I. INTRODUCTION

The behavior of embedded hardware and software systems is determined by at least three dimensions: control flow, data aspects, and real-time requirements. To specify the different dimensions of a system with the best-suited techniques, the formal language CSP-OZ-DC [1] integrates Communicating Sequential Processes (CSP) [2], Object-Z (OZ) [3], and Duration Calculus (DC) [4] into a declarative formalism equipped with a unified and compositional semantics. In this paper, we provide evidence that CSP-OZ-DC is a convenient language for modeling systems of industrial relevance. To this end, we examine the emergency message handling in the European Train Control System (ETCS) [5] as a case study with uninterpreted constants and infinite data domains. We automatically verify that CSP-OZ-DC ensures real-time safety properties, which crucially depend on the system’s data handling.

Related work on ETCS case studies focuses on stochastic examinations of the communication reliability [6], [7]. The components’ data aspects are neglected, though.

II. A CSP-OZ-DC MODEL OF THE ETCS

In this section, we introduce the case study and present its CSP-OZ-DC model in more detail.

The ETCS [5] aims at replacing national train control systems with the goal of ensuring cross-border interoperability and improving railway safety as well as track utilization. In the final ETCS implementation level, trains communicate over a radio connection with radio block centers (RBCs). Controlling the traffic in well-defined areas, the RBCs grant movement authorities to trains. In order to increase the traffic density, movement authorities are given up to a position—called limit of authority (LOA)—closely behind the preceding train. In case of an accident, the train control system has to stop all trains safely. Focusing on the emergency message handling, we ensure in our case study that the trains never collide.

In our CSP-OZ-DC model the case study’s components are specified in an object-oriented way using classes. To give an idea of both the specification language and the case study, we introduce the class Train in more detail. Every CSP-OZ-DC class comprises an interface part defining typed channels that can be used for the inter-class communication.

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) and the Graduate School “TrustSoft” (GRK 1076/1).

Train(ID : TrainID)

\[
\begin{align*}
\text{chan} & \text{send, receive} : \{m : \text{Message}, id : \text{TrainID}\} \\
\text{chan} & \text{computeSBI} : \{\text{loa}, \text{sbi} : \text{Position}\} \\
\text{local} & \text{selectSpeed, applyEB, applySB} \\
\end{align*}
\]

The interface part of the class Train declares the channels send and receive for communication with the RBC. These channels carry two values indicating the kind of message and the ID of the sending train. In addition, we define the channel computeSBI carrying two position values. It computes the service brake intervention limit (SBI), i.e., the last position on the track the train has to apply the service brakes to halt before reaching the LOA. If the train exceeds this point the emergency brakes have to be used. The remaining channels (speed or brake instructions) are declared as local.

The external and internal behavior of CSP-OZ-DC components is described with CSP processes [2]. The processes communicate by sending data values over channels, respecting the channel declarations in the interface part. We call the occurrence of a communication an event. The process Run in the CSP part of Train determines a train’s running behavior.

\[
\begin{align*}
\text{Run} \subseteq & \text{updatePosition}\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!.\!\!
An initial schema (not depicted here) constrains the attributes’ initial values. Operation schemata associated with channels specify data changes that are performed at the moment the CSP part communicates over the channel. The operation schema for computeSBI contains a Δ expression indicating that the sbi attribute is changed by this schema. The input loa? is provided by the environment and contains the LOA position. The new value for sbi is represented by the primed variable sbi’, the return value by sbi! . These values are calculated depending on loa? and the distance StopDist the train needs to stop when exceeding the SBI.

We use an uninterpreted constant for the distance StopDist, i.e., this value is restricted by constraints, but not defined explicitly. Furthermore, our case study comprises variables of infinite data types like position and speed. We model the domains as reals: Position == R, Speed == R+. The values of these variables are also transferred via channels.

Real-time constraints are described using the dense real-time logic DC [4]. Since we aim at automatic verification, we use the subclass of counterexample-trace formulae [1].

\[
\neg \left( \text{true} ; [\text{receive.warning.ID} ; 0.5 < t \land \Box \text{applyEB}] \right) \\
\ldots
\]

The given DC formula states that after receiving a warning event (\[\text{true}\]) we do not allow an interval greater than 0.5 time units (0.5 < \(t\)) to follow without (\(\Box\)) an applyEB event.

III. Verification of the Case Study

The formal semantics of CSP-OZ-DC is given in terms of timed automata extended with data variables, so-called phase event automata (PEA). Each part of a CSP-OZ-DC specification is translated into a single PEA [1]. A distinguished parallel composition is defined for PEA such that they synchronize on both events and data variables. This operation allows for a compositional semantics and it guarantees that once a safety property holds for one PEA it also holds for every parallel composition with this PEA.

In order to check a (safety) property given as DC formula for a CSP-OZ-DC model, we use the automata theoretic approach of Vardi and Wolper [8]. We first compute the PEA semantics of the model. The given DC formula, that has to be a counterexample-trace formula, is automatically transformed (the tool is available on [9]) into a set of test automata, i.e., PEA with distinguished bad states. The model satisfies the formula on an interval iff a bad state is reachable in the parallel composition of the PEA semantics and the test automata (cf. [10]). To cope with infinite data types and uninterpreted constants, we apply the abstraction refinement model checker ARMC [11] to check reachability in the product automaton.

Our aim is to ensure that two trains never collide. We express this property in the DC formula \(\neg \left( \text{true} ; [\text{position}_1 > \text{position}_2 - \text{Length}] \right)\). It shows that the property crucially depends on the trains’ data behavior. The model of the full case study is too large to verify the property in a single step. However, we benefit from the compositionality of the CSP-OZ-DC semantics and verify local properties for the parallel components of our model. The compositional semantics ensures that those properties hold for the entire system.

Table I gives a subset of our experimental results for a range of verification tasks. It lists the numbers of program locations (Locs), transitions (Trans), variables (Vars), predicates generated by ARMC (Preds), abstract states (Abstr), refinements loops (Refs), runtimes in seconds for generating test automata and parallel product (TA), and for model checking (ARMC). For instance, we consider the running behavior of the train in isolation and verify the safety property stated above, assuming that the first train does not apply the emergency brakes. To this end, we take only those PEA into account that influence the running behavior, i.e., the automata for the subprocess Run together with the appropriate automata for the OZ and the DC part. The result of this verification task is listed as “Run” in Table I. The table additionally lists the remaining verification tasks for the decomposition of our safety property.

From our case study, we experienced that decomposition of models as well as properties is necessary to handle industrial-scale systems. Therefore, the key prerequisite in our verification approach is the compositional semantics preserving the subcomponents’ properties for the full system. Our results also demonstrate that the automata theoretic approach presented here is well-suited to check large-scale models (>18000 transitions) having infinite data types and uninterpreted constants. This is important for properties that cannot be decomposed further and depend on a number of components. The separate modeling of control flow, data part, and real-time requirements helped us handle the complexity of our case study.

REFERENCES