Aspect-Oriented Formal Techniques of Cyber Physical Systems

Lichen Zhang
Guangdong University of Technology, Guangzhou, China
Email: zhanglichen1962@163.com

***Abstract*—Cyber-physical systems pose considerable technical challenges, ranging from the distributed programming paradigms to networking protocols with timeliness as a structuring concern, including systems theory that combines physical concerns and computational concerns. Formal specification techniques for such systems have to be able to describe all these concerns. Unfortunately, a single specification technique that is well suited for all these concerns s is yet not available. Instead one finds various specialized techniques that are very good at describing individual concerns of cyber-physical system. This observation has led to research into the combination and semantic integration of specification techniques. This paper proposes a formwork for specifying cyber physical systems based on aspect-oriented formal method, which exploits the diversity and power of existing formal specification languages. There is no requirement that different aspects of a system should be expressed in the same language. So the different aspects can be specified by one formal specification technique or different formal specification techniques. The proposed aspect–oriented formal framework is such a formwork. On the one hand, it can deal with continuous-time systems based on sets of ordinary differential equations. On the other hand, it can deal with discrete-event systems, without continuous variables or differential equations. We present a combination of the formal methods Timed-CSP, ZimOO and differential (algebraic) equations or differential logic. Each method can describe certain aspects of a cyber physical system: CSP can describe communication, concurrent and real-time requirements; ZimOO expresses complex data operations; differential (algebraic) equations model the dynamics and control (DC) parts. Two case studies illustrate the specification process of aspect-oriented formal specification for cyber physical systems.**

***Index Terms*—Aspect-oriented,, Cyber Physical Systems, Formal Method, ZimOO,Timed-CSP**

I. INTRODUCTION

Cyber-Physical Systems (CPS)[1] are integrations of computation with physical environments. Embedded computers and networks monitor and control the physical environments, often with feedback loops where physical environments affect computations and vice versa. In the physical world, the passage of time is inexorable and concurrency is intrinsic. Neither of these properties is present in today's computing and networking abstractions. Example CPSs include automobiles, aircraft, air traffic control, power grids, oil refineries, medical devices, patient monitoring, and smart structures. Software is becoming an increasingly important element of the operation of these systems, and must do so dependably, safely, securely, efficiently and in real-time. A cyber physical system [2]is a computer system whose correctness depends not only on the output results, but also the time at which the output results are produced. Consider for example an anti-lock brake system in a modern vehicle. Sensors provide it within formation of the current wheel speed and it must react in a timely fashion when the driver applies the brake. If it does not react timely, it is not correct and will probably be of more harm than help to the driver. The design of cyber physical systems (CPS) poses many challenges because of their complexity, high safety requirements, distribution, and real time nature[3].

The design and verification of cyber physical systems requires a good understanding of formal mathematical methods that are found in both computer science and the traditional engineering disciplines. These formal methods are used to model, verify, and design complex embedded systems in which the interaction of computational and physical processes must be approached in a holistic manner. Formal methods treat system components as mathematical objects and provide mathematical models to describe and predict the observable properties and behaviors of these objects. There are several advantages to using formal methods for the specification and analysis of cyber physical systems: the early discovery of ambiguities, inconsistencies and incompleteness in informal requirements, the automatic or machine-assisted analysis of the correctness of specifications with respect to requirements and the evaluation of design alternatives without expensive prototyping. Formal Description Techniques FDTs like Object-Z[4]and Timed-CSP [5] have been successfully applied to the specification of ''traditional'' communication protocols, services and network applications[6].

Aspect oriented software development (AOSD)[7] is a relatively new type of development that simplifies maintenance and increases usability of software. Aspect oriented development can be implemented at the inception of software or current software can be reworked to use the process. Development using aspects can also allow proprietary and open source software to complement each other by separating features from the main code. AOSD programming technologies provide linguistic mechanisms for separate expression of concerns, along with implementation technologies for weaving these separate concerns into working systems.

© 2012 ACADEMY PUBLISHER
doi:10.4304/jsw.7.4.823-834

Aspect-oriented software engineering (AOSE) technologies are emerging for managing the process of developing systems within this new paradigm. Cyber physical systems could be separated into different crosscutting concerns and designed independently by using AOP techniques. When designing the system, aspects are analyzed and designed separately from the system's core functionality, such as real-time, security, error and exception handling, log, synchronization, scheduling, optimization, communication, resource sharing.[8]

In this paper, we provide some ideas for the aspect – oriented formal specification of cyber physical systems and two well known case studies to validate aspect-oriented formal specification.

II. CYBER PHYSICAL SYSTEMS

The integration of physical systems and processes with networked computing has led to the emergence of a new generation of engineered systems: Cyber-Physical System s(CPS). Such systems use computations and communication deeply embedded in and interacting with physical processes to add new capabilities to physical systems. These cyber-physical systems range from miniscule (pace makers) to large-scale (the national power-grid). Because computer-augmented devices are everywhere, they are a huge source of economic leverage[9]. A CPS is a system:

- in which computation/information processing and physical processes are so tightly integrated that it is not possible to identify whether behavioral attributes are the result of computations (computer programs), physical laws, or both working together;
- where functionality and salient system characteristics are emerging through the interaction of physical and computational objects;
- in which computers, networks, devices and their environments in which they are embedded have interacting physical properties, consume resources, and contribute to the overall system behavior.

CPS as a discipline is an engineering discipline, focused on technology, with a strong foundation in mathematical abstractions. CPS needs shares many of these abstractions with computer science, but requires adapting them to embrace the dynamics of the physical world. Computer science, as rooted in the Turing-Church notion of computability, abstracts away core physical properties, such as the passage of time, that are required to include the dynamics of the physical world in the domain of discourse[10]. We think that models of CPS software should satisfy the following requirements[11]:

- Composability: An embedded software model should be able to encode non-functional properties and how they compose to form the final system.
- Correctness: An embedded software model must be able to reflect to the designer the full consequences of his/her design choices and prevent the use of incorrect ones. For example, the software model can be restricted to the use of constructs that are analyzable
- Fidelity: A software model should be able to model the software down to its implementation and deployment. In other words, the running code of the final system and the model of the system should not deviate from one another. The best approach to avoid these deviations is perhaps to rely on code generators

A notable possible culprit is the lack of timing in computing abstractions. Indeed, this lack has been exploited heavily in such computer science disciplines as architecture, programming languages, operating systems, and networking[12]. The temporal (behavioral) models of real-time and embedded systems can be grouped into three main categories as follows[39]:

- Asynchronous/Causal models are merely concerned with the proper ordering of activities (instructions, actions, so on), due to some control or data flow prescription. Some amount of scheduling may be needed if the specified flow is partial. Therefore, in such cases, time is viewed in terms of causal dependencies rather than specific quantities or durations. This model is used widely at the algorithmic software level (and in software models of hardware at the transaction level). In the presence of concurrency, the varying speeds of asynchronous components (with synchronous or asynchronous communications) generally lead to non-deterministic behavior.
- Synchronous/Clocked models add the notion of simultaneity of events and activities. Time is modeled as a discrete set of instants, and need not be connected to any physical reality, in the sense that the corresponding clock need not be regular. Henceforth we shall call this time "logical". This type of time model is used in (discrete step) simulation formalisms such as Simulink/Stateflow, in synchronous languages and Statecharts, as well as in hardware description languages at the register transaction level (e.g., VHDL, Verilog, SystemC, etc.).
- Real/Continuous time models take physical durations into account. These are important for doing various time-related analyses (e.g., deadline matches) and, in particular, for real-time scheduling as in RMA approaches . They are also used for modeling the temporal characteristics of the physical environment or system with which the embedded system is interacting (usually before discretization).

A real-time and embedded modeling language for cyber physical systems needs concepts for dealing with different models of time, an, in particular, the three models of time described above, since they represent most of the common cases.

Cyber-physical systems by nature will be concurrent. Physical processes are intrinsically concurrent, and their coupling with computing requires, at a minimum, concurrent composition of the computing processes with

© 2012 ACADEMY PUBLISHER

the physical ones. The most interesting and revolutionary cyber-physical systems will be networked. The most widely used networking techniques today introduce a great deal of timing variability and stochastic behavior

In order to meet the challenge of cyber-physical system design, We need to realign abstraction layers in design flows and develop semantic foundations for composing heterogeneous models and modeling languages describing different physics and logics[13]. We need to introduce or develop mathematical frameworks that make semantics not only mathematically precise, but also explicit, understandable and practical for system developers as well as tool developers. We need to develop new understanding of compositionality in heterogeneous systems that allows us to take into account both physical and computational properties.

One of the fundamental challenges in research related to CPSs is accurate modeling and representation of these systems. The main difficulty lies in developing an integrated model that represents both cyber and physical aspects with high fidelity. Among existing techniques, aspect-oriented modeling is a suitable choice, as it can encapsulate diverse attributes of cyber physical systems.

III. Aspect-Oriented Formal Specification

The primary goal of a software development methodology is to facilitate the creation, communication, verification and tracing of requirements, design, and implementation. To be truly effective, a modern methodology must also automatically produce implementations from designs, test cases from requirement specifications, analyses of designs and reusable component libraries. Our experience in the development of cyber physical systems and research into software development methodologies have led us to conclude that existing public-domain methodologies do not permit us to achieve these goals. Most often software development methods offer excellent solutions to specific, partial aspects of system development, providing only little of help for other aspects. Classical methods for specifying and analyzing real-time systems typically focus on a limited subset of system characteristics. RTSA , for example, focuses primarily on the functionality and state-oriented behavior of real-time systems. STATEMATE provides three different graphic or diagrammatic languages for three different aspects. Module charts represent the structural aspect of the system, activity charts represent functional aspect of system, and state charts represent the behavior aspect of the system. At the other extreme, formal specification and verification methods strive for fool-proof or error-free designs. they can be used to specifying and analyzing some properties such timing constraints.. Thus integration of different specification methods is desired:

- integration of the methods used to specify systems requirements.
- integration of tools that support these methods and
- integration of the multiple specification fragments produced by applying these methods and tools.

In our opinion, an acceptable cyber physical system design methodology must synthesize the different aspects of systems, use a number of different methods and tools, consist of an orderly series of steps to assure that all aspects of the requirement and all the design aspects have been considered[14].The cyber physical system design methodology should address these problems:

- supporting specification and analysis of a system from multiple aspects to capture different perspectives on a system,
- providing methods and tools that are appropriate for different aspects for improved
- understandability of specifications, employing a formal basis to integrate multiple aspects and perform analyses with mathematical rigor, and providing methods to handle the size and complexity required by large-scale systems.

When specifying cyber physical systems, decomposition and composition are the primary methods for coping with complexity[15]. Typically, decomposition can be done in two orthogonal dimensions. First, the system is decomposed in the vertical dimension by partitioning the cyber physical systems into loosely-coupled subsystems with well-defined interfaces. Each subsystem is then decomposed further in the horizontal dimension usually in a top-down fashion. However, with this approach it is easy to introduce unnecessary complexity due to two facts. Firstly, the interfaces between subsystems are defined before capturing their collective behavior. Secondly, application-specific parts, that crosscut the two dimensions, are scattered around in different components of the subsystems. Furthermore, they are tangled with other parts. In recent years the recognition of problems related with scattering and tangling in object-oriented software systems has led to an introduction of an aspect-oriented design methodology enabling different, possibly overlapping, design concerns to be decomposed separately before composing them together. Aspect-oriented approaches use a separation of concern strategy, in which a set of simpler models, each built for a specific aspect of the system, are defined and analyzed. Each aspect model can be constructed and evolved relatively independently from other aspect models. This has three implications:

First, an aspect model can focus on only one type of property, without burden of complexity from other aspects. Hence an aspect model is potentially much simpler and smaller than a traditional mixed system model. This is expected to dramatically reduce the complexity of understanding, change, and analysis.

Second, different levels of detail or abstraction can be used in the construction of different aspect models. This allows us to leverage of existing understanding of certain aspect of the system to reduce the complexity of modeling and analysis. For example, if the timing property of a component/subsystem is well understood,

© 2012 ACADEMY PUBLISHER

we can build an extremely simple timing model for the component.

Third, Existing formal notations normally are suitable for describing one or a few types of system properties. By adopting the aspect concept, we can select the most suitable notation to describe a given aspect. Likewise, we can select the most suitable analysis techniques to analyze a given property.

"Formal Methods" refers to mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems. The phrase "mathematically rigorous" means that the specifications used in formal methods are well-formed statements in a mathematical logic and that the formal verifications are rigorous deductions in that logic (i.e. each step follows from a rule of inference and hence can be checked by a mechanical process.) The value of formal methods is that they provide a means to symbolically examine the entire state space of a digital design (whether hardware or software) and establish a correctness or safety property that is true for all possible inputs. Although the use of mathematical logic is a unifying theme across the discipline of formal methods, there is no single best "formal method". Each application domain requires different modeling methods and different proof approaches. Furthermore, even within a particular application domain, different phases of the life-cycle may be best served by different tools and techniques. For example, a theorem prover might be best used to analyze the correctness of a RTL[16] level description of a Fast Fourier Transform circuit, whereas algebraic derivational methods might best be used to analyze the correctness of the design refinements into a gate-level design. Therefore there are a large number of formal methods under development throughout the world. Formal methods in software draw upon some advanced mathematics. Mathematical topics of interest include formal logic, set theory, formal languages, and automata. Formal methods support precise and rigorous specifications of those aspects of a computer system capable of being expressed in the available formal languages. Since defining what a system should do, and understanding the implications of these decisions, are the most troublesome problems in software engineering, this use of formal methods has major benefits.

Many of the existing models[17] used to specify functional system behavior – operational techniques such as automata, Petri nets, process algebra, and descriptive techniques like logics – have been enriched by means to express non-functional real-time properties. Some work of is based on variants of timed automata. Time restrictions are introduced by labeling transitions or states of extended finite state machines with time limits, clocks and time variables. An upper bound u and a lower bound l is assigned to each transition of the timed automaton. Once enabled, the transition may be executed not sooner than l and not later than u time units after the enabling. Similar conditions for the enabling of transitions can be formulated referring to the values of time variables. These are model-theoretic techniques in that they distinguish all execution sequences of a system into those that satisfy the timing constraints the ‘‘good’’ ones, and those that do not satisfy them the ‘‘bad’’ ones. Only those systems that can only reveal ‘‘good’’ execution sequences implement the specification. Similar timed extensions have been defined for Petri Nets, *Time Petri Nets, Object Composition Petri Nets* and *Time Stream Petri Nets* .There have also been numerous real-time extensions to process algebras,. A process algebra-based QoS specification language based on the FDT LOTOS has been proposed. Temporal logics are the descriptive counterpart to specifying state transition systems by automata .Temporal logics specify qualitative temporal relationships between states. A program satisfies a temporal logic specifications if all of its execution state sequences satisfy these temporal relations. As can be expected, extensions have been introduced to augment temporal logics with constructs that specify quantitative real-time relations between states. Examples are Metric Temporal Logic MTL and Quality of Service Temporal Logic QTL that use real-time interval annotations to the temporal operators, and techniques introducing explicit timer variables as in Real-Time Temporal Logic RTTL or Temporal Logic of Actions TLA .LOTOS is used to specify the functional requirements and multimedia systems. At this stage, no real-time constraints or QoS guarantees are taken into account. At a second stage, a temporal logic based language is used to describe the ordering constraints over the functional behaviors and also the time-critical QoS requirements. LOTOS have been proven to be an efficient and abstract language for the specification of concurrent and non-determinism behaviours. QTL features real-time capabilities which are required to express real-time properties.

Z [18]is a formal language used to define data types and to show the effect of operations on these types. It lacks, however, features to express the order in which the operations are executed. Process algebras[19], like CSP, on other hand, are suitable for showing the order of the occurrence of events but lack the ability to handle complex abstract data types and operations. Finally, formalisms like temporal logics and its derivatives concentrate on time aspects. The increasing complexity of real-time systems has made the use of formal specification more frequent in this area, and new languages have proliferated. Timed CSP is an extension of CSP that includes new operators like *Wait* and _ (timeout). It has a new semantic model, derived from the untimed models of CSP, to represent dense time information, and a proof system. Timed CCS has an operational semantics; a communication on a described by $a(t)e$, e is restricted to happen in the closed time interval , and, after the communication, the variable t holds the time at which it occurred. Proof rules for timed CCS have been elaborated and shown to be independent of the time domain used. Logic approaches to specification benefit from clear notations and automated validation using existing theorem provers. Modal logic adds time reasoning in logical formulae. Temporal logic is a modal logic in which new operators for quantification

© 2012 ACADEMY PUBLISHER

either in the future or in the past are included. To overcome difficulties with modularity, temporal logic is often used in combination with other techniques. An example is the work of Duke and Smith , in which temporal logic is used in the invariants of Z specifications to define liveness properties. Duration calculus (DC) is concerned with intervals instead of time instances. Real-time logic (RTL) extends predicate logic by relating events with the time in which they occur. In contrast to other logics, RTL allows specification of the absolute timing of events, and not only their relative ordering; it also provides a uniform way of incorporating different scheduling disciplines. Timed automata extends state-transition systems with finitely many real-valued clock variables that are used in annotations. Analysis is based on a finite quotient of the infinite space of clock valuations. Work has been carried out on verification algorithms, including heuristics, and tools. Petri Nets allow mathematical modeling of discrete event systems in terms of conditions and events, and the relationship between them. Time information has been added to Petri Nets in a number of forms. The most common approach is to add time delays to transitions. A similar approach assigns delays to places instead of transitions, and creates a delay between the time the token arrives in a place and the time it enables a transition to fire. A more flexible approach assigns intervals to the transitions and time stamps to the tokens. Specifications of complex systems normally involve a mixture of data types, operations, and time constraints; current research has focused on more comprehensive languages. Circus combines CSP, Z, specification statements and guarded commands to provide a notation for both specification and programming, and for verification by refinement. Time is continuous by nature, but a discrete representation of time is also satisfactory in most cases. In specification languages, time is often represented by real numbers, but in programming languages, time is represented by integers. In principle, the continuous time model is more appropriate because it can express time in both forms, and time in the real world is continuous. A continuous time model, however, cannot be implemented by a software system. Object-Z is an object-oriented extension of the formal specification language Z. It adds, to Z, notions of classes and objects, and inheritance and polymorphism. TCOZ[20] is a blending of Object-Z and Timed CSP.

ZimOO[21] is based on Object-Z, an object-oriented extension of Z, A system specification in Z consists of a description of the global system state and a set of operations describing how this state can be changed. Object-Z provides additional means for describing systems in an object-oriented style, an Object-Z specification consists of a set of classes, each containing a state description and a set of operations Z and Object_Z support only specifications of discrete systems. ZimOO is an extended subset of Object-Z allowing descriptions of discrete and continuous features of a system in a common formalism ZimOO supports three different kinds of classes: discrete as in Object-Z, continuous and hybrid classes. Thus, the system can be structured better and the well-known suitable formalisms can be applied to describe, analyze, and refine the different parts of the system. The bridge between the continuous and the discrete world is built by hybrid classes.

The differential dynamic logic (dL) [22] is a logic for specifying and verifying hybrid systems [23]. The logic dL can be used to specify correctness properties for hybrid systems given operationally as hybrid programs. The basic idea for dL formulas is to have formulas of the form $[\alpha]\varphi$ to specify that the hybrid system α always remains within region φ, i.e., all states reachable by following the transitions of hybrid system α statisfy the formula φ. Dually, the dL formula $<\alpha>\varphi$ expresses that the hybrid system α is able to reach region φ, i.e., there is a state reachable by following the transitions of hybrid system α that satisfies the formula φ.

For instance, the following formula expresses that for the state of a train controller train, the property $y \le m$ always holds true when starting in a state where $v^2 \le 2b(m-y)$ is true: $v^2 \le 2b(m-y) \rightarrow [train]y \le m$[22][23].

In many ways, Object-Z and Timed CSP complement each other in their capabilities. Object-Z has strong data and algorithm modeling capabilities. The Z mathematical toolkit is extended with object oriented structuring techniques. Timed CSP has strong process control modeling capabilities. The multi-threading and synchronization primitives of CSP are extended with timing primitives. Moreover, both formalisms are already strongly influenced by the other in their areas of weakness. Object-Z supports a number of primitives which have been inspired by CSP notions such as external choice and synchronization. CSP practitioners tend to make use of notation inspired by the Z mathematical toolkit in the specification of processes with internal state. The approach taken in the TCOZ notation[20] is to identify operation schemas (both syntactically and semantically) with (terminating) CSP processes that perform only state update events; to identify (active) classes with non-terminating CSP processes; and to allow arbitrary (channel based)communications interfaces between objects. An example of timing expression[20][33] is shown as Fig.1.

Aspect-oriented specification is made by extending TCOZ notation with aspect notations. The schema for aspect specification in has the general form as shown in Fig.2, Fig.3 and Fig.4.

IV. Related Works

Mohammad Mousavi et al. present an extension to the GAMMA formalism[], which they name AspectGAMMA[24], and we show how non-computational aspects can be expressed separately from the computation in this framework. They discuss the main characteristics of an aspect-oriented formal specification framework, which is based on a multiset transformation language called GAMMA, a formalism based on multiset rewriting they illustrate how having a tailor-made formalism for each aspect that is abstracted from other

© 2012 ACADEMY PUBLISHER

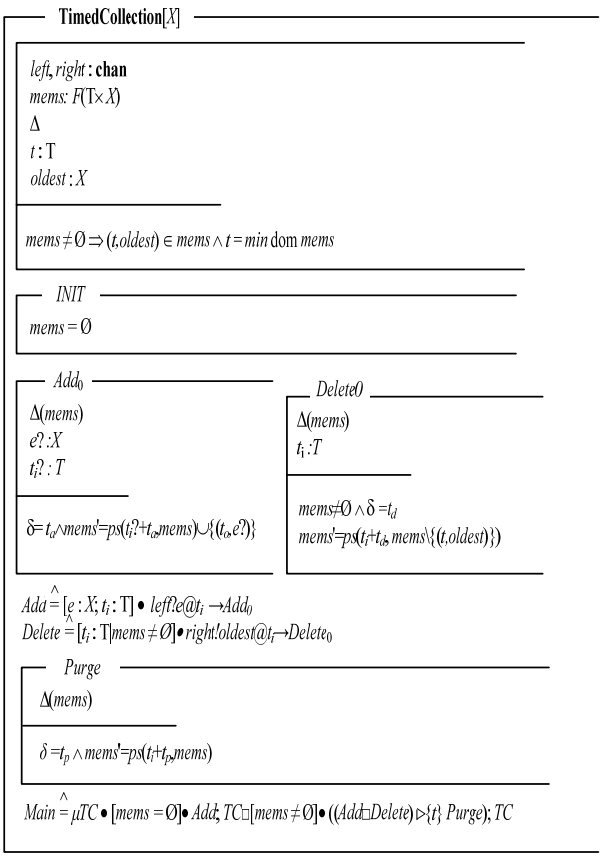
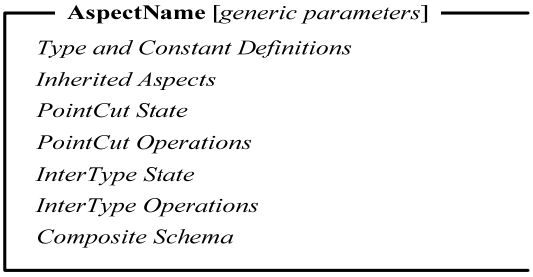


Figure 1. Timed Object-Z Model of the TimeCollection



Figue2.. Aspects of Model Structure

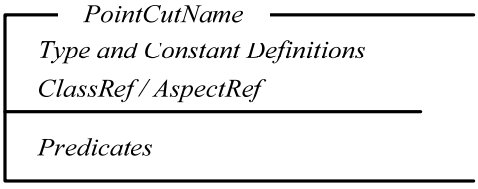


Figure 3. PointCut Operation Schema of Structure

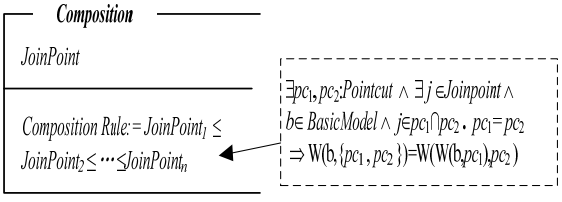


Figure 4. Composition Schema of Structure

aspects is a key benefit of such a formal design framework. To clarify the discussions, they sketch an architecture specification and design method for reactive distributed real-time embedded systems. In the approach they describe in their paper, they propose separating the concerns of computation, coordination, timing, and distribution, through different simple and abstract notations for these aspects. They also describe a weaving process that maps all these different aspects to a single semantic domain. The method is based on a formal semantics that should ultimately enable automated reasoning about designs. The idea exploited in this method can be extended to other aspects, and extended with more complex weaving criteria[25].

Lynne Blair proposes multi-paradigm approach to formal specification and shows how this approach can be successfully used in the specification of distributed multimedia systems[26]. He takes an example, a published description of an algorithm to establish the initial synchronization of distributed stored media streams that avoids the need for large buffers (e.g.if the locations of the media sources are widely distributed). He shows how this algorithm can be specified using a combination of real-time temporal logic and timed automata. He then describes how the different specifications (languages) can be combined in order to analyze the overall behaviour[27].

Huiqun Yu et al. propose a formal aspect-oriented modeling language called AspectZ, and an aspect-oriented modeling method in AspectZ[28]. They lift aspect-oriented method from code level to design level, which enhances quality assurance in the early stage of software life cycle. AspectZ is an extension to Z . The basic idea is to provide means for observing behaviors of Z schemas and depicting their interrelationships, and to provide ways for weaving interrelated schemas. Correctness of aspect weaving can be formally verified by reasoning mechanisms of Z notations.

Jochen Hoenicke uses a combination of three techniques for the specification of processes, data and time: CSP, Object-Z and Duration Calculus[29]. The basic building block in our combined formalism CSP-OZ-DC is a class. First, the communication channels of the class are declared. Every channel has a type which restricts the values that it can communicate. There are also local channels that are visible only inside the class and that are used by the CSP, Z, and DC parts for interaction. Second, the CSP part follows; it is given by a system of (recursive) process equations. Third, the Z part is given which itself consists of the state space, the Init schema and communication schemas. For each communication event a corresponding communication schema specifies in which way the state should be changed when the event occurs. Finally, below a horizontal line the DC part is stated. The combination is used to specify parts of a novel case study on radio controlled railway crossings. Johannes Faber formally specifies a part of the European Train Control System (ETCS) with the specification language CSPOZ-DC treating the handling of emergency messages.

© 2012 ACADEMY PUBLISHER

Naoyasu Ubayashi and Shin Nakajima examine a software development method starting with the feature-oriented modeling method to have VDM-based formal design. In order to overcome the problem that a feature may be scattered over the VDM design description, the notion of the aspect is adapted to propose AspectVDM[30]. The identified features are concisely represented in AspectVDM to demonstrate modular descriptions of cross-cutting concerns in VDM. refinement and weaving are similar in that both concerns with the model transformations. However, in the refinement, the description becomes concrete toward the programming level implementation. On the other hand, the woven description stays at the same level as before because the base and the aspect look at the same abstraction level. Further, weaving provides a basis for the incremental development of the design. The key point of the aspect is that it provides an explicit language construct *Aspect* module which contains all the necessary information to modify the base description. What, otherwise scattered, can be described in one *Aspect* module.

ZimOO is based on Object-Z, an object-oriented extension of Z[21]. A system specification in Z consists of a description of the global system state and a set of operations describing how this state can be changed. Object-Z provides additional means for describing systems in an object-oriented style, i.e., an Object-Z specification consists of a set of classes, each containing a state description and a set of operations. Z and Object-Z support only specifications of discrete systems. ZimOO is an extended subset of Object-Z allowing descriptions of discrete and continuous features of a system in a common formalism. ZimOO supports three different kinds of classes: discrete (as in Object-Z), continuous, and hybrid classes. In the current version of ZimOO, discrete classes can inherit only from discrete classes, continuous only from continuous classes, and hybrid only from hybrid classes. Hybrid classes are created by introducing discrete and continuous object-valued variables. The information interchange between discrete and continuous objects is realized by control variables, which transfer data from the discrete to the continuous world and vice versa.

G.K. Palshikar presents the use of formal specifications in small automatic train operation system ATO-2000. ATO-2000 is a safety-critical, real-time, distributed, mobile computing system. The formal specifications, design and implementation of the safety constraints in ATO-2000 is described. The formal specifications (in Z) of the core safety requirements in ATO-2000 are presented which include a new representation of the track topology.

Hybrid systems are models for complex physical systems and are defined as dynamical systems with interacting discrete transitions and continuous evolutions along differential equations. With the goal of developing a theoretical and practical foundation for deductive verification of hybrid systems, Andre Platzer introduces a dynamic logic for hybrid programs, which is a program notation for hybrid systems. As a verification technique that is suitable for automation, we introduce a free variable proof calculus with a novel combination of real-valued free variables and Skolemisation for lifting quantifier elimination for real arithmetic to dynamic logic. The calculus is compositional, i.e., it reduces properties of hybrid programs to properties of their parts. The main result proves that this calculus axiomatises the transition behaviour of hybrid systems completely relative to differential equations. In a case study with cooperating traffic_ agents of the European Train Control System, the case study show that our calculus is well-suited for verifying realistic hybrid systems with parametric system dynamics[22][23][31].

B. Mahony and J.S. Dong.propose a timed, multithreaded object modeling notation for specifying real-time, concurrent, and reactive systems. The notation Timed Communicating Object Z (TCOZ)[33] builds on Object-Z's strengths in modeling complex data and algorithms, and on Timed CSP's strengths in modeling process control and real-time interactions. TCOZ is novel in that it includes timing primitives, properly separates process control and data/algorithm issues and supports the modeling of true multi-threaded concurrency. TCOZ is particularly well suited for specifying complex systems whose components have their own thread of control. The expressiveness of the notation is demonstrated by a case study in specifying a multi-lift system that operates in real-time.

V. Case Study One: Aspect-Oriented Formal Specification of A Train Control System

The Problem that must be addressed in operating a railway is numerous in quantity, complex in nature, and highly inter-related . For example, collision and derailment, rear-end, head-on and side-on collisions are very dangers and may occur between trains. Trains collide at level crossing. Derailment is caused by excess speed, wrong switch position and so on. The purpose of train control is to carry the passengers and goods to their destination, while preventing them from encountering these dangers. Because of the timeliness constraints, safety and availability of train systems, the design principles and implementation techniques adopted must ensure to a reasonable extent avoidance of design errors both in hardware and software. Thus a formal technique relevant to design should be applied for train systems development. The purpose of our exercise is to apply aspect -oriented formal method s to develop a controller for train systems that tasks as input: a description of track configuration and a sequence of description of the moves of each of these trains.

The Controller should take care of trains running over the track. It should control the safety of the configuration, ie. No two trains may enter the critical section. When one critical section is occupied, some others, which share some part of section with this one , should be locked. The controlled can control the status, speed, position of trains[34-37].

In order to keep the description focused, we concentrate on some particular points in train control

© 2012 ACADEMY PUBLISHER

systems rather than the detailed descriptions of all development process. The specification is made by integrating Object-Z, Timed CSP, ZimOO and DL.

Assuming the train starts in a controllable state, the following global and unbounded-horizon safety formula about the system y≤m holds. As system invariant we choose[22][23][31]:

$$inv \equiv v^2 \leqslant 2b(m-y) \wedge \varepsilon > 0 \wedge v \geqslant 0$$

Which expresses that it is possible to completely stop the train within the distance left to the end of the movement authority. This constraint describes a controllable state of the train and therefore we choose inv as initial configuration of the system. When an movement authorities(MA) has been granted up to the track position m and the train is currently located at position y then dL can analyze, for example, the following safety statement about the (simplified) acceleration system[22][23][31]:

$$\Psi \rightarrow [((m - y < s?; a := -b) [(m - y \geq 2s?; a := 0.1)); \ddot{y} = a] y < m$$

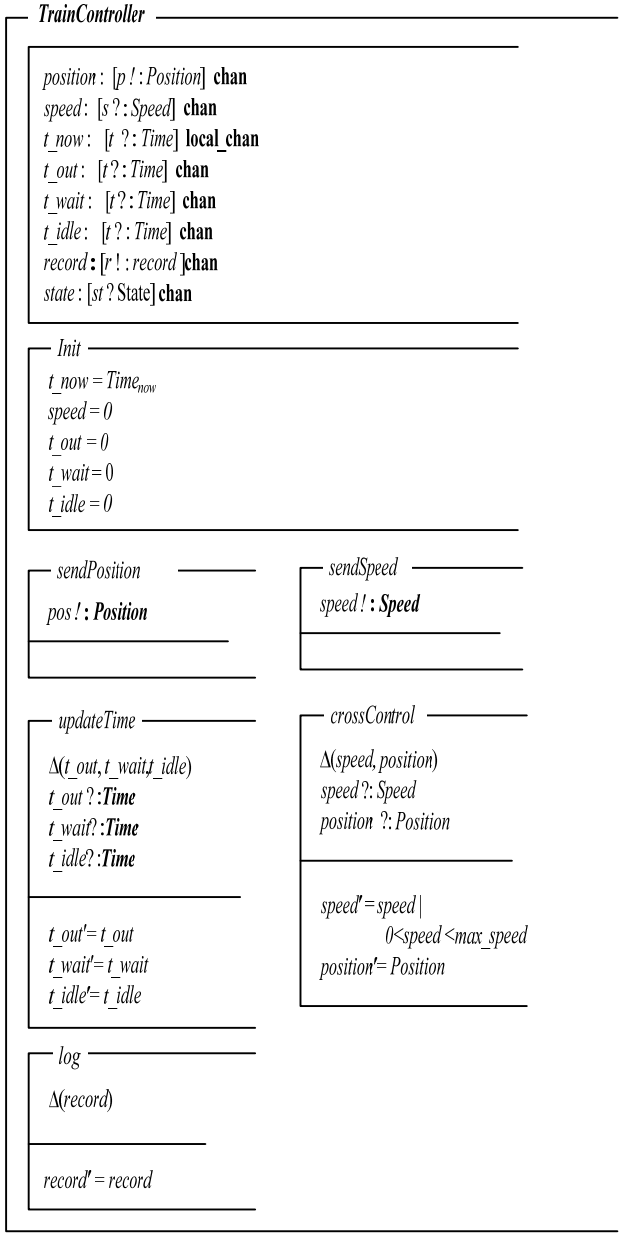


Figure 5. Control Model of Train Dispatching System

It expresses that, under a conditionΨ about parameters, trains always remain within their MA m. Further, it specifies that the train decelerates using engine brakes of force b if the safety envelope is underrun (m−y < s). It slowly accelerates if there is sufficient distance (m − y≥2s).

A train controller [21][29][34-37]limits the speed of the train, decides when it is time to switch points and secure crossings, and makes sure that the train does not enter them too early as shown in Fig.5.

In automatic railway crossing system, The sensors detect the presence of the train near rail crossing and barrier shuts down when train is approaching to the railway crossing. Once train crosses the rail crossing barriers opens by itself as shown in Fig.6 and Fig.7.

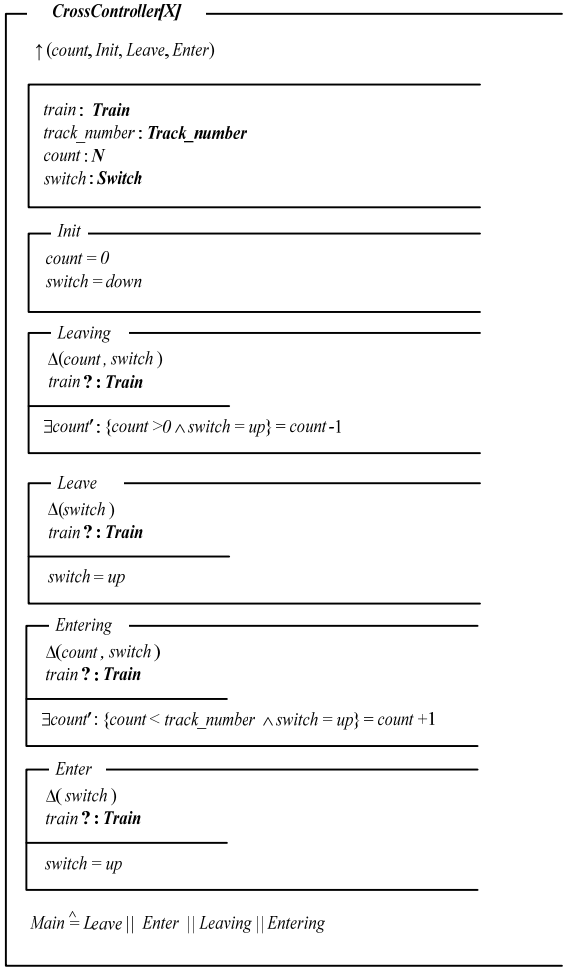


Figure 6. Trains through the intersection mode

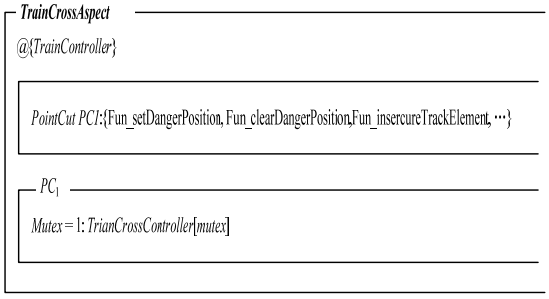


Figure 7. Cross roads aspects model

© 2012 ACADEMY PUBLISHER

To extend state-based and behavioural techniques with real-time aspects, different approaches exist. One approach is unifying a state based language and Timed CSP, an extension of CSP with a time-out operator. In this paper ,we take a different approach. We integrate CSP and Object-Z with aspect-oriented technique to specify real-time systems as shown in Fig.8 and Fig.9.

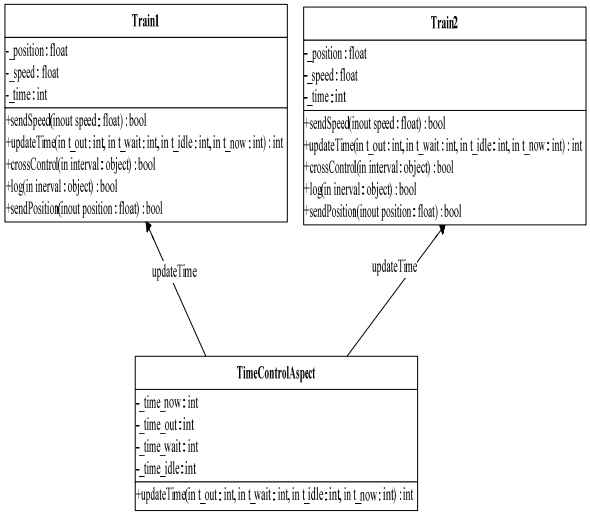


Figure 8. Aspects of Time Model

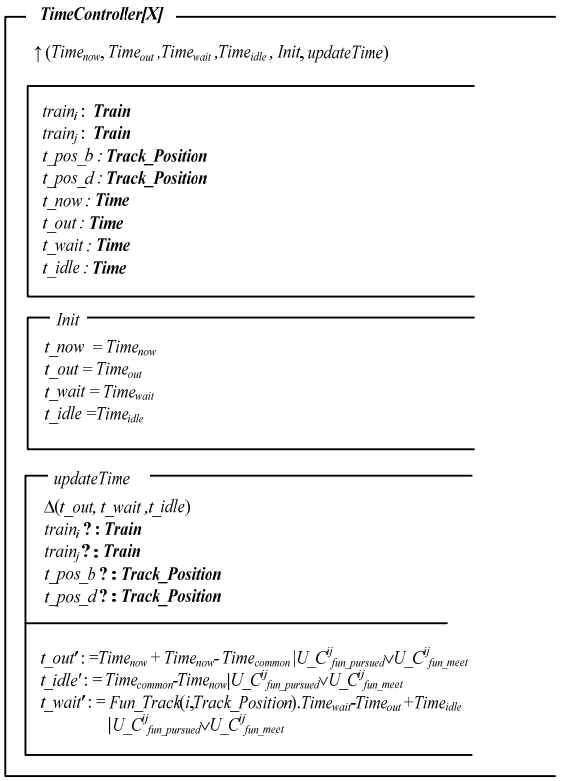


Figure 9. Time Aspect Formalization Method

Logs monitor system behavior. Whether you are running train control systems, whether each event happens, all of these have some sort of event logging inbuilt into them. Any time your system reacts to anything an event log is generated as shown in Fig.10.

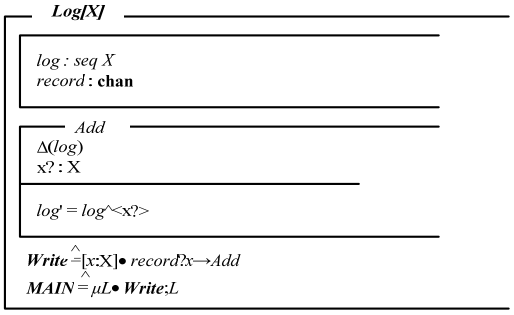


Figure 10. Log Class Model of Train

Finally, woven model is shown as Fig.11

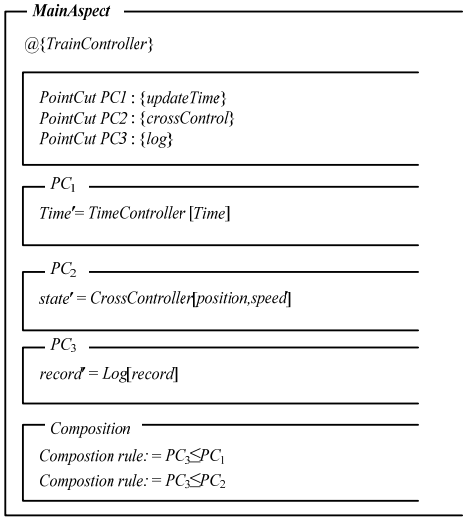


Figure 11. Woven Aspects of Diagram.

VI. CASE STUDY TWO: ASPECT-ORIENTED FORMAL SPECIFICATION OF LIFT SYSTEM

A lift consists of four parts [33][38]: a door for allowing access to and from the lift, a shaft for transporting the lift, an internal queue for determining the lift itinerary, and a controller for coordinating the behaviour of the other components. The door cycle is initiated by receipt of an *open* signal from the lift controller and completed by sending a *close* signal as shown in Fig.12[33][38]. Floors may be divided into two classes, those from which it is possible to travel upward and those from which is possible to travel downward. The *TopFloor* is a floor from which only downward travel is possible , A *MiddleFloor* is a floor from which both upward and downward travel is possible. A *MiddleFloor* is a floor from which both upward and downward travel is possible. The different floor model is shown as Fig.13. The following timing properties must be captured in the model: lift travel time between two consequent floors is a constant, however there is a constant time delay for acceleration and braking; without interrupts, the lift door should be kept at the `open' state for a fixed time period before closing. The maximum time to pass from one floor to another is *t* for each floor travelled plus a delay of *delay*

© 2012 ACADEMY PUBLISHER

caused by initial acceleration and final braking of the lift. The shaft model is captured in Fig.14.

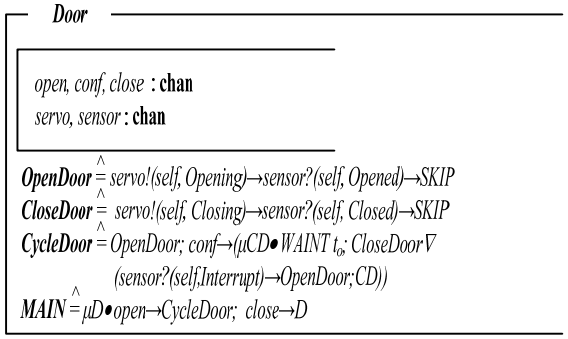


Figure 12. Life Door Mode

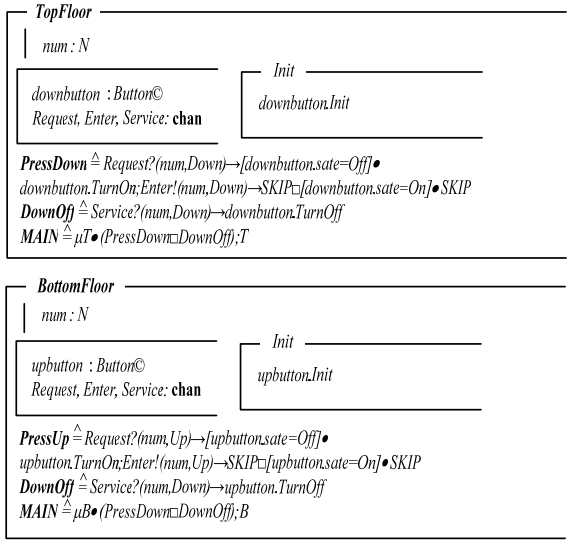


Figure 13. Different Floor Mode

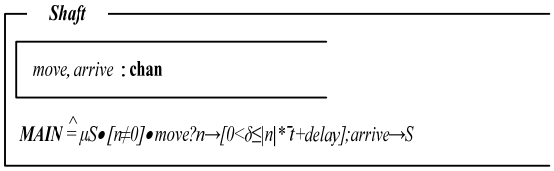


Figure 14. Lift Shaft Model

The lift controller keeps record of the current floor and movement direction and provides the interface between the lift environment and the other lift components as shown in Fig.15.

A lift can not go upward and downward simultaneously, Lift Move Aspect of Control Model is expressed as Figure 16 and Figure 17.

VII. CONCLUSION

In this paper we proposed to use aspect-oriented formal specification for cyber physical systems, we present a combination of the formal methods Timed-CSP and Object-Z. Two case studies of illustrate the specification process of aspect-oriented formal specification for cyber physical systems.

The further work is devoted to integrated aspect-oriented formal development tool.

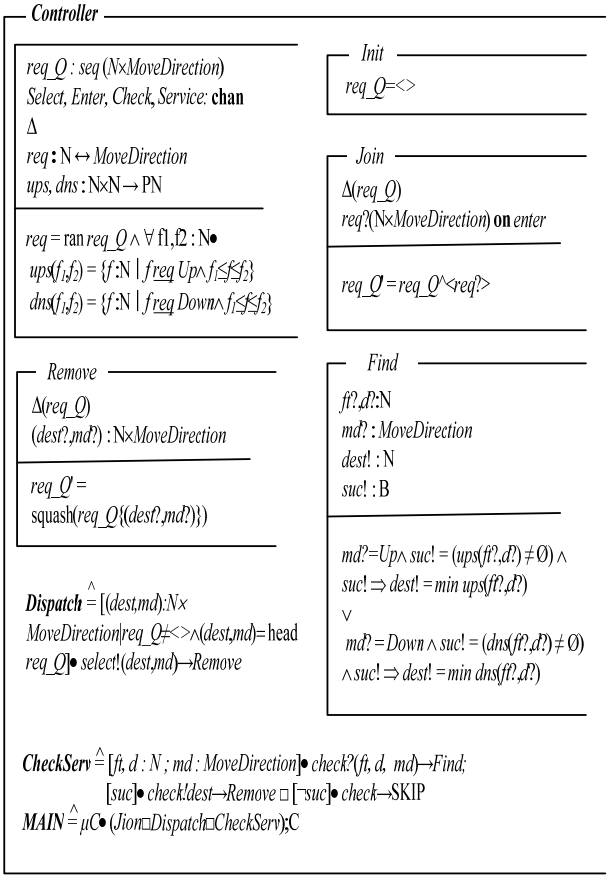


Figure 15. Lift System Controller Model

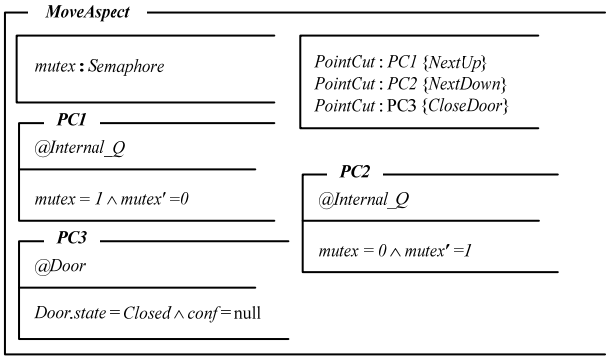


Figure 16. Lift Move Aspect of Control Model

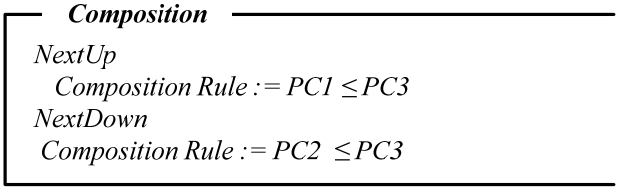


Figure 17. Lift Move Aspect of Composition Model

© 2012 ACADEMY PUBLISHER

ACKNOWLEDGMENT

This work is supported by the Major Program of National Natural Science Foundation of China under Grant No.90818008, National Natural Science Foundation of China under Grant No. 61173046 and Natural Science Foundation of Guangdong province under Grant No.S2011010004905.

REFERENCES

[1] Lui Sha, Sathish Gopalakrishnan, Xue Liu and Qixin Wang: Cyber-Physical Systems: A New Frontier. ISBN 978-0-387-88734-0, Springer，2009

[2] Edward A. Lee, Sanjit A. Seshia, Introduction to Embedded Systems, A Cyber-Physical Systems Approach, Published by authors, First Edition, 2011, 978-0-557-70857-4G.

[3] Wolf.W. Cyber-physical Systems. Computer,Volume: 42 Issue: 3,88-89,2009

[4] Graeme Smith. The Object-Z Specification Language[M]. Software Verification Research Centre University of Queensland. 2000.

[5] Reed G M, Roseoe A W. A timed model for communicating sequential processes[C]//Pro ICALP'86. Lecture Notes in Computer Science. Berlin:Springer,1986.

[6] Adnan Sherif, Ana Cavalcanti, Jifeng He, Augusto Sampaio: A process algebraic framework for specification and validation of real-time systems. Formal Asp. Comput. 22(2): 153-191 (2010).

[7] Aspect-Oriented Software Development. http://aosd.net/.

[8] Wehrmeister, M.A., Freitas, E.P., and Pereira, C.E., et al., "An Aspect-Oriented Approach for Dealing with Non-Functional Requirements in a Model-Driven Development of Distributed Embedded Real-Time Systems ", 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, Santorini Island, Greece, May7-9, 2007, IEEE Computer Society, pp.428-432.

[9] Edward A.Lee. Cyber Physical Systems Design Challenges. Object Oriented Real-Time Distributed Computing (ISORC), 2008.11th IEEE International Symposium. January 23,2008.

[10] E．A Lee .“Cyber-physical systems - are computing foundations adequate?”in Position Paper for NSF Workshop On Cyber-Physical Systems:Research Motivation, Techniques and Roadmap, October 2006

[11] Dionisio de Niz and Raj Rajkumar.Model-Based Embedded Real-Time Software Development.http://www1.cse.wustl.edu/~cdgill/RTAS03/published/TimeWeaverPosition.pdf

[12] Kaiyu Wan ， K.L. Man and D. Hughes. Specification, Analyzing Challenges and Approaches for Cyber-Physical Systems (CPS). Engineering Letters, 18:3, EL_18_3_14

[13] Holger Giese, Gabor Karsai, Edward A. Lee, Bernhard Rumpe, Bernhard Schätz. "Model-Based Engineering of Embedded Real-Time Systems". Lecture Notes in Computer Science, Springer, 6100, 2010.

[14] Christopher Brooks, Chihhong Cheng, Thomas Huining Feng, Edward A. Lee, Reinhard von Hanxleden. "Model Engineering using Multimodeling". 1st International Workshop on Model Co-Evolution and Consistency Management (MCCM '08), September, 2008.

[15] Patricia Derler, Edward A. Lee, Alberto Sangiovanni-Vincentelli. "Modeling Cyber-Physical Systems". *Proceedings of the IEEE special issue on CPS*, December 2011.

[16] Farnam Jahanian, Aloysius K. Mok.Safety Analysis of Timing Properties in Real-Time Systems. IEEE Trans. Software Eng. 12(9): 890-904 (1986)

[17] Carlo Ghezzi, Miguel Felder and Carlo Bellettini, Real-time systems: A survey of approaches to formal specification and verification, Lecture Notes in Computer Science, 1993, Volume 717/1993, 11-36.

[18] J. Spivey: The Z Notation: A Refernce Manual (2rd Edition). Prentice Hall, UK, 1992

[19] Davies J, Schneider S. A Brief History of Timed CSP[J]. Theoretical Computer Science, 1995, 138(1):243-271.

[20] B. P. Mahony and J.S. Dong. Blending Object-Z and Timed CSP: An introduction to TCOZ. ICSE'98, April 1998.

[21] Viktor Friesen. An Exercise in Hybrid System Specification Using an Extension of Z. citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.30.2010&rep.

[22] André Platzer. Differential dynamic logic for hybrid systems. Journal of Automated Reasoning, 41(2), pp 143-189, 2008.

[23] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. LNCS 4548, pp 216-232. Springer, 2007.

[24] M.R. Mousavi, G. Russello, M. Chaudron, M. Reniers, T. Basten, A. Corsaro, S. Shukla, R. Gupta, D. Schmidt, Using Aspect-GAMMA in the Design of Embedded Systems, *Proceedings of the Seventh IEEE International Workshop on High Level Design, Verification and Test (HLDVT'02)*, Cannes, France, pp. 69--75, IEEE CS, October 2002

[25] M.R. Mousavi, G. Russello, M.R.V. Chaudron, T. Basten, M.A. Reniers. Separation of Quality Concerns in the Development of Distributed Real-time Systems. , *Proceedings of the Third Workshop on Embedded Systems (PROGRESS'02),* Utrecht, The Netherlands, pp. 124--127, Progress/STW Technology Foundation, October 2002.

[26] Blair L., *The Role of Temporal Logic and Time Automata in Distributed Multimedia Systems*, Proceedings of Modal & Temporal Logic Based Planning for Open Networked Multimedia Systems (PONMS '99), Cape Cod, MA, November 5-7, pp 1-7, 199

[27] Blair G.S., Blair L., Chitchyan R., Rashid A., Moreira A., Arao J., *Engineering Aspect-Oriented Systems*, invited chapter in *Aspect-Oriented Software Development*, pp 379-406, R. Filman, T. Elrad, S. Clarke, M. Aksit (eds), Boston: Addison-Wesley, 2005.

[28] Huiqun Yu, Dongmei Liu, Li Yang, Xudong He. Formal aspect-oriented modeling and analysis by AspectZ. *Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, Taipei, Taiwan, July 14-16, 2005

[29] Jochen Hoenicke.Specification of Radio Based Railway Crossings with the Combination of CSP, OZ, and DC. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.4394.

[30] Naoyasu Ubayashi and Shin Nakajima:*Context-aware Feature-Oriented Modeling with an Aspect Extension of VDM*,22nd Annual ACM Symposium on Applied Computing (SAC 2007)---Programming for Separation of Concerns (PSC) Track,ACM PRESS, pp.1269-1274 ,2007

[31] André Platzer,*Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, 2010. 426 p. ISBN 978-3-642-14508-7.

[32] Johannes Faber, Swen Jacobs, Viorica Sofronie-Stokkermans: Verifying CSP-OZ-DC Specifications with

© 2012 ACADEMY PUBLISHER

Complex Data Typesand Timing Parameters. Integrated Formal Methods 2007, July 3rd

[33] B. Mahony and J.S. Dong. Timed Communicating Object Z**.** IEEE Transactions on Software Engineering, 26(2):150-177, Feb 2000.

[34] Jochen Hoenicke. *Combination of Processes, Data, and Time*. PhD thesis, University of Oldenburg, July 2006.

[35] Jochen Hoenicke and Patrick Maier. Model-checking of specifications integrating processes, data and time. In J.S. Fitzgerald, I.J. Hayes, and A. Tarlecki, editors, *FM 2005*, volume 3582 of *LNCS*, pages 465-480. Springer, 2005.

[36] J. Hoenicke and E.-R. Olderog. CSP-OZ-DC: A combination of specification techniques for processes, data and time. *Nordic Journal of Computing*, 9(4):301-334, 2002.

[37] J. Hoenicke and E.-R. Olderog. Combining Specification Techniques for Processes Data and Time. In M. Butler, L. Petre, and K. Sere, editors, *Integrated Formal Methods*, volume 2335 of *Lecture Notes in Computer Science*, pages 245-266. Springer-Verlag, May 2002.

[38] Carsten Suhl, An Integration of Z and Timed CSP for Specifying Real-Time Embedded Systems, Ph.D thesis, Technischen Universitat Berlin, 2002

[39] Holger Giese, Gabor Karsai, Edward Lee, Bernhard Rumpe, Bernhard Schätz, Model-based Engineering of Embedded Real-time Systems 2007. Revised Selected Papers Series: Lecture Notes in Computer Science, Vol. 6100.

© 2012 ACADEMY PUBLISHER