问题。 这一点是与其他目前存在的时间模型的主要区别。 此外，通过引用参考时钟以及虚拟时间戳，Timed-pNets模型不但能够检查出分布式系统逻辑的正确性， 也可以检查模型的时间约束冲突，并验证其时间属性。

该模型主要应用于物联网/物理信息融合系统中。例如在智能交通系统中，通过建立车辆之间的实时通信模型， 我们可以分析该系统的实时性能。这对于车辆的安全监控和安全驾驶，遏制交通拥堵并帮助驾驶员们减少出行延误， 提高综合交通运输效率起到了至关重要的作用。

本文主要章节如下:

- 第一章主要给出了我们研究的应用背景，研究动机，研究方法及其主要研究贡献。

- 第二章仔细调查了一些现有的时间模型,如时间自动机,时间Pteri网,MARTE和AADL等著名的实时建模系统。
- 第三章把逻辑时钟和时钟关系引入到pNets模型中，从而定义了一种新的语义模型, 该模型具有描述分布式系统的时间约束的能力。
- 第四章介绍了timed-pNets的通信行为语义模型。 该模型是在上一章的基础上提出的一个更新更全的改进版本。 在本节中我们引入了时间规范的概念，并讨论了如果建立层次化的timed-pNets模型。 此外我们还讨论了时钟规范的兼容性等问题。 该模型可以用来描述分布式系统的具有时间约束的通信模型,包括同步通信行为和异步通信行为。
- 第五章讨论了如何计算timed-pNets模型的延迟和延迟界限。 此外,我们定义了时间冲突的概念, 并提出检测时间冲突的方法。
- 第六章讨论了提出了时钟分区和时钟合并的概念，用来简化时间规范，并更加灵活的建立模型。
- 第七章我们用一个完整的用例来演示如何建立和完善一个timed-pNets模型, 并检查其安全性和实时性能。
- 第八章总结我们目前的工作,并展望今后的工作。

References

[1] Tesnim Abdellatif, Jacques Combaz, and Joseph Sifakis. Model-based implementation of real-time applications. In Proceedings of the tenth ACM international conference on Embedded software, pages 229–238. ACM, 2010.

[2] Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. ACM Transactions on Computer Systems (TOCS), 2(2):93–122, 1984.

[3] Rajeev Alur and David Dill. Automata for modeling real-time systems. In Automata, languages and programming, pages 322–335. Springer, 1990.

[4] Rajeev Alur and David L. Dill. A theory of timed automata. Theoretical Computer Science, 126(2):183 – 235, 1994.

[5] Rajeev Alur and Thomas A Henzinger. Reactive modules. Formal Methods in System Design, 15(1):7–48, 1999.

[6] Rabéa Ameur-Boulifa, Ludovic Henrio, Eric Madelaine, and Alexandra Savu. Behavioural Semantics for Asynchronous Components. Rapport de recherche RR-8167, INRIA, December 2012.

[7] Charles André. Syntax and Semantics of the Clock Constraint Specification Language (CCSL). Rapport de recherche RR-6925, INRIA, 2009.

[8] Charles André, Frédéric Mallet, Robert De Simone, et al. Time modeling in marte. In ECSI Forum on specification & Design Languages (FDL), pages 268–273, 2007.

[9] Charles André, Frédéric Mallet, Julien DeAntoni, et al. Vhdl observers for clock constraint checking. In Symposium on Industrial Embedded Systems, 2010.

[10] André Arnold and John Plaice. Finite transition systems: semantics of communicating systems. Prentice Hall International (UK) Ltd., 1994.

[11] Eugene Asarin, Oded Maler, and Amir Pnueli. On discretization of delays in timed automata and digital circuits, 1998.

[12] Felice Balarin. Hardware-software co-design of embedded systems: the POLIS approach. Springer, 1997.

[13] Tomás Barros, Rabéa Boulifa, Antonio Cansado, Ludovic Henrio, and Eric Madelaine. Behavioural models for distributed Fractal components. Annals of Telecommunications, 64(1–2), jan 2009. also Research Report INRIA RR-6491.

[14] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in BIP. In Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006), 11-15 September 2006, Pune, India, pages 3–12. IEEE Computer Society, 2006.

[15] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in bip. In Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on, pages 3–12. Ieee, 2006.

[16] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems.

In Proc. of Workshop on Verification and Control of Hybrid Systems III, LNCS 1066, pages 232–243. Springer–Verlag, October 1995.

[17] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Lectures on Concurrency and Petri Nets, pages 87–124. Springer, 2004.

[18] Saddek Bensalem, Marius Bozga, T-H Nguyen, and Joseph Sifakis. Compositional verification for component-based systems and application. IET software, 4(3):181–193, 2010.

[19] Albert Benveniste, Benoît Caillaud, and Paul Le Guernic. From synchrony to asynchrony. Springer, 1999.

[20] Albert Benveniste, Paul Le Guernic, and Christian Jacquemot. Synchronous programming with events and relations: the signal language and its semantics. Science of computer programming, 16(2):103–149, 1991.

[21] G Berry and E Sentovich. Embedding synchronous circuits in gals-based systems. In Sophia-Antipolis conference on Micro-Electronics (SAME 98), 1998.

[22] Gérard Berry. Real time programming: Special purpose or general purpose languages. 1989.

[23] Gérard Berry. The foundations of esterel. In Proof, language, and interaction, pages 425–454, 2000.

[24] Gérard Berry, Cyprien Nicolas, and Manuel Serrano. Hiphop: a synchronous reactive extension for hop. In Proceedings of the 1st ACM SIGPLAN international workshop on Programming language and systems technologies for internet clients, pages 49–56. ACM, 2011.

[25] Gérard Berry and Ellen Sentovich. Multiclock esterel. In Correct Hardware Design and Verification Methods, pages 110–125. Springer, 2001.

[26] Gérard Berry and Manuel Serrano. Hop and hiphop: Multitier web orchestration. In Distributed Computing and Internet Technology, pages 1–13. Springer, 2014.

[27] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time petri nets. IEEE transactions on software engineering, 17(3):259–273, 1991.

[28] Bernard Berthomieu*, P-O Ribet, and François Vernadat. The tool tina–construction of abstract state spaces for petri nets and time petri nets. International Journal of Production Research, 42(14):2741–2756, 2004.

[29] Conrad Bock. Sysml and uml 2 support for activity modeling. Systems Engineering, 9(2):160–186, 2006.

[30] Frédéric Boussinot and Robert De Simone. The esterel language. Proceedings of the IEEE, 79(9):1293–1304, 1991.

[31] Marius Bozga, Jean-Claude Fernandez, Alain Kerbrat, and Laurent Mounier. Protocol verification with the aldebaran toolset. International Journal on Software Tools for Technology Transfer (STTT), 1(1):166–183, 1997.

[32] Marius Bozga, Susanne Graf, Ileana Ober, Iulian Ober, and Joseph Sifakis. The if toolset. In Formal Methods for the Design of Real-Time Systems, pages 237–267. Springer, 2004.

[33] Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg. Proceedings of an Advanced Course on Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986-Part I. Springer-Verlag, 1986.

[34] Denis Caromel, Ludovic Henrio, and Bernard Paul Serpette. Asynchronous sequential processes. Information and Computation, Volume 207, Issue 4, 2008.

[35] Denis Caromel, Wilfried Klauser, and Julien Vayssiere. Towards seamless computing and metacomputing in java. Concurrency Practice and Experience, 10(11-13):1043–1061, 1998.

[36] Christos G Cassandras. Discrete event systems: modeling and performance analysis. 1993.

[37] Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg, and Bernadette Charron-Bost. On the impossibility of group membership. In Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing, pages 322–330. ACM, 1996.

[38] D. M. Chapiro. Globally-asynchronous locally-synchronous systems. PhD thesis, Stanford Univ., CA., October 1984.

[39] Yixiang Chen. Stec: A location-triggered specification language for real-time systems. In ISORC Workshops, pages 1–6. IEEE, 2012.

[40] Massimoliano Chiodo, Paolo Giusto, Attila Jurecska, Harry C Hsieh, Alberto Sangiovanni-Vincentelli, and Luciano Lavagno. Hardware-software codesign of embedded systems. Micro, IEEE, 14(4):26–36, 1994.

[41] Julien Deantoni and Frédéric Mallet. TimeSquare: Treat your Models with Logical Time. In Sebastian Nanz Carlo A. Furia, editor, TOOLS - 50th International Conference on Objects, Models, Components, Patterns - 2012, volume 7304 of Lecture Notes in Computer Science - LNCS, pages 34–41, Prague, Tchèque, République, May 2012. Czech Technical University in Prague, in co-operation with ETH Zurich, Springer.

[42] John Eidson, Edward A Lee, Slobodan Matic, Sanjit A Seshia, and Jia Zou. A time-centric model for cyber-physical applications. In Workshop

on Model Based Architecting and Construction of Embedded Systems (ACES-MB), pages 21–35, 2010.

[43] John Eidson, Edward A. Lee, Slobodan Matic, Sanjit A. Seshia, and Jia Zou. Distributed real-time software for cyber-physical systems. Proceedings of the IEEE (special issue on CPS), 100(1):45 – 59, January 2012.

[44] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (aadl): An introduction. Technical report, DTIC Document, 2006.

[45] Colin Fidge. Logical time in distributed computing systems. Computer, 24(8):28–33, 1991.

[46] Colin J Fidge. Timestamps in message-passing systems that preserve the partial ordering. In Proceedings of the 11th Australian Computer Science Conference, volume 10, pages 56–66, 1988.

[47] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM (JACM), 32(2):374–382, 1985.

[48] George S Fishman. Discrete-event simulation: modeling, programming, and analysis. Springer, 2001.

[49] Gerard Florin and Stéphane Natkin. Evaluation based upon stochastic petri nets of the maximum throughput of a full duplex protocol. In Application and Theory of Petri Nets, pages 280–288. Springer, 1982.

[50] Ning Ge, Marc Pantel, and Xavier Crégut. Time properties dedicated transformation from uml-marte activity to time transition system. ACM SIGSOFT Software Engineering Notes, 37(4):1–8, 2012.

[51] Gregor Gossler and Joseph Sifakis. Composition for component-based modeling. Science of Computer Programming, 55(1):161–183, 2005.

[52] Sei. open source aadl tool environment. http://www.sei.cmu.edu/dependability/tools/osate/.

[53] Axel Jantsch. Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation. Morgan Kaufmann, 2004.

[54] David R Jefferson. Virtual time. ACM Transactions on Programming Languages and Systems (TOPLAS), 7(3):404–425, 1985.

[55] M. Jersak. timing model and methodology for autosa. In Elektronik Automotive. Special issue AUTOSAR, 2007.

[56] Gabor Karsai, Janos Sztipanovits, Akos Ledeczi, and Ted Bapty. Model-integrated development of embedded software. Proceedings of the IEEE, 91(1):145–164, 2003.

[57] Jensen Kurt. Coloured petri nets: Basic concepts, analysis methods and practical use. EATCS Monographs on Theoretical Computer Science. 2nd edition, Berlin: Springer-Verlag, 1997.

[58] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7):558–565, 1978.

[59] Edward A Lee. Modeling concurrent real-time processes using discrete events. Annals of Software Engineering, 7(1-4):25–45, 1999.

[60] Edward A Lee. Model-driven development-from object-oriented design to actor-oriented design. In Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (aka The Monterey Workshop), Chicago. Citeseer, 2003.

[61] Edward A Lee and Eleftherios Matsikoudis. The semantics of dataflow with firing. From Semantics to Computer Science: Essays in memory of Gilles Kahn. Cambridge University Press, Cambridge, 2008.

[62] K Lee, John C Eidson, Hans Weibel, and Dirk Mohl. Ieee 1588-standard for a precision clock synchronization protocol for networked measurement and control systems. In Conference on IEEE, volume 1588, 2005.

[63] Barbara Liskov and Rivka Ladin. Highly available distributed services and fault-tolerant distributed garbage collection. In Proceedings of the fifth annual ACM symposium on Principles of distributed computing, pages 29–39. ACM, 1986.

[64] Frédéric Mallet. Clock constraint specification language: specifying clock constraints with uml/marte. Innovations in Systems and Software Engineering, 4(3):309–314, 2008.

[65] Frédéric Mallet, Charles André, and Julien DeAntoni. Executing aadl models with uml/marte. In Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on, pages 371–376. IEEE, 2009.

[66] Frédéric Mallet, M-A Peraldi-Frati, and Charles André. Marte ccsl to execute east-adl timing requirements. In Object/Component/Service-Oriented Real-Time Distributed Computing, 2009. ISORC'09. IEEE International Symposium on, pages 249–253. IEEE, 2009.

[67] Friedemann Mattern. Virtual time and global states of distributed systems. Parallel and Distributed Algorithms, 1(23):215–226, 1989.

[68] Philip M Merlin and David J Farber. Recoverability of communication protocols–implications of a theoretical study. Communications, IEEE Transactions on, 24(9):1036–1043, 1976.

[69] Philip Meir Merlin. A study of the recoverability of computing systems. 1974.

[70] David L Mills. Simple network time protocol (sntp) version 4 for ipv4, ipv6 and osi. 2006.

[71] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, et al. The open provenance model core specification (v1. 1). Future Generation Computer Systems, 27(6):743–756, 2011.

[72] Douglas Stott Parker Jr, Gerald J Popek, Gerard Rudisin, Allen Stoughton, Bruce J Walker, Evelyn Walton, Johanna M Chow, David Edwards, Stephen Kiser, and Charles Kline. Detection of mutual inconsistency in distributed systems. Software Engineering, IEEE Transactions on, (3):240–247, 1983.

[73] Michel Raynal. A distributed algorithm to prevent mutual drift between n logical clocks. Information Processing Letters, 24(3):199–202, 1987.

[74] Rami R Razouk and Charles V Phelps. Performance analysis using timed petri nets. In PSTV, volume 84, pages 561–576, 1984.

[75] Sunil K. Sarin and Nancy A. Lynch. Discarding obsolete information in a replicated database system. Software Engineering, IEEE Transactions on, (1):39–47, 1987.

[76] Frank B Schmuck. The use of efficient broadcast protocols in asynchronous distributed systems. Technical report, Cornell University, 1988.

[77] Manuel Serrano, Erick Gallesio, and Florian Loitsch. Hop: a language for programming the web 2. 0.

[78] Lui Sha, Sathish Gopalakrishnan, Xue Liu, and Qixin Wang. Cyber-physical systems: A new frontier. In Machine Learning in Cyber Trust, pages 3–13. Springer US, 2009.

[79] Joseph Sifakis. Use of petri nets for performance evaluation. Acta Cybern., 4:185–202, 1980.

[80] CCITT Specification. description language (sdl). ITU-T Recommendation, (100):11, 1993.

[81] Rob Strom and Shaula Yemini. Optimistic recovery in distributed systems. ACM Transactions on Computer Systems (TOCS), 3(3):204–226, 1985.

[82] Jagadish Suryadevara, Cristina Seceleanu, Frédéric Mallet, and Paul Pettersson. Verifying marte/ccsl mode behaviors using uppaal. In Software Engineering and Formal Methods, pages 1–15. Springer, 2013.

[83] V Valero Ruiz, David de Frutos Escrig, and F Cuartero Gomez. On non-decidability of reachability for timed-arc petri nets. In Petri Nets and Performance Models, 1999. Proceedings. The 8th International Workshop on, pages 188–196. IEEE, 1999.

[84] Wil MP van der Aalst. Interval timed coloured petri nets and their analysis. In Application and Theory of Petri Nets 1993, pages 453–472. Springer, 1993.

[85] Wil MP van der Aalst and Michiel A. Odijk. Analysis of railway stations by means of interval timed coloured petri nets. Real-time systems, 9(3):241–263, 1995.

[86] Kees M Van Hee. Information systems engineering: a formal approach. Cambridge University Press, 1994.

[87] KM Van Hee, LJ Somers, and M Voorhoeve. Executable specifications for distributed information systems. Falkenberg and P. Lindgreen, editors, Information System Concepts: An In-depth Analysis, pages 139–156, 1989.

[88] Geng Wu, Shilpa Talwar, Kerstin Johnsson, Nageen Himayat, and Kevin D Johnson. M2m: From mobile to embedded internet. Communications Magazine, IEEE, 49(4):36–43, 2011.

[89] Hengyang Wu, Yixiang Chen, and Min Zhang. On denotational semantics of spatial-temporal consistency language–stec. In Theoretical Aspects of Software Engineering (TASE), 2013 International Symposium on, pages 113–120. IEEE, 2013.

[90] Gene TJ Wuu and Arthur J Bernstein. Efficient solutions to the replicated log and dictionary problems. Operating systems review, 20(1):57–66, 1986.

[91] Intelligent transportation systems. http://www.its.dot.gov/research.htm.

[92] Modeling and analysis of real-time and embedded system. http://www.omgmarte.org/.

[93] Bernard P Zeigler, Herbert Praehofer, and Tag Gon Kim. Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems. Academic press, 2000.

[94] Wlodzimierz M Zuberek. Timed petri nets and preliminary performance evaluation. In Proceedings of the 7th annual symposium on Computer Architecture, pages 88–96. ACM, 1980.

List of publications

List of the publications of the candidate

[1] Yanwen Chen, Yixiang Chen, and Eric Madelaine. "Timed-pNets: A Communication Behavioural Semantic Model For Distributed Systems." *Journal: Frontier of Computer Science (SCIE).* (2014).

[2] Yanwen Chen, Yixiang Chen. "Real-time Scheduling in Cyber Physical System." *Journal of CEAI, Vol.13, No.3, pp. 41-50,(SCIE).* (2011).

[3] Yanwen Chen, Yixiang Chen, and Eric Madelaine. "Investigation on Time Properties of Timed-pNets." *NASAC2014, Journal of Computer Science.* (2014)

[4] Yanwen Chen, Yixiang Chen, and Eric Madelaine. "Timed-pNets: A formal communication behavior model for real-time CPS system." *In Trustworthy Cyber-Physical Systems.* (2012).

[5] Yanwen Chen, Fabrice Huet, Yixiang Chen. "Implementation and optimization of RDF query using Hadoop." *First International Conference on Cloud Computing and Services Science (CLOSER).* (2011.)

list of figures

list of tables