Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 207 (2008) 3–16

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Automatic Verification of Combined Specifications: An Overview

Ernst-Rüdiger Olderog[1] ,[2]

Department of Computing Science
University of Oldenburg
Oldenburg, Germany

Abstract

This paper gives an overview of results of the project "Beyond Timed Automata" carried out in the Collaborative Research Center AVACS (Automatic Verification and Analysis of Complex Systems) of the Universities of Oldenburg, Freiburg, and Saarbrücken. We discuss how properties of high-level specifications of real-time systems combining the dimensions of process behaviour, data, and time can be automatically verified, exploiting recent advances in semantics, constraint-based model checking, and decision procedures for complex data.
As specification language we consider CS-OZ-DC, which integrates concepts from Communicating Sequential Processes (CSP), Object-Z (OZ), and Duration Calculus (DC). Our approach to automatic verification of CSP-OZ-DC rests on a compositional semantics of this languages in terms of Phase-Event-Automata. These can be translated into Transition Constraint Systems which serve as an input language of an abstract refinement model checker called ARMC which can handle constraints covering both real-time and infinite data. This is demonstrated by a case study concerning emergency messages in the European Train Control System (ETCS). For CSP-OZ-DC we also discuss a UML profile and tool support.

Keywords: Real-time systems, complex data, CSP, Object-Z, Duration Calculus, model checking, abstraction refinement, UML profile, tool support

1 Introduction

Computers are more and more used to control the behavior of complex systems, for instance in the traffic domain. Such applications are typically safety critical, i.e., a malfunction of the computers is costly and dangerous. Think of assistance systems that should guarantee the collision freedom of traffic agents such as cars, trains, and planes. Such applications necessitate the use of formal models of the

[1] This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, `http://www.avacs.org/`).

[2] Email: `olderog@informatik.uni-oldenburg.de`

doi:10.1016/j.entcs.2008.03.082

overall system and of formal verification for establishing the relevant safety properties. The models must be able to represent various aspects of the systems such as state spaces and their transformation, communication between system components, real-time constraints, interfaces to a continuously evolving physical environment, and dynamically changing system structures. To cope with such models in a manageable way, combined specification techniques have been proposed, integrating well researched specification techniques for individual system aspects. It is a major research challenge to develop methods for the automatic verification and analysis of such combined specifications modeling complex real-life systems.

To address this challenge the research center AVACS (Automatic Verification and Analysis of Complex Systems) was initiated in 2004. In AVACS, researchers of the Universities of Oldenburg, Freiburg and Saarbrücken as well as the Max-Planck-Institute for Informatics in Saarbrücken collaborate [2]. The idea of AVACS is to bring experts in semantic modeling and specification together with experts in verification and analysis techniques. Research in AVACS is organized in four layers:

(i) Complex Systems:
e.g., the European Train Control System (ETCS)

(ii) Models of Complex Systems:
real-time – hybrid – systems of systems

(iii) Combining Verification and Analysis Technologies:
combine technologies $t_1, ..., t_n$ for system s

(iv) Verification and Analysis Kernel Technologies:
Abstraction – BDDs – Constraint Solving – Heuristic Search – Integer Linear Programming – Model Checking – Lyapunov Method – SAT Solver – Theorem Proving

At the top layer (i) are complex systems like the European Train Control System (ETCS). In ETCS trains communicate wireless with radio block centers (RBCs) that control the traffic in certain areas (see Fig. 1). The RBCs grant movement authorities for trains up to a position closely behind the preceding train. In case of an emergency incident of the first train, the RBC has to ensure that this train and all successive trains will stop safely in order to avoid collisions.

At the bottom layer (iv) there are various individual verification and analysis technologies like "Abstraction" or "Heuristic Search". The idea is to combine at layer (iii) such technologies in a suitable novel way so that particlar system classes can be verified and analyzed. To be successful with such a combination, AVACS pursues a divide and conquer strategy whereby (in the first phase of the project) systems are classified into real-time systems, hybrid systems, and systems of systems. The corresponding research areas are called R, H, and S, each organized into three subprojects.

In this paper we give an overview of one subproject on real-time systems. These are systems that interact with their environment in such a way that for certain inputs the corresponding outputs have to occur within given time bounds. Many embedded systems, in particular those in safety critical applications like the ETCS, are of this

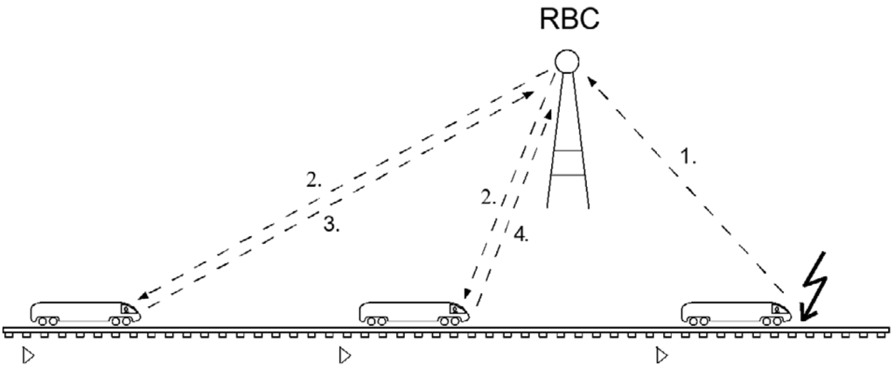

Fig. 1. Case study ETCS

type. The subproject presented here is called "R1: Beyond Timed Automata" and it is coordinated by the author. R1 is motivated by the observation that currently the behavioral verification of specifications of real-time systems is based on their representation as timed automata and relies on model checkers like UPPAAL [20]. A limitation of this approach is that model checking with timed automata is restricted to real-time systems with finite data only. However, in applications systems often exhibit both real-time and complex, infinite data structures.

The goal of R1 is to advance the state of the art in automatic verification of high-level specifications of systems with the three dimensions of process behaviour, data, and real-time — beyond the capabilities of timed automata. In the first phase of R1, the core activities comprised the development of a system specification language, an approach to the automatic verification of real-time properties, and the application to the case study ETCS. As system specification language, CSP-OZ-DC (combining subsets from Communicating Sequential Processes, Object-Z, and Duration Calculus) was developed [18,16]. A key result in this development was a compositional semantics on the basis of *Phase Event Automata* (PEA), an extension of timed automata to represent data [16]. It involves a translation of the DC subsets of counterexample formulas (with events) and so-called *test formulas* into equivalent PEA. It was shown that PEA can be translated into *Transition Constraint Systems* (TCS), which serve as input for the *abstraction refinement model checker* ARMC [29] and the deductive *slicing abstraction* model checker SLAB [4]. While ARMC is based on predicate abstraction, SLAB is a combination of deductive model checking (based on Craig interpolation) and slicing. Both tools call *decision procedures* when checking entailment of constraints [13,36,37] as well as methods for computing interpolants [35,32]. By combining CSP-OZ-DC with ARMC (or SLAB) and decision procedures, properties of systems with both real-time constraints and (certain) infinite data types can be verified automatically, as demonstrated by case studies [17]. In particular, real-time properties of *emergency messages* in the ETCS case study were verified [23,11,22].

These core activities were complemented by research into reducing the size of the state spaces of specifications with the help of *slicing techniques*. This approach has been applied both at the level of CSP-OZ-DC [5,3] and at the level of TCS [4].

This paper is organized as follows. In Section 2 the combined specification language CSP-OZ-DC is outlined. In Section 3 an approach to automatic verification of real-time properties of CSP-OZ-DC specification is presented. Section 4 reports on case studies preformed in this setting. Section 5 describes the tool support available for the approach, and Section 6 concludes the paper.

2 Combined Specifications

To specify real-time systems with a rich data part, we developed a high-level system specification language called CSP-OZ-DC [18,16] which separates the aspects of process behavior, data, and time. CSP-OZ-DC combines concepts from Communicating Sequential Processes (CSP) [14,31], Object-Z (OZ) [7,33], and the Duration Calculus (DC) [41,40]. The central notion is that of a *class*, consisting of an interface, a CSP part, an OZ part, and a DC part. An example of (part of) a class is shown Fig. 2. It models the *rear train* in the ETCS for the case of two trains.

The idea is that the rear train measures its position periodically and adjusts its speed so that is is always able to brake safely before reaching the limit of authority (LOA) along the track by applying its service brake. For this purpose, it periodically computes the *service brake intervention limit* (SBI), which represents the last position at which the train can apply the service brake in order to stop before the current end of authority.

The class has three *parameters*: the identity *ID* of the train, the position *StartPos* of the train at its start, and the position *StartSBI* of the SBI at its start. The *interface* declares channel names and types used by the class. Here we see the channels *updPos* for updating the train's position and *compSBI* for computing the next SBI.

The *CSP part* constrains the sequencing of events (communication) along the interface channels using CSP process notation. It may consist of multiple processes defined by CSP process equations, one of which is a distinguished process named `main`, which denotes the initial process. Here we see that the CSP part consists of two subprocesses working in parallel (denoted by $|||$). The subprocess *Running* is taking care of the normal operation of speed control and the subprocess *HandleEM* takes care of emergency situations. The subprocess *Running* first inputs the current position *pos* of the train and the current limit of authority *loa* and then computes the SBI in the state variable *sbi*. If the position *pos* of the train has got beyond the value *sbi* the service brake has to be applied.

The *OZ part* specifies the state space and operations upon it. It comprises a nameless *state* schema describing the state space, a schema `Init` constraining the initial state, and communication schemas `com_`c describing the transformation of the state space induced by communicating along an interface channel c. Here we see that the state contains two variables *sbi* (for the current SBI) and *curPos* (for the current position) of type *Position* and *curSpd* (for the current speed) of type *Speed*. Of the communication schemas we exhibit the one for the channel *compSBI* which specifies how the new value of the variable *sbi* is defined. The notation $\Delta(sbi)$

$$\mathit{RearTrain}(\mathit{ID} : \mathit{TrainID};\ \ \mathit{StartPos},\ \mathit{StartSBI} : \mathit{Position})$$

$$\mathtt{chan}\ \mathit{updPos} : [\mathit{id} : \{\mathit{ID}\},\ \mathit{pos}! : \mathit{Position}]$$

$$\mathtt{chan}\ \mathit{compSBI} : [\mathit{loa}?,\ \mathit{sbi}! : \mathit{Position}]$$

$$\ldots$$

$$\mathtt{main} \overset{c}{=} \mathit{Running} \,|||\, \mathit{HandleEM}$$

$$\mathit{Running} \overset{c}{=} \mathit{updPos}.\mathit{ID}\,?\,\mathit{pos} \rightarrow \mathit{getLOA}.\mathit{ID}\,?\,\mathit{loa} \rightarrow \mathit{compSBI}\,!\,\mathit{loa}\,?\,\mathit{sbi} \rightarrow$$

$$\textbf{if}\ \ \mathit{sbi} \leq \mathit{pos}\ \ \textbf{then}\ \ldots\ \textbf{else}\ \ldots$$

$$\ldots$$

$$\begin{array}{|l}
\hline
\mathit{sbi} : \mathit{Position} \\
\mathit{curPos} : \mathit{Position} \\
\mathit{curSpd} : \mathit{Speed} \\
\ldots \\
\hline
\end{array}$$

$$\begin{array}{|l}
\mathtt{com\_}\mathit{compSBI} \\
\hline
\Delta(\mathit{sbi}) \\
\mathit{loa}?,\ \mathit{sbi}! : \mathit{Position} \\
\hline
\mathit{sbi}' = \mathit{loa}? - \mathit{TargetSpdDist} - \mathit{StopDist} - \mathit{MaxDist} \\
\mathit{sbi}! = \mathit{sbi}' \\
\hline
\end{array}$$

$$\ldots$$

$$\neg \Diamond (\updownarrow \mathit{updPos}\ ;\ \ \ell < \mathit{updBound}\ ;\ \updownarrow \mathit{updPos})$$

$$\ldots$$

Fig. 2. A class in CSP-OZ-DC

of Object-Z defines that *sbi* is the only state variable changed by the schema.

The *DC part* constraints the timing of states and events. Here we see a so-called counterexample formula that states a lower time bound: any two successive communication events on the channel *updPos* should *not* be less than *updBound* seconds apart.

The ETCS with two trains consists of several classes. Besides *RearTrain* there are the classes for the *LeadingTrain*, the *RBC*, the *CommunicationNetwork* (between trains and RBC), the *Track*, and the *Driver* [22]. Objects of classes may be combined into systems using the CSP operators of parallel composition and renaming.

2.1 UML Profile

Often specifiers use diagrams to support their understanding of a system. To facilitate this, a *UML profile* in the notations of UML 2.0 [39] has been developed for CSP-OZ [25] and extended to CSP-OZ-DC. The profile comprises *class diagrams*, *protocol state machines*, and *component diagrams*. These diagrams are annotated by suitable *tags* to represent the contents of classes in the form of Z and DC expressions. The semantics of the UML profile is given by a translation of the profile into CSP-OZ-DC as illustrated by Fig. 3. For details of the CSP-OZ part we refer to the paper [25].

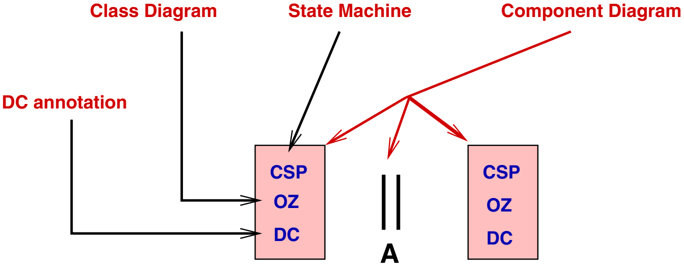

Fig. 3. Semantics of the UML profile for CSP-OZ-DC

2.2 Operational Semantics

The key for an automatic verification of CSP-OZ-DC is an operational semantics of this language defined by J. Hoenicke on the basis of Phase Event Automata (PEA) [16]. PEA extend timed automata [1] such that the parallel composition synchronizes on both phases (state formulae) and events. This permits the a *compositional* semantics definition for CSP-OZ-DC, i.e., one satisfying the equation

$$\mathcal{A}(CSP\text{-}OZ\text{-}DC) = \mathcal{A}(CSP) \parallel \mathcal{A}(OZ) \parallel \mathcal{A}(DC)$$

where $||$ denotes the (synchronous) parallel composition of PEA $\mathcal{A}(...)$. In fact, $\mathcal{A}(DC)$ decomposes even further into a parallel composition of PEA for each individual timing constraint in the DC part. An important property of this semantics is that whenever a subset of PEA in a parallel composition satisfies a requirement (represented as a DC formula) then also the full parallel composition does. This allows for a cone-of-influence verification technique.

For the DC part the class of *counterexample formulae* (with facilities to constrain the occurrences of both state changes and communication events) was introduced, extending the well-known class of “DC implementables” by A.P. Ravn [30]. The main theorem proved by J. Hoenicke in [16] is that every counterexample formula F has an operational semantics in form of a *deterministic* PEA $\mathcal{A}(F)$ such that the runs of $\mathcal{A}(F)$ are equivalent to the DC interpretations of F. The proof of this theorem uses a so-called *powerset construction* to cope with the nondeterminism arising from overlapping phases in F. Overlapping phases allow for concise specifications. The determinism of $\mathcal{A}(F)$ permits an easy treatment of negation, which underlies the specification with counterexample formulae as well as an automata-theoretic approach to model checking DC. In the latter approach, the desired property is negated and then represented as a PEA running in parallel to the system.

For the real-time requirements, R. Meyer extended the class of translatable formulae even further to so-called *test formulae* [21]. Whereas counterexample formulae are negated traces (of timed phases), test formulae contain arbitrary Boolean combinations of such traces and are closed under disjunction, conjunction, and the DC chop. Using so-called *sync events*, test formulae can be brought into a disjunctive normal form over traces and their negations, which facilitates their translation into PEA. To date, test formulae are the largest class of DC formulae that have an equivalent operational semantics in terms of automata [24].

Figure 4 shows the automaton $\mathrm{PEA}_{\mathrm{OZ}}$ representing the semantics of the OZ part of the class in Fig. 2. Here *updPos* and *comSBI* are Boolean variables representing the presence or absence of a communication event on the corresponding channel in the CSP part.

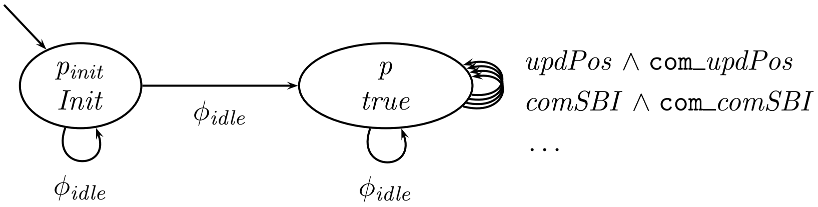
p_{init}
$Init$
ϕ_{idle}
p
$true$
ϕ_{idle}
ϕ_{idle}
$updPos \wedge \mathtt{com\_}updPos$
$comSBI \wedge \mathtt{com\_}comSBI$
$\ldots$

Fig. 4. Phase Event Automaton $\mathrm{PEA}_{\mathrm{OZ}}$ for the OZ part of the class in Fig. 2

Note that for each communication schema in the OZ part there is a corresponding transition labeled with the Boolean event variable and the formula of the communication schema, here $\mathtt{com\_}updPos$ and $\mathtt{com\_}comSBI$. The *idling transition* ϕ_{idle} is taken if none of the communications in the OZ part of the class is enabled. Here ϕ_{idle} abbreviates the formula

$$\phi_{idle} \;\Leftrightarrow\; \neg\, updPos \wedge \neg\, comSBI \wedge \ldots \wedge sbi = sbi' \wedge curPos = curPos' \wedge \ldots$$

Figure 5 shows the automaton $\mathrm{PEA}_{\mathrm{DC}}$ representing the semantics of the counterexample formula

$$\neg \Diamond(\, \updownarrow updPos \,;\; \ell < 5 \,;\; updPos \,)$$

in the DC part of the class in Fig. 2. Here c is a clock that is used to measure the duration of 5 seconds.

3 Automatic Verification

We consider the problem whether a given specification *CSP-OZ-DC* satisfies a real-time requirement expressed by a DC formula. The aim is an automatic verification method. Our approach is illustrated by the following scheme:

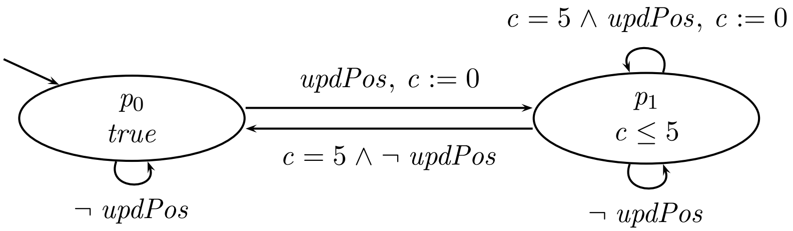
$c = 5 \wedge \mathit{updPos},\ c := 0$
$\mathit{updPos},\ c := 0$
p_0
true
p_1
$c \leq 5$
$c = 5 \wedge \neg\ \mathit{updPos}$
$\neg\ \mathit{updPos}$
$\neg\ \mathit{updPos}$

Fig. 5. Phase Event Automaton PEA_{DC} for the DC part of the class in Fig. 2

$$\begin{tikzcd}
 & \mathit{CSP\text{-}OZ\text{-}DC} \arrow[d] & \text{satisfies} & \mathit{DC}\ ? \arrow[d] \\
\text{PEA:} & \mathcal{A}(\mathit{CSP}) \parallel \mathcal{A}(\mathit{OZ}) \parallel \mathcal{A}(\mathit{DC}) & \parallel & \mathcal{A}_{test}(\mathit{DC}) \\
 & \text{Is the } \textit{bad state} \text{ of } \mathcal{A}_{test}(\mathit{DC}) \text{ reachable ?} \arrow[d] & & \\
\text{TCS:} & \mathcal{T}(...) & &
\end{tikzcd}$$

In order to check whether a specification *CSP-OZ-DC* satisfies a real-time property, represented by a test formula *DC*, both the specification and the property are translated to Phase Event Automata running in parallel. The property *DC* is translated to a so-called *test automaton* $\mathcal{A}_{test}(\mathit{DC})$, which has a distinguished *bad state* such that specification *CSP-OZ-DC* satisfies the test formula *DC* if and only if at the PEA level the bad state is reachable in $\mathcal{A}_{test}(\mathit{DC})$ as part of the overall parallel composition.

To check for reachability we apply the *abstraction refinement model checker* ARMC developed by A. Podelski and A. Rybalchenko [28,29]. ARMC takes as input Transition Constraint Systems (TCS). The PEA semantics of CSP-OZ-DC is very well suited as an intermediate language in the translation process from CSP-OZ-DC down to TCS. At the level of TCS, the clocks of PEA are represented as real-valued data variables, following the “old-fashioned recipe” advocated by L. Lamport. As an example consider the transition constraint system $\mathcal{T}(\text{PEA}_{\text{DZ}})$ for the automaton PEA_{DZ} for the DC constraint shown in Fig. 5:

$$\begin{array}{lll}
\mathcal{T}(\text{PEA}_{\text{DZ}}) & \Leftrightarrow & \mathit{ph} = 0 \wedge \neg \mathit{updPos} \wedge c' = c + \text{len} \wedge \mathit{ph}' = 0 \\
 & \vee & \mathit{ph} = 0 \wedge \mathit{updPos} \wedge c' = \text{len} \wedge c' \leq 5 \wedge \mathit{ph}' = 1 \\
 & \vee & \mathit{ph} = 1 \wedge \neg \mathit{updPos} \wedge c' = c + \text{len} \wedge c' \leq 5 \wedge \mathit{ph}' = 1 \\
 & \vee & \mathit{ph} = 1 \wedge \mathit{updPos} \wedge c = 5 \wedge c' = \text{len} \wedge c' \leq 5 \wedge \mathit{ph}' = 1 \\
 & \vee & \mathit{ph} = 1 \wedge \neg \mathit{updPos} \wedge c = 5 \wedge c' = c + \text{len} \wedge \mathit{ph}' = 0
\end{array}$$

Here c is a real-valued variable representing the corresponding clock of PEA_{DZ} and len is a real-valued variable with the constraint len > 0 that represents time progress. The variables *ph* represents the current *phase* of PEA_{DZ}.

3.1 Abstraction Refinement

Verification of temporal safety and liveness properties can be effectively automated by applying a reduction to least fixpoint computation [6,27]. Such a fixpoint computation engine serves as a basis for the verification tool ARMC [28,29]. ARMC is a model checking tool that applies *abstraction refinement* to efficiently handle the high complexity of verification tasks envisaged in the AVACS project. Its distinguishing characteristics lie in the way it applies logical reasoning to deal with abstraction [29]. ARMC is implemented in a Prolog system together with Constraint Logic Programming extensions. *Interpolation* is an important component of the abstraction refinement algorithm used by ARMC. It provides an effective means for computing the separation between the sets of ‘good’ and ‘bad’ states. ARMC uses an algorithm for the generation of interpolants for the combined theory of linear arithmetic and uninterpreted function symbols [32]. It uses a reduction of the problem to constraint solving in linear arithmetic, which allows for the application of existing highly optimized Linear Programming solvers in black-box fashion.

Note that the PEA and hence the TCS representing the semantics of CSP-OZ-DC specifications are in general infinite state systems due to both clocks and data values. So reachability is in general not decidable. Thus the fixpoint computation of ARMC need not terminate. However, as our case studies demonstrate, ARMC can be applied successfully to various examples.

4 Case Studies

A first application of this approach to verification dealt with a *parametric elevator* by J. Hoenicke and P. Maier [17]. In this example the number of floors are treated as parameters. A safety property that depended on all parts of the specification (i.e., communication, data, and time) was verified automatically with ARMC. The specification of the elevator in CSP-OZ-DC comprised both infinite data (i.e., integers representing an arbitrary number of floors) and continuous real-time.

4.1 Emergency Messages

The benchmark for the project “R1: Beyond Timed Automata” was defined as the verification of *timing requirements for the radio communication* between trains and the radio block center (RBC) in the ETCS. Starting from a comprehensive but informal description of the ETCS in [9], J. Faber defined the case study *Emergency Messages* (EM) for the scenario where an RBC controls consecutive trains on a track segment (see Fig. 1). In the case of two trains, if the first train detects an emergency situation it immediately applies the emergency brake and sends an emergency message via the radio connection to the RBC, which has to inform the follower train within a predefined time interval. The train control system has to stop the follower such that no collision occurs. This property depends on several real-time requirements for the message transfer times and the reaction times of the RBC and the follower.

In [10,12,24], this case study is modeled in the specification language CSP-OZ-DC. The model involves continuous real time, *real-valued* variables representing train positions (on an infinite track segment) and speeds and messages transferred via CSP channels. Other quantities like the length of the train and the braking distance were treated as *parameters*. A part of one class of the specification is shown in Fig. 2. Then the techniques of Section 2 [17,16] were applied to translate the CSP-OZ-DC model via Phase Event Automata (PEA) into Transition Constraint Systems (TCS) that are the input of the abstraction refinement model checker ARMC [28,29].

For the case study EM, properties formalizing *reaction times* in the communication between trains and RBC could be verified automatically with ARMC. Thus the benchmark for R1 was fully achieved.

However, the global property of *collision freedom* could not be verified automatically with ARMC. The reasons are as follows. According to [16] each CSP-OZ-DC specification is represented as a parallel composition of PEA. In the case study this composition consisted of 18 automata. At present ARMC requires the parallel product of this composition to be computed. An attempt to compute the parallel product of all 18 automata of EM failed due to memory shortage. By contrast, for the verification of the reaction times in the benchmark case it is sufficient to consider only 5–7 of the automata and compute their parallel product. By the compositional semantics [16] of CSP-OZ-DC, this allows us to infer the verified property for the full parallel composition of all automata (without computing the product).

To prove collision freedom for the EM case study, a *manual* decomposition of this property into simpler subtasks was performed. Each of these subtasks was a variant of a reaction time property that could be verified automatically with ARMC.

Moreover, variants of the case study EM were examined which had a more sophisticated and realistic data part, but a less complicated control structure: the RBC maintains an *array* of consecutive trains (on an infinite track segment) where the size of the array is kept as a parameter. In case of an emergency, *every* train behind the emergency train has to be instructed to stop (Fig. 1). Message transfer times were not considered in this extended scenario. For this variant, collision freedom for an arbitrary number of trains [19,11] could be shown. To cope with the data type used for representing the train positions (in this case: arrays with integer elements and real numbers as elements, with a parametric dimension) methods for hierarchical reasoning in theories of complex data types developed in [34] were employed.

5 Tool Support

Tool support has been developed to handle system specifications expressed in CSP-OZ-DC, real-time requirements in form of *test formulae*, and their translation into Phase Event Automata (PEA) and Transition Constraint Systems (TCS).

5.1 Syspect

For CSP-OZ-DC specifications, a graphical modeling environment called *Syspect* (System Specification Tool) [38] has been implemented on the basis of the Eclipse platform [8]. An overview of Syspect is given in Fig. 6. The graphic modeling uses the UML profile mentioned in Subsection 2.1. See Fig. 7 for a screen shot of the class editor. The graphic model is automatically converted into an internal representation of CSP-OZ-DC, which offers the possibility to export the specification into their semantic model in terms of PEA for a subsequent verification. To this end, Syspect also permits to enter real-time requirements expressed as *test formulae* (see Section 2). Test formulae also serve as the slicing criterion in a *Slicing Plugin* of Syspect that has been implemented to perform slicing of CSP-OZ-DC specifications in order to reduce their size [5,3].

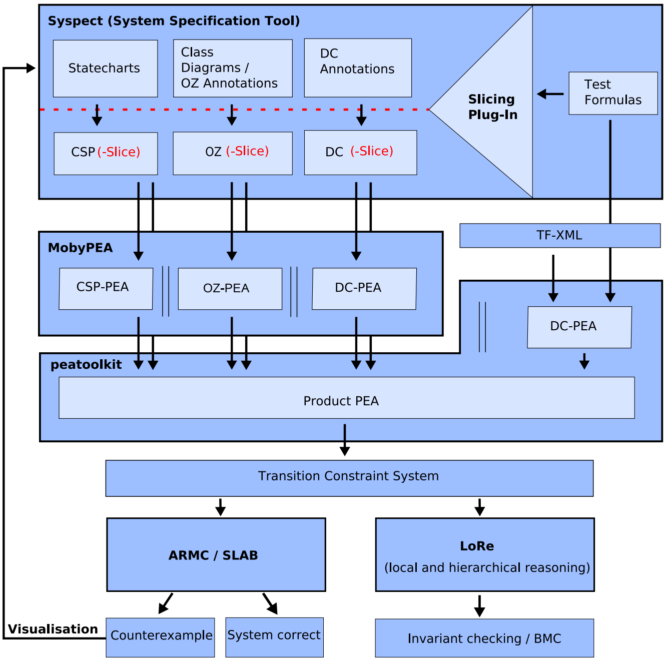

Fig. 6. Overview of tool support

Since verification is based on the transformation of PEA into TCS, the so-called *PEA toolkit* [26] provides an automatic computation of the parallel product of PEA and an automatic translation of PEA into TCS, the input representation for both model checkers developed in R1, namely ARMC [29] as well as SLAB [4]. The latter integrates of slicing techniques with abstraction mechanisms. Moreover, counterexample traces produced by ARMC can be automatically traced back to the given high-level CSP-OZ-DC specification and visualized in the Syspect tool [15].

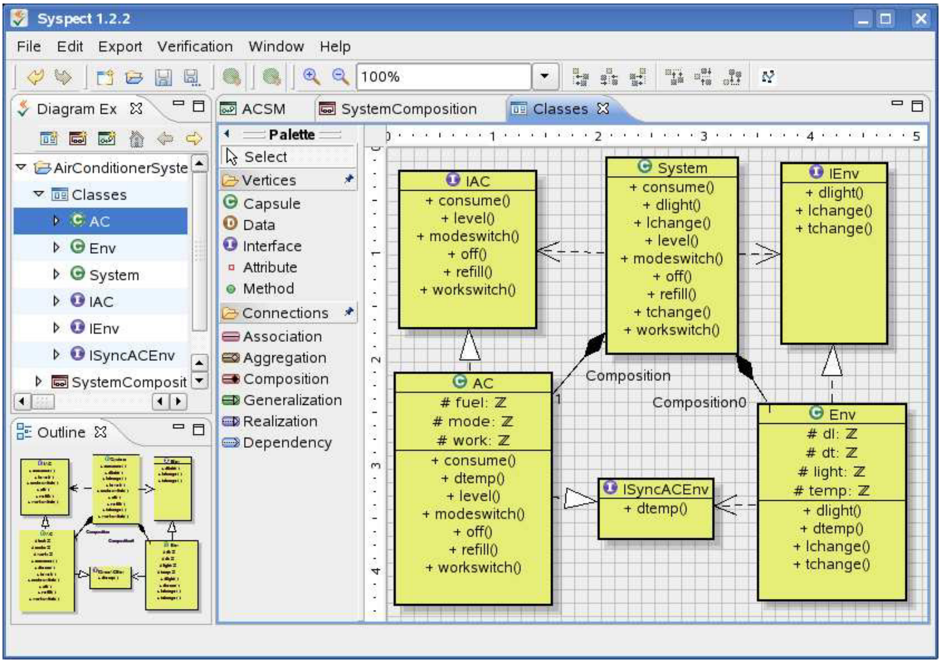

Fig. 7. Screen shot of Syspect

6 Conclusion

We have explained how real-time properties of systems specified in the combination CSP-OZ-DC can be automatically verified using recent advances in semantics, constraint-based model checking, and decision procedures for complex data. The verification is based on the abstraction refinement model checker ARMC that can deal with variables ranging over continuous real-time and infinite data.

A shortcoming of the current version of ARMC is that it cannot exploit the parallel composition that is present in the Phase Event Automata (PEA). Since ARMC expects as input Transition Constraint Systems (TCS) in disjunctive normal form, the parallel product of PEA, which corresponds to the conjunction of TCS, has to be computed before it can be handled by ARMC. For the full benchmark case study this leads to state spaces that are too large to be computed (see Section 4). This shortcoming will be addressed in the future work of the subproject R1.

References

[1] Alur, R. and D. Dill, *A theory of timed automata*, Theoret. Comp.Science **126** (1994), pp. 183–235.

[2] Becker, B., A. Podelski, W. Damm, M. Fränzle, E.-R. Olderog and R. Wilhelm, *SFB/TR 14 AVACS – automatic verification and analysis of complex systems*, it – Informatiton Technology **49** (2007), pp. 118–126, see also http://www.avacs.org.

[3] Brückner, I., *Slicing concurrent real-time system specifications for verification*, in: J. Davies and J. Gibbons, editors, *Integrated Formal Methods (IFM)*, LNCS **4591** (2007), pp. 54–74.

[4] Brückner, I., K. Dräger, B. Finkbeiner and H. Wehrheim, *Slicing abstractions*, in: *Proceedings of the International Symposium on Fundamentals of Software Engineering (FSEN)*, 2007, accepted for publication.

[5] Brückner, I., B. Metzler and H. Wehrheim, *Optimizing slicing of formal specifications by deductive verification*, Nordic Journal of Computing **13** (2006), pp. 22–45.

[6] Cousot, P. and R. Cousot, *Abstract interpretation: a unified lattice model for the static analysis of programs by construction or approximation of fixpoints*, in: *Proc. POPL* (1977), pp. 238–252.

[7] Duke, R., G. Rose and G. Smith, *Object-Z: A specification language advocated for the description of standards*, Computer Standards and Interfaces **17** (1995), pp. 511–533.

[8] *Eclipse Foundation Inc., Homepage of the Eclipse Community* (2005), http://www.eclipse.org.

[9] ERTMS User Group, UNISIG, *ERTMS/ETCS System requirements specification, version 2.2.2* (2002), http://www.aeif.org/ccm/default.asp.

[10] Faber, J., *Verifying real-time aspects of the European Train Control System*, in: *Proceedings of the 17th Nordic Workshop on Programming Theory* (2005), pp. 67–70.

[11] Faber, J., S. Jacobs and V. Sofronie-Stokkermans, *Verifying CSP-OZ-DC specifications with complex data types and timing parameters*, in: J. Davies and J. Gibbons, editors, *Integrated Formal Methods*, LNCS **4591** (2007), pp. 233–252.

[12] Faber, J. and R. Meyer, *Model checking data-dependent real-time properties of the european train control system*, in: *Proc. of the Conf. on Formal Methods in Computer Aided Design (FMCAD)* (2006), pp. 76–77.

[13] Ganzinger, H., V. Sofronie-Stokkermans and U. Waldmann, *Modular proof systems for partial functions with Evans equality*, Information and Computation **204** (2006), pp. 1453–1492.

[14] Hoare, C. A. R., "Communicating Sequential Processes," Prentice Hall, 1985.

[15] Hobelmann, U., "Verifying Properties of Processes, Data, and Time: Linking Counterexamples to High-Level Specifications," Master's thesis, University of Oldenburg (2007).

[16] Hoenicke, J., "Combination of Processes, Data, and Time," Ph.D. thesis, Report Nr. 9/2006, University of Oldenburg (2006).

[17] Hoenicke, J. and P. Maier, *Model-checking of specifications integrating processes, data and time*, in: J. Fitzgerald, I. Hayes and A. Tarlecki, editors, *FM 2005*, LNCS **3582** (2005), pp. 465–480.

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

[19] Jacobs, S. and V. Sofronie-Stokkermans, *Applications of hierarchical reasoning in the verification of complex systems*, in: *Proc. of the Fourth Intern. Workshop on Pragmatics of Decision Procedures in Automated Reasoning*, 2006, pp. 15–26.

[20] Larsen, K., P. Petterson and Wang Yi, *Uppaal in a nutshell*, Software Tools for Technology Transfer **1** (1997), pp. 134–152.

[21] Meyer, R., "Model-Checking von Phasen-Event-Automaten bezüglich Duration Calculus Formeln mittels Testautomaten," Master's thesis, University of Oldenburg (2005).

[22] Meyer, R., J. Faber, J. Hoenicke and A. Rybalchenko, *Model checking duration calculus: A practical approach*, Formal Aspects of Computing (2008), accepted for publication.

[23] Meyer, R., J. Faber and A. Rybalchenko, *Model checking duration calculus: A practical approach*, in: K. Barkaoui, A. Cavalcanti and A. Cerone, editors, *Theoretical Aspects of Computing (ICTAC)*, LNCS **4281** (2006), pp. 332–346.

[24] Meyer, R., J. Faber and A. Rybalchenko, *Model checking duration calculus: A practical approach*, in: K. Barkaoui, A. Cavalcanti and A. Cerone, editors, *3rd International Colloquium on Theoretical Aspects of Computing, ICTAC*, LNCS **4281** (2006), pp. 332–346.

[25] Möller, M., E.-R. Olderog, H. Rasch and H. Wehrheim., *Integrating a formal method into a software engineering process with uml and java*, Formal Aspects of Computing (2007), accepted for publication.

[26] *PEA toolkit*, Department of Computing Science, University of Oldenburg, http://csd.informatik.uni-oldenburg.de/projects/pea.html (2006).

[27] Podelski, A. and A. Rybalchenko, *Transition invariants*, in: *LICS'2004: Logic in Computer Science* (2004), pp. 32–41.

[28] Podelski, A. and A. Rybalchenko, *Transition predicate abstraction and fair termination*, in: J. Palsberg and M. Abadi, editors, *Proc. of the 32nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL)* (2005), pp. 132–144.

[29] Podelski, A. and A. Rybalchenko, *ARMC: the logical choice for software model checking with abstraction refinement*, in: M. Hanus, editor, *PADL'2007: Practical Aspects of Declarative Languages*, LNCS **4354** (2007), pp. 245–259.

[30] Ravn, A., *Design of embedded real-time computing systems*, Technical Report ID-TR 1995-170, Technical University of Denmark (1995).

[31] Roscoe, A. W., "Theory and Practice of Concurrency," Prentice-Hall, Englewood Cliffs, NJ, 1998.

[32] Rybalchenko, A. and V. Sofronie-Stokkermans, *Constraint solving for interpolation*, in: B. Cook and A. Podelski, editors, *8th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2007)*, LNCS **4349** (2007), pp. 346–362.

[33] Smith, G., "The Object-Z Specification Language," Kluwer Academic Publisher, 2000.

[34] Sofronie-Stokkermans, V., *Hierarchic reasoning in local theory extensions*, in: R. Nieuwenhuis, editor, *20th International Conference on Automated Deduction (CADE-20)*, Lecture Notes in Artificial Intelligence **3632** (2005), pp. 219–234.

[35] Sofronie-Stokkermans, V., *Interpolation in local theory extensions*, in: U. Furbach and N. Shankar, editors, *Automated Reasoning: Third Intern. Joint Conf. (IJCAR)*, LNCS **4130** (2006), pp. 235–250.

[36] Sofronie-Stokkermans, V., *Hierarchical and modular reasoning in complex theories: The case of local theory extensions*, in: B. Konev and F. Wolter, editors, *Proc. of the 6th Intern. Symp. on Frontiers of Combining Systems (FroCos)*, LNCS **4720** (2007), pp. 47–71.

[37] Sofronie-Stokkermans, V. and C. Ihlemann, *Automated reasoning in some local extensions of ordered structures*, in: *Proc. of the 37th Intern. Symp. on Multiple-Valued Logics (ISMVL)* (2007).

[38] *Syspect – System Specification Tool*, Department of Computing Science, University of Oldenburg, http://csd.informatik.uni-oldenburg.de/~syspect (2006).

[39] *OMG Unified Modeling Language: Superstructure, version 2.0 – final adopted specification* (2003), http://www.omg.org.

[40] Zhou, C. and M. R. Hansen, "Duration Calculus: A Formal Approach to Real-Time Systems," Springer, 2004.

[41] Zhou, C., C. A. R. Hoare and A. P. Ravn, *A calculus of durations*, Inform. Proc. Letters **40** (1991), pp. 269–276.