A Component-based Approach to Verification of Embedded Control Systems using TLA$^+$

Ondrej Rysavy and Jaroslav Rab
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
Email: {rysavy,rabj}@fit.vutbr.cz

***Abstract*—The method for writing TLA$^+$specifications that obey formal model called Masaccio is presented in this paper. The specifications consist of components, which are built from atomic components by parallel and serial compositions. Using a simple example, it is illustrated how to write specifications of atomic components and components that are products of parallel or serial compositions. The specifications have standard form of TLA$^+$specifications hence they are amenable to automatic verification using the TLA$^+$model-checker.**

I. Introduction

Software running in embedded systems necessary acquires some properties of the physical world. Usually, these properties form a part of non-functional aspects in system requirements [1]. To model embedded software, these aspects must be considered by a specification method otherwise the model of a system easily diverges from the reality and becomes inapplicable in further refinement and analysis. Constructing large systems relies on effective and systematic application of modular approach. A large class of entities playing the role of building blocks that can be composed have been defined, most notably, classes for object-oriented program construction, components in hardware design, procedures and modules in procedural programming, and active objects and actors for reactive programming [2].

This paper deals with a method based on a formalism called Temporal Logic of Actions [3] that enables to describe embedded control software in a modular manner and apply an automatized model-checker tool to verify required properties of a specification. The main contribution of this paper lies in demonstration of how the TLA$^+$specifications whose interpretation is that of a formal model called Masaccio[4] can be written in a systematic way. The formal model permits to construct a hierarchical definition of components that are built from atomic components using operations of parallel composition, serial composition, renaming of variables, renaming of locations, hiding of variables, and hiding of locations. As the resulting TLA$^+$specifications have the form of a conjunction of initial predicate and next-state actions, they are readily explorable by the TLA$^+$explicit model-checker.

The research has been supported by the Czech Ministry of Education in the frame of Research Intentions MSM 0021630528, and by the Grant Agency of the Czech Republic through the grants GACR 201/07/P544 and GACR 102/08/1429.

II. Component Model

This section gives a brief overview of a formal model for embedded components as defined by Henzinger in [4]. In this paper, only discrete components are considered, although the proposed approach relies on the language that can be applied to hybrid systems [5] as well.

A fundamental entity of the model is a component. The component structures the system into architectural units that interact through defined interfaces. It is possible to structure components into a hierarchy that can be arbitrary nested to simplify the system design. The component A consists of definition of interface and internal behavior. An interface defines disjoint sets of input variables, V_A^{in}, output variables, V_A^{out}, and a set of public locations, L_A^{intf}. An execution of the component consists of a finite sequence of jumps. A jump is a pair $(p, q) \in [V_A^{in,out}] \times [V_A^{in,out}]$[1]. An observation p is called the source of jump (p, q) and an observation q is called the sink of jump (p, q). A jump v is successive to jump u if the source of jump v is equal to the sink of jump u. Formally, an execution of A is a pair (a, w) or a triple (a, w, b), where $a, b \in L_A^{intf}$ are interface locations and $w = w_0 \dots w_n$ is a nonempty, finite sequence of jumps of A such that every jump w_i, for $1 \leq i \leq n$ is successive to the immediately preceding step w_{i-1}. We write E_A for the set of executions of the component A.

An atomic component is the simplest form of components found in Masaccio. The behavior of the component is solely specified by its jump action. The interface of atomic component exploits input variables read by the component and output variables controlled by the component. The component $A(J)$ has two interface locations, $from$ and to; that is, $L_{A(J)}^{intf} = \{from, to\}$. The entry condition of $from$ is the projection of the jump predicate to the unprimed I/O variables. The entry condition of to is unsatisfiable.

Two components A and B can be combined to form a *parallel composition* $C = A \otimes B$ if the output variables of A and B are disjoint and for each interface location a common to both A and B, the entry conditions of a are equivalent in A and in B. The input variables of component $V_C^{in} = (V_A^{in} \setminus V_B^{out}) \cup (V_B^{in} \setminus V_A^{out})$. The output variables of the component are $V_C^{out} = V_A^{out} \cup V_B^{out}$. The interface

[1] $[V_A^{in,out}]$ stands for a set of all possible assigments of values into input and output variables of a component.

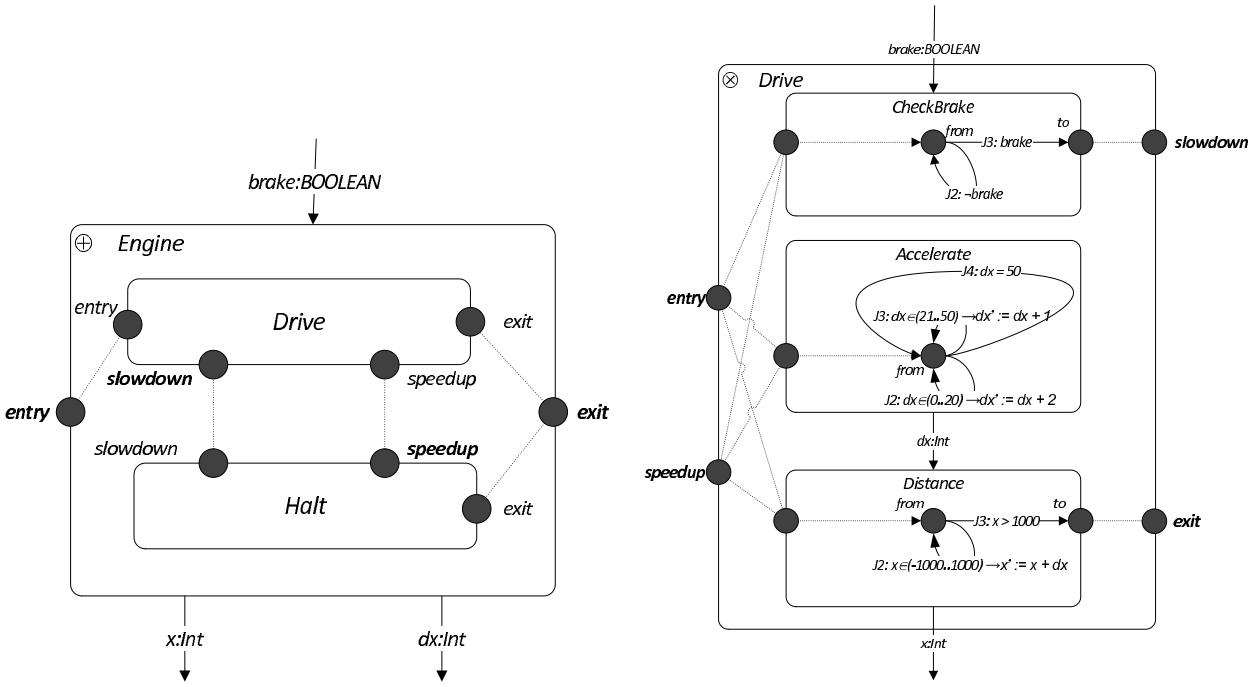

Fig. 1. The components Engine and Drive

locations of $A \otimes B$ are the interface locations of A together with the interface locations of B. An interface location a that is common to A and B and its entry conditions agree in both components has this entry condition also in $A \otimes B$. Other interface locations cannot be used to entry the component.

The definition of parallel composition specifies that each jump is done in synchronous manner in both subcomponents. Moreover, if one component reaches the exit interface location then the execution in the other component must be terminated. If both components reach their exit locations one is chosen nondeterministically. As the consequence of these properties, parallel composition operation is associative and commutative.

Two components A and B can be composed in series to form a *serial composition* $C = A \oplus B$ if the set of output variables are identical; that is, $V_A^{out} = V_B^{out}$. The input variables of composed component is $V^{in} = V_A^{in} \cup V_B^{in}$. The interface locations of $A \oplus B$ are the interface locations of A together with the interface locations of B. If a is an interface location of both A and B, then the entry condition of a in $A \oplus B$ is the disjunction of the entry conditions of a in the subcomponents A and B.

The set of execution of the component $C = A \oplus B$ contains 1) the pair (a, w) iff either $(a, w|_A)$ is an execution of A, or $(a, w|_B)$ is an execution of B, 2) the triple (a, w, b) iff either $(a, w|_A, b)$ is an execution of A, or $(a, w|_A, b)$ is an execution of B. The operator of serial composition is associative, commutative, and idempotent.

To support these two compositional operations, the renaming and hiding operations are defined for variables and locations. The renaming operation maps variables and locations of different names to each other that allows for sharing data and control between components. Hiding makes variables or locations internal to the component, which is useful when a complex behavior is modeled inside the component.

III. TLA$^+$

Temporal Logic of Actions (TLA) is a variant of linear-time temporal logic. It was developed by Lamport [3] primarily for specifying distributed algorithms, but several works shown that the area of application is much broader. The system of TLA+ extends TLA with data structures allowing for easier description of complex specification patterns. TLA+ specifications are organized into modules. Modules can contain declarations, definitions, and assertions by means of logical formulas. The declarations consist of constants and variables. Constants can be uninterpreted until an automated verification procedure is used to verify the properties of the specification. Variables keep the state of the system, they can change in the system and the specification is expressed in terms of transition formulas that assert the values of the variables as observed in different states of the system that are related by the system transitions. The overall specification is given by the temporal formula defined as a conjunction of the form $I \wedge \Box[N]_v \wedge L$, where I is the initial condition, N is the next-state relation (composed from transition formulas), and L is a conjunction of fairness properties, each concerning a disjunct of the next-state relation. Transition formulas, also called actions, are ordinary formulas of untyped first-order logic defined on a denumerable set of variables, partitioned into sets of flexible and rigid variables. Moreover, a set of primed flexible variables, in the form of v', is defined. Transition formulas then can contain all these kinds of variables to express a relation between two consecutive states. The generation of a transition system for the purpose of model checking verification or for the simulation is governed by the enabled transition formulas. The formula $\Box[N]_v$ admits system transitions that leave a set of

```
1  ──────────── MODULE Accelerate ────────────
2  EXTENDS Naturals
3  VARIABLES dx, clock, location
4  ──────────────────────────────────────────
5  J1 ≜  ∧ location = "from" ∧ location' = "from"
6        ∧ dx ∈ (0 .. 20) ∧ dx' = dx + 2
7        ∧ clock' = ¬clock
8
9  J2 ≜  ∧ location = "from" ∧ location' = "from"
10       ∧ dx ∈ (21 .. 49) ∧ dx' = dx + 1
11       ∧ clock' = ¬clock
12
13 J3 ≜  ∧ location = "from" ∧ location' = "from"
14       ∧ dx = 50 ∧ dx' = 50
15       ∧ clock' = ¬clock
16
17 Init ≜  ∧ dx ∈ (0 .. 50) ∧ clock ∈ BOOLEAN ∧ location = "from"
18
19 Next ≜  J1 ∨ J2 ∨ J3
20 ──────────────────────────────────────────
```

Fig. 2. The TLA$^+$ specification of component Accelerate

variables v unchanged. This is known as stuttering, which is a key concept of TLA that enables the refinement and compositional specifications. The initial condition and next-state relation specify the possible behaviour of the system. Fairness conditions strengthen the specification by asserting that given actions must occur. The TLA+ does not formally distinguish between a system specification and a property. Both are expressed as formulas of temporal logic and connected by implication $S \implies F$, where S is a specification and F is a property. Confirming the validity of this implication stands for showing that the specification S has the property F. The TLA+ is accompanied with a set of tools. One of such tool, the TLA+ model checker, TLC, is state-of-the-art model analyzer that can compute and explore the state space of finite instances of TLA+ models. The input to TLC consists of specification file describing the model and configuration file, which defines the finite-state instance of the model to be analysed. An execution of TLC produces a result that gives answer to the model correctness. In case of finding a problem, this is reported with a state-sequence demonstrating the trace in the model that leads to the problematic state. Inevitably, the TLC suffers the problem of state space explosion that is, nevertheless, partially addressed by a technique known as symmetry reduction allowing for verification of moderate size system specifications.

IV. Specification of Components

Using a simple example as required by space constraints, this section explains the construction of TLA$^+$specifications that corresponds to Masaccio embedded components.

An example represents a specification of component $Engine$ taken from [4]. This component is a part of a complex specification that models the control of a railway crossing. In particular, the $Engine$ component controls acceleration and deceleration of a train that is moving in a near distance to the railway crossing. Although this example is rather trivial, it is sufficient to demonstrate basic principles of the specification method as it contains both parallel and serial compositions.

The component $Engine$ and its subcomponents are visually modeled in figure 1. Components are represented by rectangles. Input and output variables are represented by arrows connected to component boundaries. Locations are represented by solid discs. Jump actions are represented by arrows. A jump is labeled with condition predicate and action predicate, which computes new values of output variables.

Component $Engine$ consists of a serial composition of two subcompoments, namely, $Drive$ and $Halt$. An entry location is directly connected with one of the locations of $Drive$ component. There is one exit location that is accessible from both subcomponents. Other interface locations, namely $slowdown$ and $speedup$, serve for passing the control flow between $Drive$ and $Halt$ components. The component interacts with the environment by reading input variable $brake$ and controlling output variables x and dx. These variables are also available to both subcomponents.

Component $Drive$ governs train acceleration. The component is a parallel composition of three atomic components: $CheckBrake$, $Accelerate$, and $Distance$. Input variable $brake$ determines whether the train accelerates or decelerates. Its value is observed by $CheckBrake$ component that takes away control from $Drive$ component if variable $brake$ signalizes the application of train's brake. In component $Accelerate$, the actual speed of the train is computed. The train dynamics is simplified by considering that the train accelerates by $1ms^{-2}$ if its velocity is greater $20ms^{-}1$ and by $2ms^{-2}$ if its velocity is less than $20ms^{-1}$, respectively. Finally, component $Distance$ is responsible for computing the actual distance from the railway crossing.

Component $Halt$ has similar structure to component $Drive$. It's purpose is to slow the train down as long as input variable $brake$ is set to true. If $brake$ is released it passes the control back to $Drive$ component through location $speedup$.

To show that TLA$^+$ specifications conform to Masaccio interpretation, the interpretation of TLA$^+$ expressions needs to be defined. The following simplified system is used (for complete semantics see e.g. [6]). The TLA$^+$ module is called

```
1  ──────────── MODULE Drive ────────────
2  EXTENDS Integers, Sequences
3  VARIABLE brake, x, dx, clock, loc1, loc2, loc3
4  ──────────────────────────────────────
5  driveBrake ≜ INSTANCE DriveBrake WITH location ← loc1
6  accelerate ≜ INSTANCE Accelerate WITH location ← loc2
7  distance ≜ INSTANCE Distance WITH location ← loc3
8  ──────────────────────────────────────
9  Init ≜ driveBrake!Init ∧ accelereate!Init ∧ distance!Init
11 Next ≜ driveBrake!Next ∧ accelereate!Next ∧ distance!Next
12 ──────────────────────────────────────
```

Fig. 3. The TLA$^+$ specification of component $Drive$

a standard module if it has the form of conjunction of an initial state predicate and a next-state action predicate. The meaning of a standard TLA$^+$ module $M = \langle V, I, N\rangle$, where V is a finite set of variables, I is an initial predicate, and N is a set of next-state actions, is then defined by valuation function $\mathcal{V}_s(x)$, which assigns a value to each variable $x \in V$ and each state s, and model satisfying relation, $s \models_{\mathcal{M}} p$, which asserts that proposition p is true in state s in the model $\mathcal{M}$ of module M. A model $\mathcal{M}$ is a graph that consists of a set of nodes $\mathcal{M}_N$ representing states, and a set of edges $\mathcal{M}_E$ representing transitions between states. Obviously, a set of initial states of model $\mathcal{M}$ is defined as all states satisfying the initial predicate; that is, $\mathcal{I} = \{s \in \mathcal{M}_N : s \models_{\mathcal{M}} I\}$. Each next-state action n can be split into a part n_1, where only unprimed variables occur, and a part n_2, where also primed variables occur. If $s \models_{\mathcal{M}} n_1$ and $r \models_{\mathcal{M}} n_2$ then, necessary, $s, r \in \mathcal{M}_N$ and $\langle s, r\rangle \in \mathcal{M}_E$. Masaccio interpretation is defined in terms of execution traces. Obviously, an execution is a trace that can be generated by traversing a graph $\mathcal{M}$. Formally, a trace w consists of jumps (p, q), such that $p, q \in \mathcal{M}_N$ and $\langle p, q\rangle \in \mathcal{M}_E$.

A. Specifying atomic components

According to Masaccio semantics, an atomic discrete component $A(J)$ is completely specified by a jump predicate that defines a set of legal jumps J. Further, an atomic component has an arbitrary number of input and output variables. In each atomic component, there are only two interface locations, denoted as $from$ and to.

The representation of atomic component is straightforward in TLA$^+$ language. In figure 2, TLA$^+$ description of $Accelerate$ component is shown. A set of jumps is a conjunction of three next-state actions. Action $J1$ represents acceleration of a train in lower speeds. Action $J2$ represents acceleration of the train in higher speeds. Finally, action $J3$ specifies that if the train reaches its maximal speed it maintains this speed. In addition to state variable $location$ and controlled variable dx, which keeps the actual train's speed, the module declares a boolean variable $clock$, which models the passing of the time. Introducing the system clock is necessary for synchronization of the components. While this simple approach seems to be appropriate in this case, more flexible approach, e.g. [7], might be considered in more involved real-time specifications.

The following definition generalizes atomic component specification rules.

Definition 1 (atomic component): An atomic component $A(J)$ is a TLA$^+$ module $M = \langle V, I, N\rangle$ such that:

- it declares a variable for each I/O variable of the atomic component; that is, $\forall v : T \in V^{in,out}_{A(J)} : \exists v \in V$ such that $s \models_{\mathcal{M}} v \in T$ for all $s \in \mathcal{M}_N$.
- it declares a location variable; that is, $location \in V$, and $s \models_{\mathcal{M}} v \in \{from, to\}$ for all $s \in \mathcal{M}_N$.
- the meaning of next-state action N agrees with the predicate φ^{jump}_J; that is, $(p, q) \models_{\mathcal{M}} N$ if each unprimed variable x from N is assigned the value $\mathcal{V}_p(x)$ and each primed variable y from N is assigned the value $\mathcal{V}_q(y)$.
- the meaning of initial predicate I agrees with the predicate $\varphi^{en}_{A(J)}(from)$; that is $p \models_{\mathcal{M}} I$ for every trace of atomic component $A(J)$ with prefix $(from, (p, q))$ if each variable x from I is assigned the value $\mathcal{V}_p(x)$.

As it can be seen from the TLA$^+$specification in figure 2, the atomic specification contains variable denoted as $clock$. This variable serves to synchronization purposes. It enforces that parallel actions are executed at the same time. Therefore all jumps include the condition stating $clock' = \neg clock$. Except proper actions, there are also specific actions supporting serial compositions as described later in this section. These specific actions violate this condition requiring that the time is stopped; that is, $clock' = clock$.

A component can be entered at location a if an entry condition $\varphi^{en}_A(a)$ is satisfied at (p, q'); that is, $(p, q') \models \varphi^{en}_A(a)$. A valid expression of entry condition is similar to next-state relation in TLA. It has form of conjunctions of expressions that can contain unprimed and primed variables. Contrary to TLA, the entry condition is enabled if both unprimed and primed parts are satisfied in (p, q), while the TLA action is enabled if the unprimed part is satisfied in state p. This is important for guarantee of the dead-lock free property. As TLC automatically checks whether the given specification is dead-lock free, it is possible to relax the entry condition into its weaker form $p \models \varphi^{en}_A(a)$, which does not contain the primed variables.

B. Specifying Composition of Components

The component $Drive$ shown in figure 1 is a result of parallel composition of three subcomponents. The corresponding

TLA$^+$ specification is given in figure 3. The semantics of parallel composition corresponds to joint-action specification as described by Lamport in [3, p.147]. Its encoding in TLA is straightforward.

The $Drive$ module contains input and output variables $brake$, x, dx and also variables $loc1$, $loc2$ and $loc3$ that keeps the state of subcomponents $DriveBrake$, $Accelerate$ and $Distance$, respectively. These location variables are bound to variable $location$ in each component during the component instantiation as declared on lines 5-7. Line 9 defines a collection of initial states of the subcomponents. The initial predicate $Init$ is a conjunction of initial predicates of all subcomponents. Next-state action predicate $Next$ is a conjuction of next-state predicates of subcomponents, which gives the intended execution interpretation of the component; that is, the jumps of subcomponents are executed in parallel and in synchronous manner.

Definition 2 (parallel composition): A component $C = A \otimes B$ composed in parallel from subcomponents A and B can be written as TLA$^+$module $M_C = \langle V_C, I_C, N_C \rangle$, where

- V_C is a set of module's variables that includes all input and output variables of the subcomponents and location variables (an implicit renaming of location variables is considered to prevent the slash of their names in module M_C); that is $V_C = V_A \cup V_B$.
- I_C is an initial predicate that is a conjunction of initial predicates of both submodules and a component specific constraints; that is $I_C = I_A \wedge I_B \wedge I$.
- N_C is a next-state action predicate that is defined as a conjunction of next-state predicates of both submodules; that is $N_C = N_A \wedge N_B$.

A state space of a composed component is generated according the initial predicates and next-state actions of its subcomponents. The conjunction of next-state actions requires that there are simultaneous jumps in each of the subcomponent. Moreover if one of the subcomponent reaches its end location, which causes that such component has not enabled action, it is not possible to execute any jump in any of the contained components. This configuration is then recognized as the end location of the component.

The serial composition of components requires that only one contained component has control at a time. This needs to be reflected in a location configuration. Therefore a special location, denoted as empty string (""), has been added to represent a state of a component without a control. A component whose location configuration is "" cannot execute any of its jumps. To enable the passing of control between components, specific actions that modify only location variables are added into the specification. Their purpose is similar to that of connector elements that can be found in many architecture description languages, e.g. [8].

The example of a component composed in series is shown in figure 4. The module $Engine$ instantiates two subcomponents, namely $Drive$ and $Halt$. The initial predicate specifies that the component $Drive$ will have control when component $Engine$ is first executed. This means to define valid initial interface location for subcomponent $Drive$, in particular, to assign value $from$ to $dl1$, $dl2$ and $dl3$ variables, and to define that $Halt$ subcomponent is in the idle state that is expressed by assigning "" to variables $hl1$, $hl2$ and $hl3$. To define next-state action predicates we assume that specification of $Halt$ and $Drive$ were both extended with the following definition:

$$Idle \triangleq loc1 = "" \wedge loc2 = "" \wedge loc3 = ""$$

This definition asserts that component is in the idle state. Therefore, action $L1$ and action $L2$ define a behavior of the containing component as an execution of component $Drive$ and component $Halt$, respectively, assuming that a complementing component is being idle during this execution. Finally, two connector actions are necessary to allow switching between $Drive$ and $Halt$ components. In particular, connector $C1$ specifies that if $Drive$ reaches the end location $slowdown$, which is represented by interface locations $dl1 = "from" \wedge dl2 = "to" \wedge dl3 = "to"$, the control is passed to $Halt$ component entering its $slowdown$ location. This location is represented by interface location $hl1 = "from" \wedge hl2 = "to" \wedge hl3 = "to"$. The control is removed from component $Drive$ by assigning $dl1' = "" \wedge dl2' = "" \wedge dl3' = ""$.

Definition 3 (serial composition): A component C composed in series from subcomponents A and B; that is, $C = A \oplus B$, can be written as TLA$^+$module $M_C = \langle V_C, I_C, N_C \rangle$, where

- V_C is a set of module's variables that includes all input and output variables of the subcomponents and location variables.
- I_C is an initial predicate that is a disjunction of initial predicates of both submodules annotated with control flow information in the form of assertions on interface locations; that is, $I_C = (I_A \wedge L_B) \vee (I_B \wedge L_B)$, where L_A or L_B specifies that control can be assigned to component A or B, respectively.
- N_C is a next-state action predicate that is defined as a disjunction of next-state predicates of both submodules and all necessary connectors C_i; that is $N_C = N_A \vee N_B \vee C_i$.

V. Verification using TLC

In this section, a brief elaboration on results of verification experiments is presented. The TLC tool was used to check the basic properties of specifications composed in the style of Masaccio model.

Each component can be verified using TLC tool separately. Nevertheless, often a component depends on its environment and the environment specification needs to be supplied in order to get a meaningful results. For instance, component $Distance$ that computes a distance according to the actual velocity requires to provide a specification that sets boundaries on the behavior of velocity variable dx. Moreover, the dependency among the components can be circular. Therefore to verify a component, a suitable context needs to be provided. The approach used in this paper for verification of the components stems from the assume-guarantee principle that constraints the context of a component. This principle was studied in the frame of Masaccio formalism in [9]. The context does not need to be specified from scratch. Instead, existing specifications of

```
1  ──────────── MODULE Engine ────────────
2  EXTENDS Integers
3  VARIABLES brake, x, dx, clock
4  VARIABLES hl1, hl2, hl3, dl1, dl2, dl3
5  ──────────────────────────────────────
6  drive ≜ INSTANCE Drive WITH loc1 ← dl1, loc2 ← dl2, loc3 ← dl3
7  halt ≜ INSTANCE Halt WITH loc1 ← hl1, loc2 ← hl2, loc3 ← hl3
8  ──────────────────────────────────────
9  I1 ≜ ∧ dl1 = "from" ∧ dl2 = "from" ∧ dl3 = "from" ∧ drive!Init
10        ∧ hl1 = "" ∧ hl2 = "" ∧ hl3 = ""
11
12 I2 ≜ ∧ dl1 = "" ∧ dl2 = "" ∧ dl3 = ""
13        ∧ hl1 = "from" ∧ hl2 = "from" ∧ hl3 = "from" ∧ halt!Init
14
15 Init ≜ I1 ∨ I2
16
17 L1 ≜ halt!Idle ∧ drive!Next ∧ UNCHANGED ⟨hl1, hl2, hl3⟩
18
19 C1 ≜ ∧ dl1 = "to" ∧ dl2 = "from" ∧ dl3 = "from"
20        ∧ hl1 = "" ∧ hl2 = "" ∧ hl3 = ""
21        ∧ dl1′ = "" ∧ dl2′ = "" ∧ dl3′ = ""
22        ∧ hl1′ = "from" ∧ hl2′ = "from" ∧ hl3′ = "from"
23        ∧ UNCHANGED ⟨brake, x, dx, clock⟩
24
25 L2 ≜ drive!Idle ∧ halt!Next ∧ UNCHANGED ⟨dl1, dl2, dl3⟩
26
27 C2 ≜ ∧ hl1 = "to" ∧ hl2 = "from" ∧ hl3 = "from"
28        ∧ dl1 = "" ∧ dl2 = "" ∧ dl3 = ""
29        ∧ hl1′ = "" ∧ hl2′ = "" ∧ hl3′ = ""
30        ∧ dl1′ = "from" ∧ dl2′ = "from" ∧ dl3′ = "from"
31        ∧ UNCHANGED ⟨brake, x, dx, clock⟩
32
33 Next ≜ L1 ∨ C1 ∨ L2 ∨ C2
34 ──────────────────────────────────────
```

Fig. 4. The TLA$^+$ specification of component $Engine$

components that form the environment of the component being verified can be turned into a context like specification. Moreover, this context specification can be proved as appropriate if it satisfies a refinement relation. In many cases, it can be checked automatically using TLC tool. Interface refinement is described in [3, p.163] as $LSpec \triangleq \exists \hat{h} : IR \wedge HSpec$, where $\hat{h}$ is a vector of free variables of $HSpec$ and IR is a relation between variables of $\hat{h}$ and lower level variables of $\hat{l}$ of specification $LSpec$.

Verification of component $Engine$ required to compute the state space consisting of 816288 states. TLC completes this task roughly within 30 seconds on a computer with 1.66 GHz processor. It should be noted, that the specification shown in figure 4 was extended with $loop$ action for the sake of TLC verification procedure. The loop action is required to prevent TCL to complain on finding a deadlock state. It just loops forever if the end location of component $Engine$ is reached.

$$loop \triangleq (hl3 = "to" \vee dl3 = "to") \wedge \text{UNCHANGED } vars$$

The specification sent to TLC for verifying properties was as follows:

$$Spec == Init \wedge \Box[Next \vee loop]_{vars}$$

The verification of component $Near$ (see [4] for its specification) took much longer (approx. 15 minutes) and the state space searched was greater than 7 millions of distinct states. An issue lies in the use of integer variables for meassuring distance and actual speed of the train and the necessity to check whether the property holds for any combination of these values. The solution is to merge concrete values into significant intervals, i.e. for dx there are two such intervals, in particular, $hispeed = (21..50)$ and $lospeed = (0..20)$. Also distance variable x can be defined to be from a set of intervals, e.g. $extdist^+ = (\infty, 5000), fardist^+ = (5000, 1000), neardist^+ = (1000, 0)$.

VI. Conclusion

In this paper, the overview of the method capable of formal specifying and verifying embedded control systems has been presented. The method is based on the TLA$^+$, which allows to produce clear and simple specifications because of its very expressive language. An accompanying tool, TLC model checker, can be employed to show that the specification exposes intended properties. This method was illustrated on a simple example in this paper. The semantics of specification can be defined in terms of Masaccio interpretation, including serial and parallel component compositions.

In addition to clarification of the basic facts on the method for writing TLA$^+$specifications under Masaccio interpretation, the several topics for future work were revealed during the work on this paper:

- Deeper understanding of the assume-guarantee refinement in the TLA$^+$specification framework is required and

the proof that these specifications obey assume-guarantee principle as specified for Masaccio model should be given.

- Specification of hybrid systems as proposed by Masaccio was not addressed in the paper. As shown in [5], TLA$^+$is expressive enough to capture a large class of hybrid system specifications. The question is whether the verification can be adequately supported by the tools available for TLA$^+$.
- As expected and shown by the example, the state explosion is a problem in the case of verification of non-trivial systems. While TLA$^+$model-checker can explore several hundreds of millions of states, there is also possibility to apply state space reduction techniques. The TLC poses symmetry reduction mechanism [3, p.245] that can reduce significantly state space for design that contains multiple same or similar parts.

The formal model and the presented specification and verification method is suitable, in particular, for application to the domain of distributed time-triggered systems [10]. The intention is to integrate this method in a visual modeling framework [11] to enable automatic checking of properties of systems being visually modeled.

REFERENCES

[1] P. Cousot and R. Cousot, "Verification of embedded software: Problems and perspectives," *Lecture Notes in Computer Science*, vol. 2211, pp. 97–114, 2001.

[2] E. A. Lee, "Embedded software," *Advances in Computers*, vol. 56, pp. 56–97, 2002.

[3] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Professional, 2003.

[4] T. A. Henzinger, "Masaccio: A formal model for embedded components," in *TCS '00: Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics*. London, UK: Springer-Verlag, 2000, pp. 549–563.

[5] L. Lamport, "Hybrid systems in tla$^+$," in *Hybrid Systems*, ser. Lecture Notes in Computer Science, vol. 736. Springer, 1992, pp. 77–102.

[6] M. Kaminski and Y. Yariv, "A real-time semantics of temporal logic of actions," *Journal of Logic and Computation*, vol. 13, no. 6, pp. 921–937, 2001.

[7] L. Lamport, "Real-time model checking is really simple," in *CHARME*, 2005, pp. 162–175.

[8] K.-K. Lau, V. Ukis, P. Velasco, and Z. Wang, "A component model for separation of control flow from computation in component-based systems," *Electronic Notes in Theoretical Computer Science*, vol. 163, no. 1, pp. 57–69, September 2006.

[9] T. A. Henzinger, M. Minea, and V. Prabbu, *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, January 2001, vol. 2034/2001, ch. Assume-Guarantee Reasoning for Hierarchical Hybrid Systems, pp. 275–290.

[10] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, ser. The Springer International Series in Engineering and Computer Science. Springer Netherlands, 2002, vol. 395, ch. The Time-Triggered Architecture, pp. 285–297.

[11] M. Faugere, T. Bourbeau, R. de Simone, and S. Gerard, "Marte: Also an uml profile for modeling aadl applications," in *ICECCS '07: Proceedings of the 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 359–364.