energies

MDPI

Article

Formal Verification of the European Train Control System (ETCS) for Better Energy Efficiency Using a Timed and Asynchronous Model

Andrzej Kochan [1,*], Wiktor B. Daszczuk [2], Waldemar Grabski [2] and Juliusz Karolak [1]

[1] Faculty of Transport, Warsaw University of Technology, 00-662 Warsaw, Poland
[2] Institute of Computer Science, Warsaw University of Technology, 00-665 Warszawa, Poland
* Correspondence: andrzej.kochan@pw.edu.pl; Tel.: +48-22-2347882

Abstract: The ERTMS/ETCS is the newest automatic train protection system. This is a system that supports the driver in driving the train. It is currently being implemented throughout the European Union. This system's latest specifications also provide additional functions to increase the energy efficiency of train driving in the form of ATO (automatic train operation). These functions of the ETCS will be valuable, provided they operate without failure. To achieve errorless configuration of the ETCS, a methodology for automatic system verification using the IMDS (Integrated Model of Distributed Systems) formalism and the temporal tool Dedan was applied. The main contribution is asynchronous and timed verification, which appropriately models the distributed nature of the ETCS and allows the designer not only to analyze time dependencies but also to define the range of train velocities in which the operational scenario is valid. Additionally, the novelties of the presented verification methodology are the graphical design of the system components and automated verification freeing the designer from using textual design. We express the verified properties as observer automata rather than in temporal logic. Moreover, we check partial properties related to system fragments, which is crucial in distributed systems. This paper presents the verification of an example ETCS system application. The verification results are presented as sequence diagrams leading to a correct/incorrect final state.

Keywords: energy efficiency of train operation; ETCS system verification; timed integrated model of distributed systems; timed model checking; asynchronous modeling; energy efficiency

Citation: Kochan, A.; Daszczuk, W.B.; Grabski, W.; Karolak, J. Formal Verification of the European Train Control System (ETCS) for Better Energy Efficiency Using a Timed and Asynchronous Model. *Energies* **2023**, *16*, 3602. https://doi.org/10.3390/en16083602

Academic Editor: Giovanni Lutzemberger

Received: 20 February 2023
Revised: 5 April 2023
Accepted: 19 April 2023
Published: 21 April 2023

1. Introduction

The European Rail Traffic Management System (ERTMS) is a system supported by the European Union aimed at unifying the rail traffic management and control system in Europe, aiming at ensuring the interoperability of rail transport [1]. The ERTMS system consists of the ERTMS/ETCS European Train Control System (ETCS) [2] and the ERTMS/GSM-R, a digital railway radio communication system based on the GSM standard (GSM-R) [2]. The ERTMS system is being implemented in all European Union countries and outside Europe (e.g., in Middle East countries). In Poland, it is to be installed on railway lines with a total length of over 7000 km. Equally important, apart from the interoperability and safety of rail traffic, is the quality of transport services and their energy efficiency. The latest specifications of the ETCS system, which are currently being consulted on with the railway industry, extend its functionality towards the functionality of ATO (automatic train operation). The key function of the ATO system is to control train velocity, which should be done in a way that minimizes energy consumption and ensures passenger comfort [3]. Unfortunately, the automatic train driving algorithm requires faultless configuration of the ETCS application on the line. In the event of errors, ETCS can cause undesirable braking of the train, which significantly reduces the energy efficiency of the train run. This problem

is described in more detail in the next section, and further sections describe a verification method to minimize energy losses.

Some ETCS verification attempts are described in the Related Works section. The thesis of this article and its contribution is that this verification should be performed in an asynchronous timed model to produce relevant results. Most of the approaches mentioned in the Related Work section deal with timeless checking, and time-based ones use synchronous models. Asynchrony is needed in modeling distributed systems so that actions in one of the components of the system can affect the other components only by sending signals to them, not by directly enabling/disabling their operation. This supports the locality of actions in a distributed system. Timing relationships can significantly change system behavior compared to sequence-based analysis, so real-time analysis must be used to obtain reliable results. For example, it supports the range of train velocities where desired properties hold.

The article is organized as follows: Section 2 describes the goal of verification from the point of view of the ETCS designer. Section 3 covers the ETCS system's characteristics. Section 4 covers related work. The criteria for choosing the verification environment are reported in Section 5. Section 6 describes the elements of this environment. Section 7 covers the example of verification of the behavior of the train driver in operation during the confirmation of balise telegrams. Section 8 concludes the article.

2. Energy Efficiency as Verification Goal

The authors of [4] introduced a new parameter describing the quality of traffic, bearing in mind that quality is treated as a collective property, indeterminate and challenging to measure, but capable of being described and quantified as the resultant intensity of the most important factors affecting it. The parameter proposed by the authors is the smooth running of the train, which aligns with the idea of a balanced value. This parameter can be used to analyze the energy efficiency mentioned above.

As mentioned earlier, a malfunctioning ETCS can cause trains to brake unnecessarily. The scale of costs incurred by operators due to disturbances occurring on the railway network, including the need for frequent train braking and acceleration at speed limit locations, is presented in the articles [5,6]. The authors present calculations for an unplanned stop and start of a 490 t passenger train led by a EU07 locomotive. This traffic scenario increased energy consumption by 75.6 kWh, while for a goods train of 1900 t led by an ET22 locomotive the increase in energy consumption is 191.4 kWh. In a study by the same authors, based on the results of simulation runs on the E59 line, a fast train between Poznań Gł. and Wrocław Gł. showed an approximately 40% increase in traction energy due to disturbances in the movement of the tested train.

In contrast, report [7] pointed out that traffic fluidity is a significant problem for the energy efficiency of train running, as each additional stop (and subsequent acceleration) of a train requires additional traction energy. Measurements carried out on IC-2000 trains running between Lucerne and Zurich showed that crossings in which unexpected stops at signals occurred resulted in energy consumption 10–15% higher than non-stop crossings.

The impact of unjustified braking of the train on the increase in energy consumption must also take into account the energy loss resulting from the increase in the running time of the train (the difference in running time from the expected scheduled train running time). Suppose we accept the criterion of the punctuality of a train, in terms of the carrier's costs for delaying the train. In that case, it must be taken into account that, in the process of controlling the train, the driver will seek to maximize the traction power to shorten the journey. However, this method of driving the train will increase energy consumption due to the use of maximum acceleration and running at maximum resistance (maximum permissible speed).

The increase in running time is, therefore, an essential element to be taken into account in the traction energy consumption of the train.

In terms of maintaining the train's scheduled running time, running the train at the maximum running speed on the line is most advantageous. However, when approaching speed restriction locations, a decision has to be made regarding when to apply the brakes and the braking force used. The braking force, on the other hand, is proportional to the required rate of change in the train's kinetic energy and, therefore, to the consequences of its loss on train movement parameters such as travel speed, distance, and time, ultimately translating into energy consumption.

For example, slowing down an ED250-series ETR610 train weighing 427 tonnes from 200 km/h to 180 km/h results in a distance loss of approximately 636 m and an increase in journey time of 12 s for movement at a constant speed of 200 km/h. Regaining speed requires an estimated mechanical energy consumption of 152 kWh (assuming constant resistance to motion and power supply), while in the case of slowing down to 170 km/h the loss of distance is already 1687 m and the extension of the driving time is 31 s, which requires a mechanical energy consumption of 240 kWh.

In addition, response time and delay in the change from braking to the traction system must also be taken into account. Based on EN 14531-1:2015, the estimated response time of the braking system ('removal' of the braking force), depending on the braking force applied, is approximately 3–6 s. In comparison, the time for control actions by the driver is assumed to be within 5 s [8]. Therefore, it can be assumed that the restoration of traction power will take place after about 10 s, causing the train's speed to decrease by a further 3 km/h as a result of the resistance to motion, with a consequent increase in the running time of 3 s and a final increase in mechanical power consumption of about 30 kWh.

Those problems require implementing new technologies for designing and verifying ETCS applications. Developing solutions that will comprehensively prove the system's correctness for defined criteria is necessary.

The contribution described in this article consists of:

- Timed verification concerning real-time dependencies in a checked system. These dependencies can significantly modify the operation of a system compared to event-sequence-only checking. However, this is seldom used in the literature.
- The application of asynchronous modeling. To our best knowledge, no cited article describes asynchronous modeling of the ETCS control system. The synchronous model is inadequate for distributed systems.
- Modeling of the verified system based on the graphical specification in distributed automata, in which the semantics is identical to the formal model IMDS used in the proposed methodology. The graphical design is close to the intuitive mode of operation of designers who do not know temporal logics.
- The cited articles use the global state for both modeling and checking properties. We argue that such a state does not exist, and that only local features should be applied. Our observer concept meets this requirement.
- Automated verification, without specification of properties as temporal formulas. The features automatically checked are deadlocks and termination. Using observers, these two features are enough to express a wide range of behaviors.
- Checking partial features, in which only a subset of system processes participate: for example, it is usual that in a distributed system, some processes fall into a deadlock while others work correctly.
- Verification concerns not only meeting correctness conditions, but, additionally, we check the scope of train velocities within which those conditions are valid.

3. ETCS Characteristics

The key functions of the ETCS system have already been outlined in the introduction. For further consideration, it is necessary to give at least a basic overview of the system's design and working principles [2].

The ETCS system consists of a trackside part and an on-board part. The on-board part has a fixed structure for all application levels (with the precision of the EDOR (ETCS

data-only radio) broadband modules). The trackside part structure strongly depends on the track layout, the parameters of the railway route, organizational and operational rules, and the technical solutions used in the control system base layer. These structures are formed by the different layouts and configurations of the interoperability constituents defined in the Technical Specifications for Interoperability.

A railway traction vehicle with ETCS installed (ETCS vehicle) is equipped with the following on-board components:

- European vital computer, EVC,
- Maintenance computer, MC,
- Driver–machine interface, DMI,
- Juridical recording unit,
- Odometry,
- Balise transmission module, BTM,
- Antennae for the reception of eurobalise telegrams,
- Communication module and antenna for GSM-R communication,
- Specific transmission module, STM.

These devices form the equipment of the so-called ETCS Eurocab. A detailed analysis of all the devices on the list is irrelevant to the present discussion. Selected aspects of individual devices will be noted at appropriate points.

The structure of the trackside part may consist of elements such as:

- Eurobalise (henceforth: balise),
- Euroloop,
- Radio infill unit, RIU,
- Line-side electronic unit, LEU,
- Radio block centre, RBC.

A balise is a trackside transponder responsible for the transmission to/from an ETCS train passing over it (i.e., at a specific point) of telegrams containing ETCS language packets. Balises provide point-to-point interaction with a train passing over them. The system specification distinguishes between switchable and non-switchable balises. Typically, balises are installed as a group to increase the number of telegrams sent to the train from a single location. The term balise will be used instead of balise group for simplicity in the following text.

The Euroloop and the radio infill unit (RIU) provide a continuous update of balise data on a particular section of the track.

The LEU encoder is an intermediary circuit that converts base-layer signals into telegrams, which are then sent to the switching balises via the C interface, adapting the information sent to the train to the existing traffic situation.

The RBC, or radio control center, is a characteristic component of application levels 2 and 3 of the ETCS system. It enables area-based train safety supervision in close cooperation with the GSM-R subsystem for area-based track-to-vehicle data transmission.

The interaction between the track-side and onboard can take place according to different principles. These rules are determined by the ETCS system application level and the mode of operation of the on-board equipment at a given track-side location.

The ETCS system can operate in one of five application levels, of which two main ones are currently used in practice:

- ETCS level 1 (ETCS L1),
- ETCS level 2 (ETCS L2),

ETCS L1 is a variant where transmission of the movement authority (MA) from the track-side to the train is point-to-point using balises. Similarly, all other information is also transmitted in the same way. There is no RBC in this ETCS application configuration.

ETCS L2 uses continuous two-way data transmission between the train and the RBC implemented via GSM-R, which is used to transmit the movement authority (MA) and

report the position and speed of the train. Balises serve as location points and send additional information, including text telegrams.

Only the principles of train–balise interaction, the data flow in the on-board equipment, and the driver's interaction with the train will be relevant for further consideration. A simplified diagram of ETCS train–balise principles is shown in Figure 1.

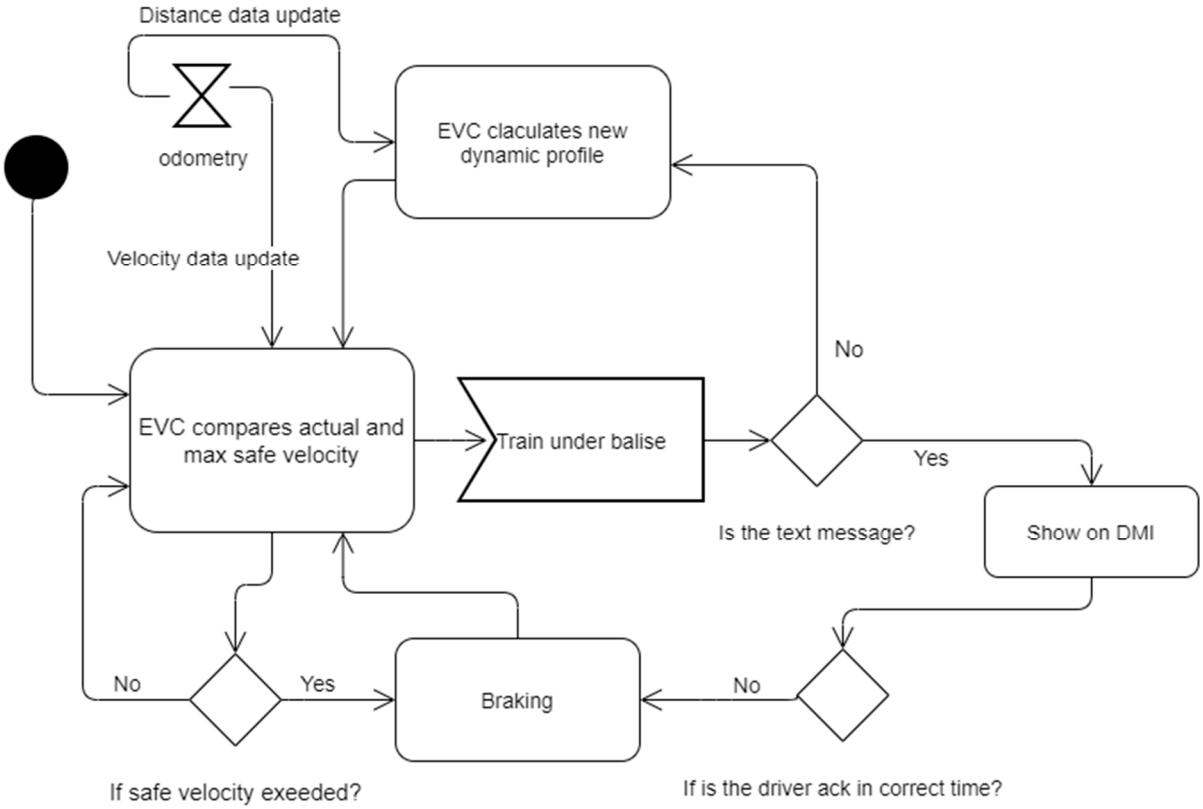

Figure 1. Simplified diagram of ETCS system operation.

4. Related Work

The use of various formalisms to verify the ETCS is a topic covered in several articles. For instance, the B-Method and the Atelier-B toolbox are utilized in [9,10]. The use of CSP-B and Pro-B is explained in [11]. Conversion of the UML specification to B is used in [12]. Two compositional methods utilizing RT-Tester and NuSMV are suggested in [13]. NuSMV is used for invariant checking in [14] in the railway control tables. The process of using the SMT solver to determine whether safety rules defined as invariants have been violated is detailed in [15]. Study [16] discusses invariant verification using a Key-ABS verifier and ABS (Abstract Behavioral Specification). Many verifiers may be used to validate Petri nets, including [17,18]. The S3 tool, which offers bounded model testing and theorem proving, is used by the authors of [19]. A special language for railway control specifications, TLA+, and the verifier TLC are described in [20]. The model-checking technique is used to exhaustively analyze the behavior of the system in terms of changes in the operating mode of the ETCS system [21]. Formal methods were used to prove the lack of collisions of trains running supervised by ETCS level 3 in a limited network area [22].

Using NuSMV and OCRA (a tool for examining the refinement of temporal contracts), the operation of the rail traffic control system under incorrect signal sequences is explored [23]. An alternative verification technique to model checking—theorem proving using PERF—is used in [24]. The deductive verification tool KeYmaera, for parametric analysis of train control in ETCS, is covered in [25].

The abovementioned works concern timeless verification. However, real time plays a significant role in ETCS behavior. Thus, timed verification is the subject of some articles.

Paper [26] presents failure analysis based on system log analysis with Uppaal [27], a verifier with real-time constraints based on timed automata. The designer must typically express the attributes as temporal formulae in order to use the higher-level verification methods that have been presented. Uppaal-SMC is used for statistical ETCS verification of automotive systems [28]. In [29], real-time verification of component-based systems using TRD-Finder for overestimated models and counterexample analysis is described.

The mentioned verification methods lack the features that are present in our method:

- Specification in languages specific to the verifier rather than a graphical form convenient for the designer;
- The concept of observer automata is seldom used; we found such an approach only in [28]; observers are used at a lower level of abstraction for verification of relay-based control equipment [30,31]; temporal formulas are still required but are applied to the observers and not to the system itself;
- Automatic checking only of total deadlocks; other properties must be specified as temporal formulas (except [28,30,31]),
- The models used are generally of a synchronous nature, like CSP-B [11] or timed automata [27], which breaks the locality of actions in a distributed environment. For example, the execution of an action in a device can disable an action in another device that was previously enabled.
- The mentioned works prove numerous desired properties, primarily of an event-sequence-based nature. The timed ones additionally check some real-time-related features. However, none of them analyzes the conditional properties that hold only in specific time circumstances, like the range of train velocities in which they hold, which is subject of our research.

5. Tool Selection Criteria

Model checking allows for evidence of the correctness of system behavior over time. In testing, we cannot execute all possible system runs, especially when concurrent processes are involved. Unlike in testing, model checking does not follow specific system executions, but analyzes them "all together", giving formal evidence of whether or not specific properties are met.

It is crucial to choose such a verification methodology to reflect the features of the system being verified as well as possible, and, on the other hand, to facilitate multiple verifications of many versions of the project under preparation.

- A railway line is a distributed system, consisting of many autonomously operating components: trains, track equipment, control centers, and, finally, people serving the line. Many modeling formalisms come from concurrent systems transferred into a distributed environment, ignoring some of its characteristics. For example, modeling based on knowing the global state of the system is entirely unrealistic in a distributed environment. It is best when the system can be modeled as a set of independent components that cooperate using mechanisms like messages.
- Line elements communicate asynchronously, with "asynchrony" being understood as the fact that nothing happens synchronically between components, that is, at the same time (unless by chance). Synchronous communication between system components, assuming that both components must reach given states in a coordinated way to exchange a message, cannot be achieved. Communication with a component cannot occur on the condition that it is in a particular state, because how would this state be known? Therefore, mechanisms such as the simultaneous execution of actions by two components, like in Büchi automata [32] or timed automata [33], cannot come into play. The components cannot execute the actions synchronously; instead, they send asynchronous messages that wait for acceptance at other components' inputs.
- Numerous verification tools automatically locate deadlocks in checked systems, but they are global deadlocks in which all processes participate, for example in [34]. In distributed systems, it is not uncommon for some components to deadlock, while

others still perform their functions. Some verifiers can find local deadlocks on the condition that the checked system has a strictly limited structure [35,36]. Otherwise, the designer can specify temporal formulas for partial deadlocks or other local properties. Therefore, automatic detection of partial deadlocks, rare among verifiers, is desirable.

- The inference of global state correctness may be an issue during the verification of global behavior, and the verifier knows the global state that is inaccessible to system components. The components cannot take any actions based on the global state, like in solitaire or 8-puzzle benchmarks, used in numerous articles [37]. However, if the inference is not made from the global state, it can also be made dynamically at runtime, not only statically. For this purpose, "observers" are helpful, as they record actions or changes in the state of individual components and, based on them, make decisions about the correctness of the entire behavior [38]. The possibility of adding such observers to a model of the verified system, and to the system itself at the time of execution, is an important issue.
- Multiple verifications of many versions of the same line require verification process automation. The questioning itself should also be facilitated: practice shows that designers are reluctant to learn the temporal logic in which properties are expressed [39]. Here, it is possible to help observers that check the achievement of specific goals by the components. Deadlocks should be found in a "push-the-button" style, while the designer supplies only the model and the observers.
- The assumption that communication and internal components' actions take zero time does not stand up to scrutiny. Some properties depend on relations between the time durations of the actions undertaken in the system. Time dependencies change the behavior of the system. Technically, some new deadlocks can arise when events do not occur in the expected time. On the other hand, proper time relations can prevent the system from falling into a deadlock. Therefore, timed verification is needed to judge the correctness of the system. Two general approaches are continuous time [33] and discrete time verification [40]. The latter can be applied to systems based on a clock or other timed lock-step mechanisms, such as centralized computers operated by a central clock. For distributed systems, real-time verification better reflects the asynchrony of components' operation.

6. Timed IMDS

6.1. IMDS-Overview

The Integrated Model of Distributed Systems (IMDS [41,42]) is explicitly tailored to the verification of such systems. Formalism is based on the collaboration of autonomous communicating servers. Communication is asynchronous. That is, messages are sent to the destination server without knowing its history and current state. The server also does not have access to the global system state (which does not actually exist in a distributed system). The system defines sequences of actions on servers threaded by a sequence of messages, which we call distributed agents. Agents are as independent of each other as servers; they communicate with each other only through the states of servers that are set as a result of performing a server action.

Inference about the system's characteristics is carried out based on "external" knowledge of the global state, i.e., the states of all servers and the messages on the way. Inference can also be carried out by introducing additional agents called observers [43]. An observer performs its calculations based on polling nodes about their state changes, and a specific behavior of the observer is tantamount to fulfilling a particular feature. Specifically, the inevitability of the system reaching a particular state can be modeled as agent termination after verification that the required actions in the system have been performed. On the other hand, the possibility of the system falling into some undesirable state should be modeled as an "artificial" observer deadlock. In this case, we are asking about the possibility of a deadlock as opposed to the inevitability of a positive result. Of course, deadlocks in components other than observers can also arise, which should be analyzed after verification.

Because a deadlock can occur locally, in which only distinguished components participate, identification of partial deadlocks is crucial in verification. Especially in a distributed system, it is not unusual that some of its components fall into a deadlock while others, perhaps not cooperating at this moment, remain operable.

The system features being verified are written as temporal formulas over the system state space, which usually deters designers from formal verification of their systems. Therefore, universal formulas are defined in IMDS, which the designer does not need to know. The price for this is a limited set of features that can be researched. As we will see, this is not a hard limitation with the use of observers. The features checked in IMDS automatically are the partial (or total) deadlock of a set of servers, partial (or total) deadlock of a set of agents, partial (or total) inevitable termination of a set of agents, and partial (or total) possible termination of a set of agents. In numerous verifications, we showed that such a set of features, together with the introduction of observers, allows for the successful verification of quite complicated systems, like the Karlsruhe production cell [44] or a fragment of the Rome metro [45].

Informally, a server is in a deadlock if messages are pending in it but no action can be taken on that server now or in the future. An agent is deadlocked if its message would never be handled (regardless of whether the message is waiting on the deadlocked server or the working server). The termination of an agent consists in the execution of a special, terminating action by it. Inevitability of termination is that the possible agent run may branch, but the branches only lead to paths terminating in a non-deadlock or to loops from which there is an exit. Termination possibility means at least one agent path leading to termination. Those features (in total or partial form) are examined by temporal formulas hidden in our verification tool Dedan, so the designer can only select the verified feature and run the verification. The verifier confirms the desired behavior by showing the witness or the violation by means of a counterexample. Additionally, the simulation of a witness/counterexample on graphical component behavior is possible [46].

6.2. IMDS-Formal Description

Formally, IMDS is a relation between server states P and agent messages M, $\Lambda \subset (M \times P) \times (M \times P)$. This is a set of system actions, $\lambda = ((m,p), (m',p'))$, $\lambda \in \Lambda$. The pair (m,p) is the input of the action, which means that the message m is pending on the server with the state p, which invokes the joint action of the server and the agent containing this message. As a result, the server gets the new state p' and issues the next message of the agent m'. The system starts from the initial states of all servers and the starting messages of all agents, which is the initial configuration T_0 of the system. The configuration $T \in 2^{M \cup P}$ is the set of all server states (one state for every server) and messages of all non-terminated agents (one message for every non-terminated agent). Every action $\lambda = ((m,p), (m',p'))$ transforms a configuration T_{inp} containing m and p to a next configuration T_{out} in which m is replaced by m' and p is replaced by p'. There are also agent-terminating actions that do not produce output messages. They are defined in another domain: $\Lambda_{term} \subset (M \times P) \times (P)$. An agent-terminating action replaces p with p' and simply consumes an agent: its message is not present in the output configuration. In such a way, a labeled transition system (LTS [47]) is constructed with vertices being the configurations, transitions being the actions, and the root being T_0.

For modeling purposes, we denote states as pairs (s,v), where s is a server and v is a value. Likewise, we denote messages as triples (a,s,r), where a is an agent, s is a server, and r is a service, distinguishing different services that can be called by the agent on the server; for example, services can be on and off of a device. It is natural that in a pair (m,p) firing an action, $m = (a,s_1,r)$, $p = (s_2,v)$, $s_1 = s_2$. For the input message and output message of an action, $m = (a_1,s_1,r_1)$, $m' = (a_2,s_2,r_2)$, $a_1 = a_2$. For an input state and an output state, $p = (s_1,v_2)$, $p' = (s_2,v_2)$, $s_1 = s_2$.

The input language of the Dedan verifier is an IMDS formalism, supported by server and agent types (with formal parameters) and variables (with actual parameters). This is beyond the scope of the article and can be found in [48].

6.3. Timed IMDS

Untimed verification can only concern the sequences of configurations. However, the behavior of a really asynchronous system can depend on real-time dependencies between events occurring in a system. Timed automata is one of the most used formalisms (TA [33]). TA are similar to Büchi automata, as distinct automata make transitions on a common symbol synchronously, as in Figure 2. Additionally, a set of real-time clocks is defined, all starting from 0. There are two types of transitions in a system: progress transitions are made spontaneously by individual automata on their own symbols, or synchronously by pairs of automata on shared symbols, as in Figure 2. The timed transitions consist in advancing all clocks by equal real values. The timed transition must not exceed time invariants defined for locations (counterparts of automata states) in which clocks can be compared with constants. A subset of clocks can be reset to 0 on a progress transition. Timed transitions and clock resets are the basis of constructing time regions having constant values for the integer parts of all clocks and given relations between the fractional parts of all clocks. The graph is bounded by the maximum integer values to which the clocks are compared. The example of a graph of two clocks x and y, with the maximum values $x = 3$ and $y = 2$, is presented in Figure 3a. Figure 3b shows the types of clock regions: in vertices, integer values for all clocks; in horizontal and vertical segments, one of the clocks has an integer value; in diagonals, the clocks are not integers, with an equal fractional part. Inside triangles, the clocks are not integers, with a constant inequality relation between the fractional parts of the clocks. Time progress is always diagonal on timed transitions. Some clocks can be reset on progress transitions, while the rest of the clocks preserve their values. Figure 3c shows all the possible regions between two integer values of every clock. Figure 3d presents the regions near the upper restrictions.

Timed automata are very efficient in verification, as shown in many articles, including [31]. These articles mostly use the Uppaal verifier [27], in which additional variables are incorporated, which can be checked and assigned on progress transitions. However, in our opinion, modeling distributed systems using automata that follow synchronous transitions is unrealistic. Therefore, we developed a real-time version of IMDS (T-IMDS [46]). First, we recalled that IMDS is really asynchronous, which we illustrate in Figure 4 with automata that send messages to their input structures represented by putting messages into individual input message sets of the automata. An automaton takes a message from its input set when it is ready for it and in a locally defined order (determined by the actions that are currently enabled in the server). Therefore, the automata (a graphical counterpart of IMDS servers) are in some aspects similar to pushdown automata, PDA [49], and message passing automata, MPA [50], which store messages on their inputs, organized in LIFO stacks or FIFO queues, respectively. Our automata do not impose any order on incoming messages.

In T-IMDS, two time-dependent elements are added: time of message delivery (channel delay) and action duration. Both delay and duration can be specified as time ranges with open or closed integer bounds. Instead of square brackets, we use triangle brackets to distinguish them from indexing: <2,3>, (2,3), <2,3). The duration can also be deterministic: <2,2> for short <2>. The instant action has a duration of <0> (the same can apply to delays). The duration of an action is given between its input and output: ((*m*,*p*),<2,3) (*m*′,*p*′)). The state p' in the action is available as the input state for the next action after a time drawn between 2 and 3 expires. The action m' is ready to invoke an action in the target server after the same time plus the channel delay drawn in the same manner. Timed IMDS is precisely defined in [46], together with its translation to timed automata. The translation converts every server to a timed automaton with its own clock and appropriate time invariants. The rule of translation is given in Figure 5, where the maximum action duration

is applied as a time-invariant limit and the bounds of the duration restrict the clock values at which the progress transition can be executed. Every inter-server channel is modeled as an additional timed automaton that gets a message from the sending automaton, holds it for a drawn delay time, and then deposits the message with the target automaton. The precise translation rules are given in [46]. In this study, we only use the action durations; the message delivery is instant and does not require additional automata.

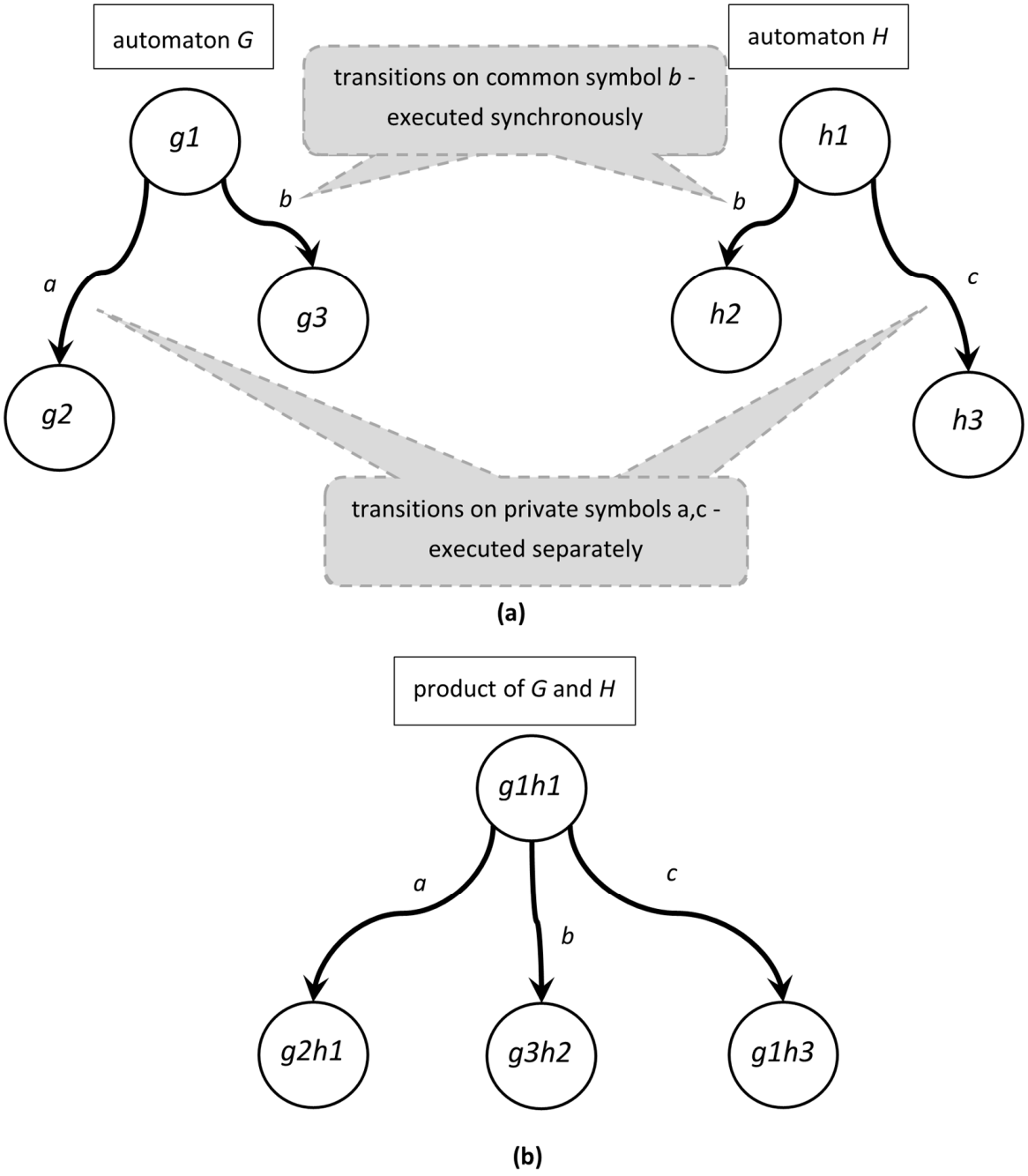

Figure 2. Automata in synchronous models: (**a**) two automata G and H with independent internal actions a and c and one common synchronous action, b; (**b**) the product of automata G and H.

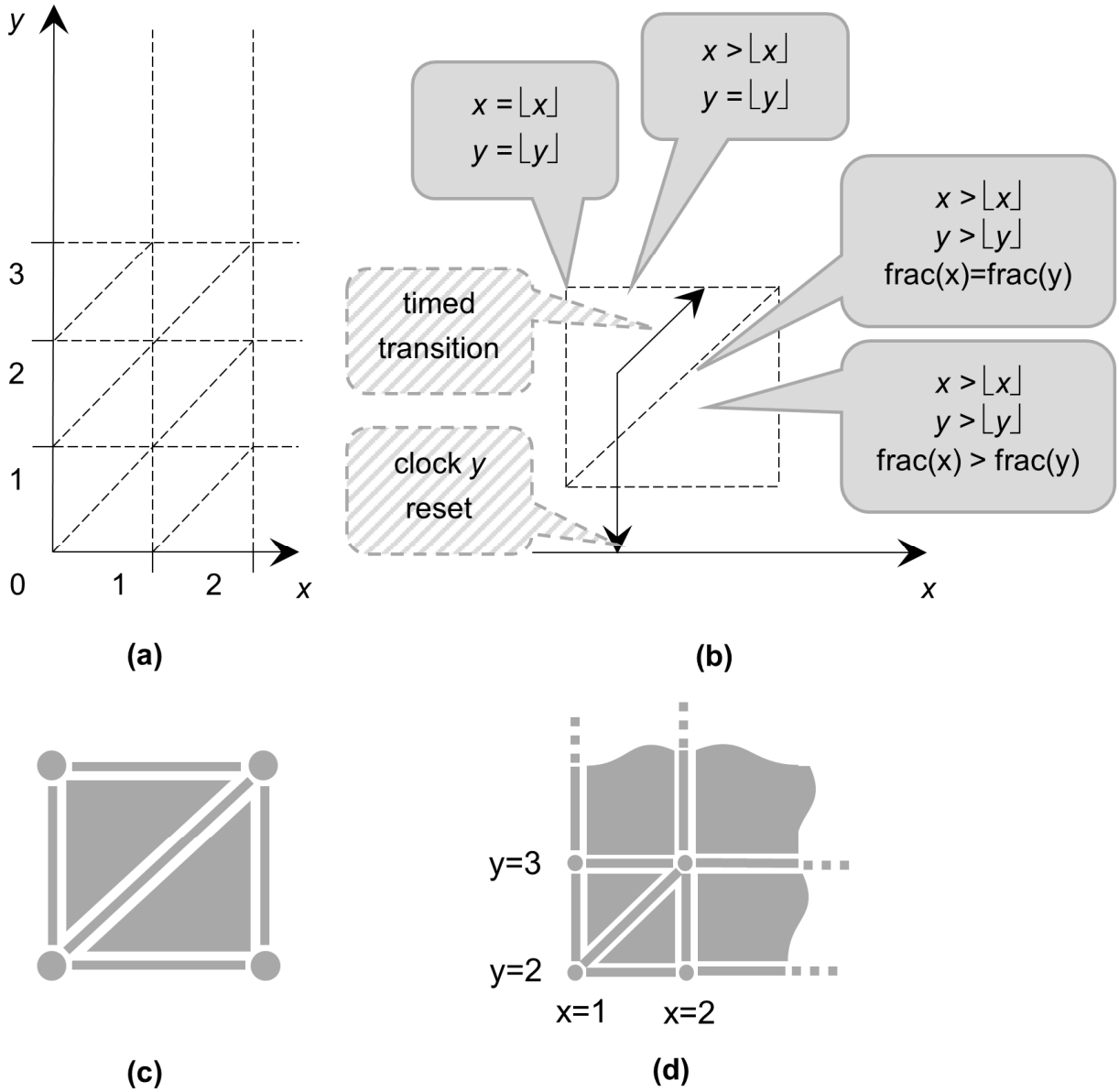

Figure 3. Figure 3. (**a**) Example region net for two clocks x and y and maximum integers of 2 for x and 3 for y; (**b**) example regions where $\lfloor x \rfloor$ stands for an integer part of x and frac $(x) = x - \lfloor x \rfloor$ is a fractional part; (**c**) all regions between two integers on both axes; (**d**) regions near the upper restrictions of both clocks.

Let us describe how timed regions work. Consider the region R1: $\lfloor x \rfloor = 1$, $\lfloor y \rfloor = 2$, $x > 1$, $y > 2$, $x > y$ in Figure 3a. For any point inside this region, a timed transition to the same region can be executed, or a transition to the next region R2: $\lfloor x \rfloor = x = 2$, $y > 2$, $y < 3$. Additionally, a progress transition can occur, resetting clock x (R3: $2 > y > 3$, $x = 0$), clock y (R4: $1 > x > 2$, $y = 0$), or both (R5: $x = 0$, $y = 0$). Therefore, the regions are abstraction classes of clock values, and we can build transition abstraction classes of transitions. The abstract transitions represent all individual transitions leading from one region to the next region. In the example, the successors of region R1 are all R1–R5. As the clock resets can occur anytime, the difference between clock values can be a real number. The equivalence of semantics between T-IMDS and its translation to timed automata, shown schematically in Figure 5, is proved in [46]. Model asynchrony is achieved by inserting additional TA separating the servers.

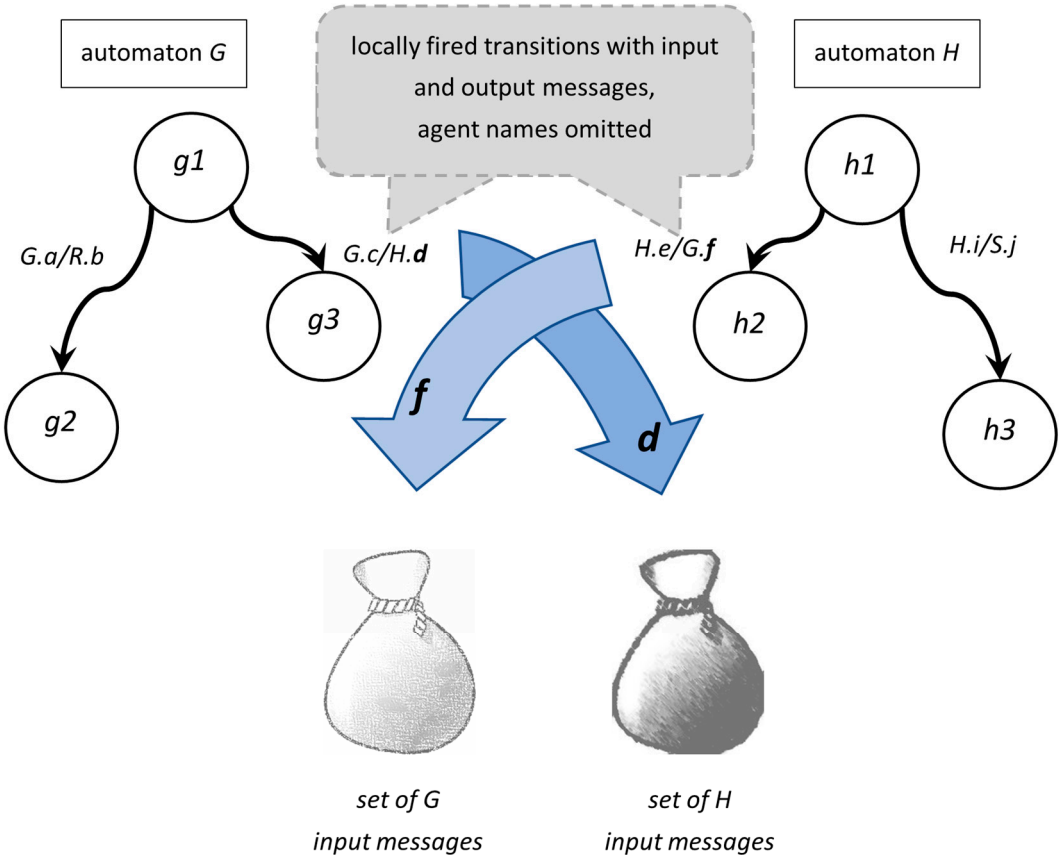

Figure 4. Asynchronous automata model.

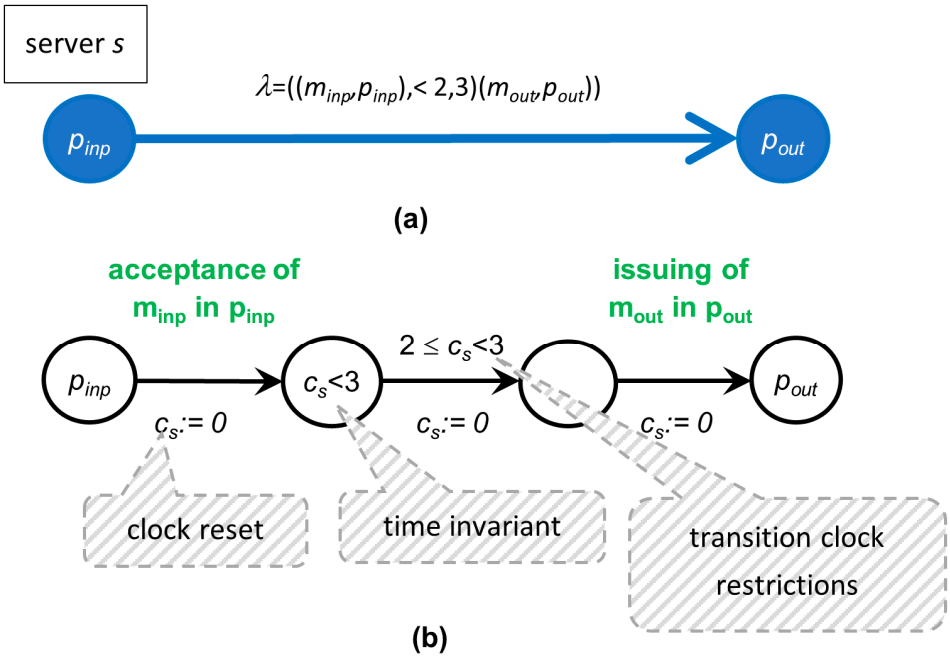

Figure 5. Rule of translation of IMDS-timed action to a sequence of transitions in a timed automaton: (**a**) IMDS timed action with duration <2,3); (**b**) timed automaton s transitions, implementing action duration.

6.4. Graphical Notation

The T-IMDS model is presented in the DA^3 formalism (Distributed Autonomous and Asynchronous Automata [51]), which is the graphic counterpart of IMDS, with identical semantics. We used the transitions of those automata above informally. The basic element of modeling is an automaton that represents a server. It follows the transitions that are equivalent to IMDS actions. The action is a transition in the server graph, starting from its state, triggered by an agent's message. The transition leads to a new server state and sends a new message, usually to a different server but not necessarily. This is illustrated by the example in Figure 6a. It is a fragment of the *Node* server automaton. The action leads from the *s1* state to the next *s2* state. The states are pairs (*server, value*): in our case, *Node.s1* and *Node.s2*. The action (*A.Node.service, Node.s1/ A.Othernode.some_service, Node.s2*) is triggered by agent *A's* message of the form of a triple (*agent, server, service*). The server is the addressee of the agent's message. The third message element—service—distinguishes between different messages an agent can direct to the same server. In the example, the message *A.Node.service* triggers the action shown, and, within the same agent, the next message *A.Othernode.some_service* is sent to the *Othernode* server automaton, calling its service some_service. Figure 6b shows a simplified notation of an action (*A.~~Node~~.service, ~~Node.s1~~/A.Othernode.some_service, ~~Node.s2~~*), where input and output states are suppressed as they are the origin and the target of the transition. The agent name is also omitted in the output message as it is the same agent as in the input message. The server name is suppressed in the input message because it is the name of the automaton: (*A.service/Othernode.some_service*). Special actions are used to terminate agents if they have already done their job. Such an action has no output message, as shown in Figure 6c: (*A.service*/-).

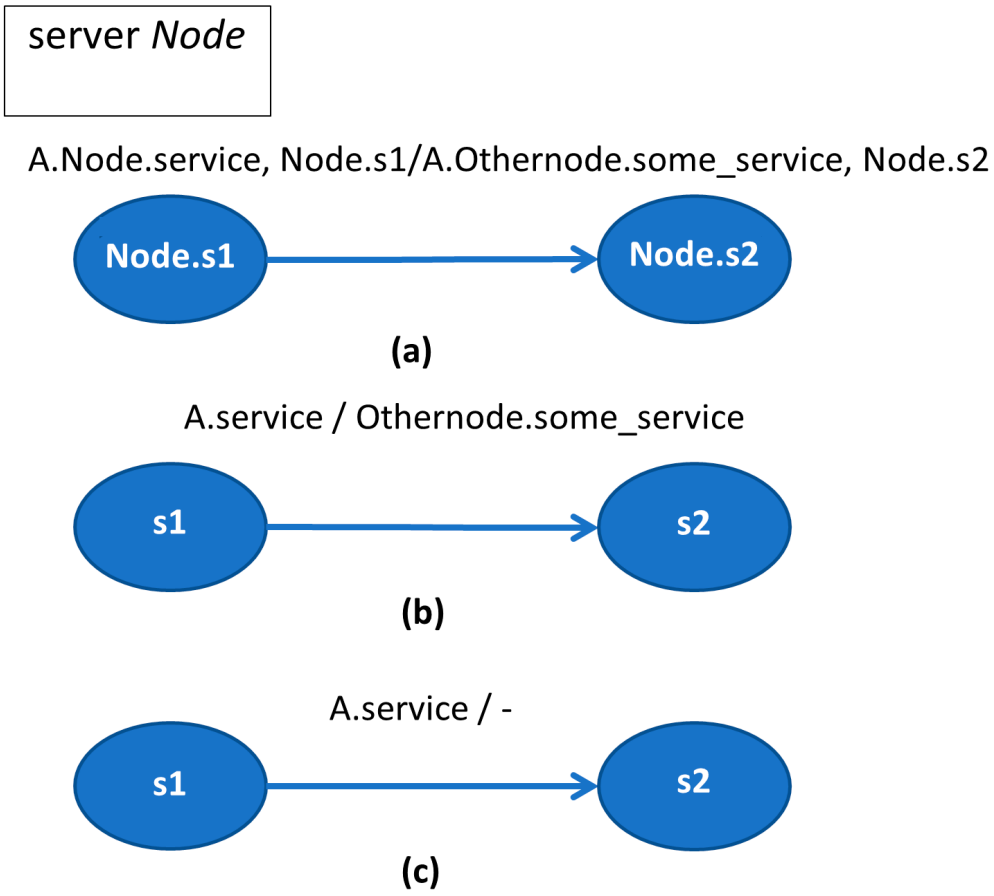

Figure 6. Rule of translation of IMDS-timed action to a sequence of transitions in a timed automaton: (**a**) IMDS-timed action with duration <2,3); (**b**) timed automaton s transitions, implementing action duration, (**c**) an agent-terminating action.

7. Verification of a Model

7.1. Operational Scenario

As an example of verification, we present an operational scenario where the infrastructure model consists of the straight track of a railway line. There are three balises on the line, separated in pairs by a distance of 650 m. The traffic situation that will be exemplified further is as follows (Figure 7). A train is standing at station P2 (right) by semaphore B. After the departure time has passed, it gets permission to proceed and starts on the track. Continuing on the track, it passes three balises (BG07_F01, BG06_F01, BG05_F01) which send a telegram with packet 72 to the driver. These are text telegrams that require confirmation. On reaching station P1, the train stops in front of semaphore J. Driving under ETCS supervision, the train can theoretically reach speeds of up to 500 km/h (in practice, these speeds are little more than 300 km/h).

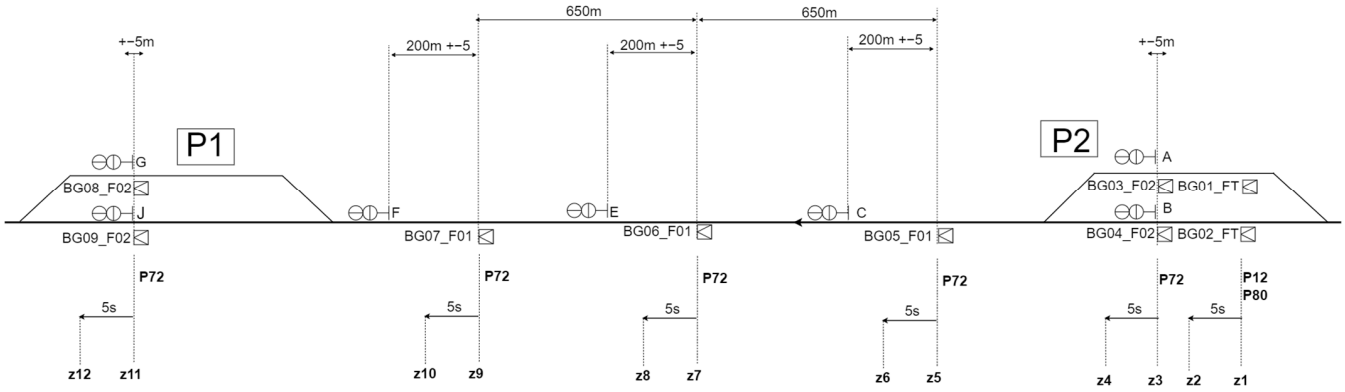

Figure 7. Configuration of the track layout and the functional infrastructure necessary for the implementation of the exemplary operational scenario.

The balise, powered by the moving train, sends it a telegram after one second. Each balise sends the telegram to the train with the text for the driver. The telegram is received by the train's on-board equipment and displayed to the driver. As every telegram must be acknowledged, the equipment is modeled as a buffer in which unacknowledged telegrams are waiting. Every telegram received by the equipment increments the content of the buffer. The confirmation by the driver decrements the content of the buffer. The driver is obliged to confirm the receipt of the telegram within 10–15 s by pressing a key, and the times of the mental and physical reaction of the driver are known. The driver is displayed one telegram even if there are more of them for confirmation. The confirmation of all three telegrams is considered a success, while the "loss" of any of them results in an emergency braking of the train. Therefore, to achieve smooth movement, the appropriate environment should be defined (balise placement) and ride events should be adequately scheduled (moments in time when formally independent telegrams are issued to the driver). Velocity should be planned to allow the driver to confirm the telegrams in time in every circumstance. The verification can define this velocity correctly.

7.2. The Model

The basic model node is the train, shown in Figure 8 (*OBU*; onboard unit). The operation of the train and driver passing a single balise is as follows:

- The train approaches the balise,
- The train powers the balise remotely,
- The powered balise sends a telegram to the driver using *OBU*,
- *OBU* displays the telegram to the driver,
- the driver confirms the telegram.

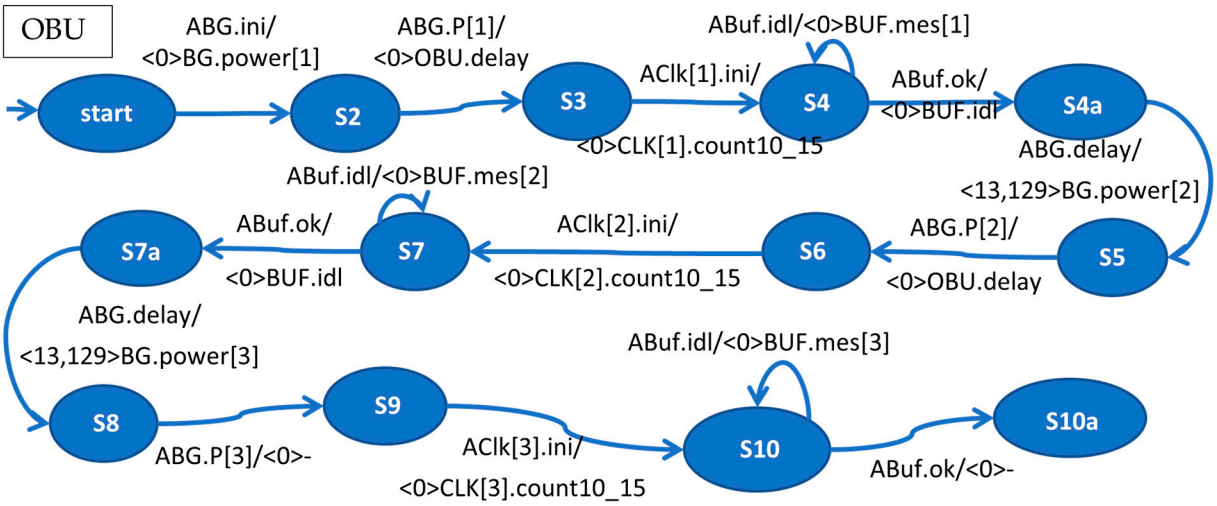

Figure 8. The automaton modeling the train onboard unit (OBU).

The balises are energized strictly in series because their reaction time is very short, and a balise must have time to send a telegram before the next one is energized. The counters must use separate agents because multiple telegram confirmations can overlap in time so that the response times can be counted down simultaneously. Therefore, telegrams are put into the buffer one after another, regardless of whether the previous one has been confirmed. Therefore, the times needed to confirm individual telegrams can overlap, and, if there are too many telegrams in a given period, some telegrams can remain unconfirmed, causing a train to brake. This breaks the ride's smoothness and energy effectiveness through additional braking and then acceleration.

In the model, when the *OBU* receives a telegram, it puts it into the buffer and starts the driver reaction time counter (real-time clock). For this purpose, it uses a set of agents:

- *ABG*—the balise agent; this agent energizes balises and returns to the train holding a telegram (Figures 8 and 9),
- *ABuf*—the buffer agent; this stores the incoming telegram in the buffer and returns to the *OBU*; it can be a single agent because storing in the buffer occurs instantly; therefore, it manages to return to the *OBU* before the next telegram comes (Figure 8 and Figure 11),
- The three-element agent vector *AClk*[3] for starting the counters; as the driver reaction clocks run independently in parallel, every clock is started by a separate agent (Figures 8 and 10); this agent performs time counting and causes a ring message after the timeout (Figure 10 and Figure 13).
- *ADrv*—the driver agent, which takes the telegrams from the buffer and informs the observer after the reaction (Figures 12 and 13).

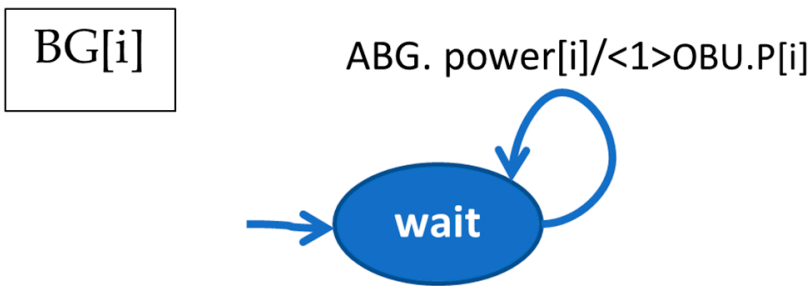

Figure 9. The automaton modeling an ith balise (*BG* [i]).

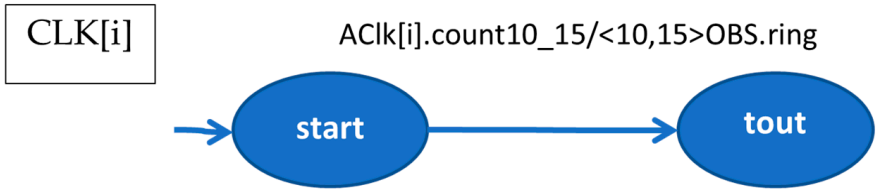

Figure 10. The automaton modeling an ith clock, *CLK* [i].

The time between energizing the balises is calculated based on the velocity range from 5 m/s to 44 m/s (from 14 to 130 s). One second is subtracted from this time for the one-second reaction of a balise.

The balise model simply replies one second after energizing (Figure 9). The balise number is distinguished by the index of energizing message power and (the same) index of the message sent *P*, which models a telegram. The server name comes from balise, which is irrelevant to this article.

The clock model *CLK* (Figure 10) has two states: it waits for a start message from *OBU*, measures 10–15 s using the time duration of the clock agent action, and then informs the observer automaton about the timeout with the ring message. There are three such automata, each of them with its own *AClk* [*i*] agent, as the counted time periods can overlap.

The buffer model (Figure 11) just stores telegrams. The *ABuf* agent from *OBU* inserts a telegram into the buffer when time counting has just started, and the driver acknowledgment removes the telegram from the buffer. Thus, the buffer state reflects the number of unconfirmed telegrams, i.e., the number of telegrams received from balises minus the number of telegrams confirmed by the driver. The telegram messages constitute the vector *mes*[3], but they can come into the buffer in arbitrary order (this property will be important in the second verification).

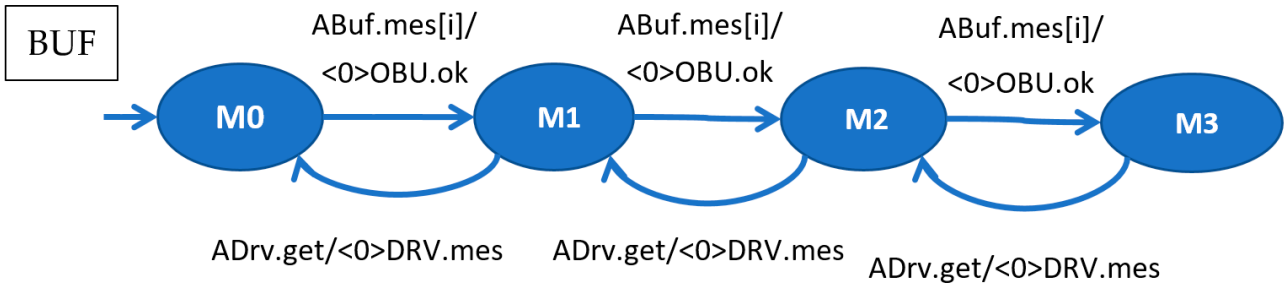

Figure 11. The automaton modeling the buffer, *BUF*.

The driver (Figure 12) "takes" the telegram from the buffer by the message *get*. He or she has 3–5 s for the mental reaction and 2–3 s for the physical reaction pressing a key. Finally, this informs the observer about the successful reaction and loops, asking the buffer about the next telegram. All those actions are invoked by the *ADrv* agent.

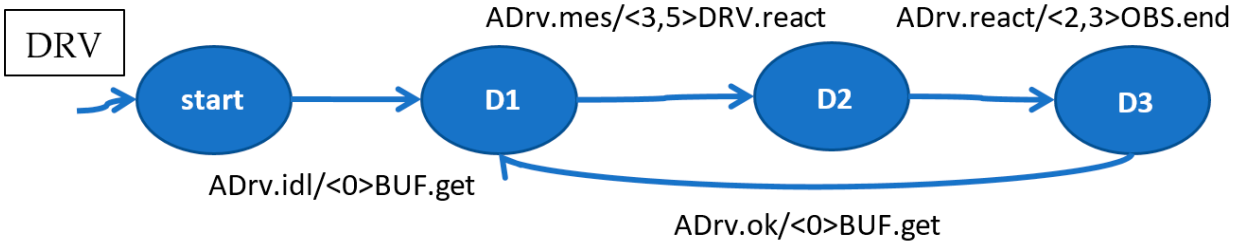

Figure 12. The automaton modeling the driver behavior, *DRV*.

The observer is a server with no equivalent in the physical system (although it can be placed in the system as a diagnostic process using the digital twin concept—Figure 13). If

the observer notices a ring counting the driver's reaction time after the telegram arrives from the balise, it initiates the alarm. This is modeled by the *ERR* observer state. If the driver acknowledges the telegram, the ring of this clock is just ignored—it does not invoke any action. The messages of the driver agent *ADrv* are acknowledged to let him or her continue their work, but the messages of the clock agents *AClk* [i] terminate in the observer as they are no longer needed. The third telegram confirmation finishes the observer's operation in the *SUCCESS* state.

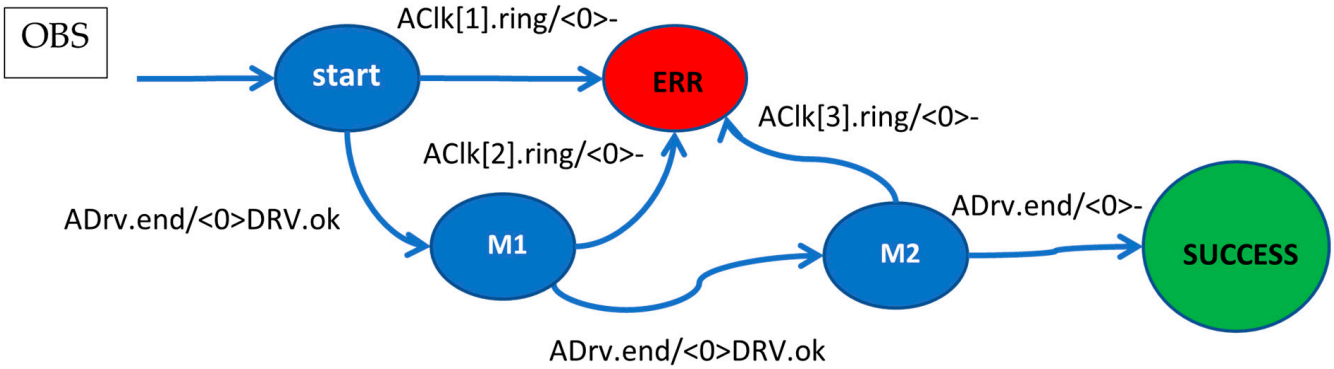

Figure 13. The observer automaton OBS.

7.3. Verification Process

Verification consists in checking whether the observer can reach the *ERR* state or if the observer inevitably reaches the *SUCCESS* state. The Dedan verifier automatically finds deadlocks and distributed terminations in distributed systems specified in the IMDS formalism in a timeless verification. For timed verification, the model is exported to the Uppaal [27] verifier. The temporal property that must be met for eventually reaching the observer termination is **A<>** (OBS.SUCCESS). This formula, which denotes reaching the given state (**<>**) on every path (**A**) is hidden in Dedan for automated verification, but, after exporting to Uppaal, it must be supplied manually. In the future, we plan to send both the model and the formulas to Uppaal. Verification confirmed the safety of the model, which means that the *SUCCESS* state is inevitably reached.

Subsequent checking was done by increasing the train's velocity to a level at which the driver may not be able to confirm the telegram. This means that, at this velocity, the telegrams start to overlap in the queue, so, within the assumed maximum reaction time, the driver confirms the previous telegram. Checking was carried out for the velocity ranges of 44–60 m/s, 60–80 m/s, and 80–100 m/s. Failure occurs only in the range of 80–100 m/s (above 250 km/h), so the line can be considered safe for ordinary trains, but attention should be paid to the case of high-speed rail. Nowadays, trains run in Poland at 200 km/h, but there are plans to increase the velocity to 250 km/h. For this reason, such a line should be reconstructed.

7.4. Modified Example

We also checked whether the driver could confirm an additional telegram coming from other line elements than the balises. This required the following modifications:

- Adding one state to the buffer (maximum four telegrams instead of three),
- Adding one clock to count the confirmation time (the vector of four *CLK* instances instead of three),
- Appropriate observer change, presented in Figure 14.

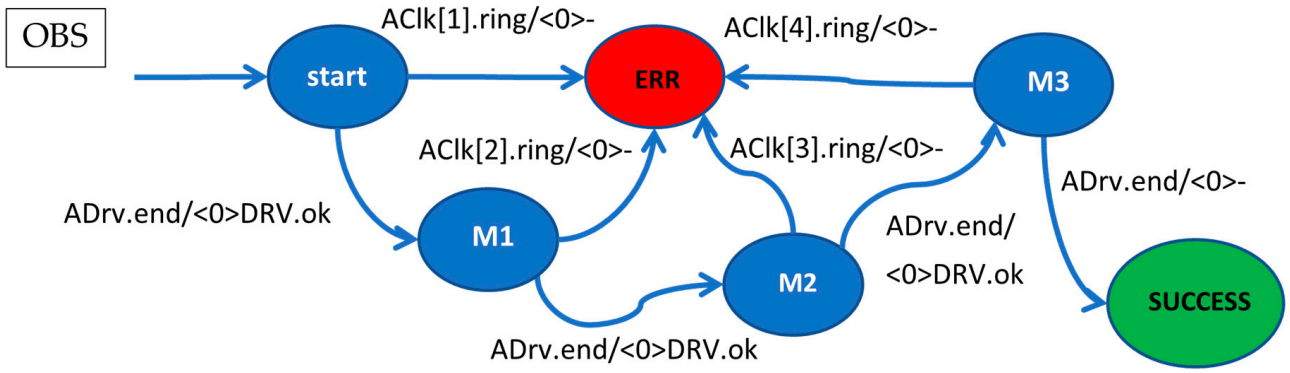

Figure 14. The modified observer automaton *OBS*.

The verification shows the possibility of delaying the driver's reaction, even at the basic speed range of 5–44 m/s. This is very dangerous, because it is usual that the track equipment sends telegrams to the train. Therefore, we can judge that the balises are put on the line too close to each other. The beginning and end of the counterexample leading to the failure are shown in Figures 15 and 16. They are parts of the counterexample produced by the Uppaal verifier. The counterexample is a sequence diagram in which lifelines represent the individual automata. In the picture, we can see from left to right: *BG*, *OBU*, *CLK*[1] ... *CLK*[4], *BUF*, *EXT* (similar to *BG*, but issuing a telegram spontaneously instead of after balise power by the train), and *OBS*. The sources and destinations of every message are shown near the arrows representing the message flow, but they have too small lettering to be observed in this figure. For this reason, one of them is enlarged—a message sent by the *ADrv* agent to the input channel (*ic*) of the *OBS* server, shown in ellipsis. Unfortunately, the message ID is not shown here—it can be observed in another Uppaal window. In this counterexample, it is the OK message. In the future, we plan to pull the counterexample into Dedan and display it in a more convenient form, showing message IDs (formally: service names in a target server).

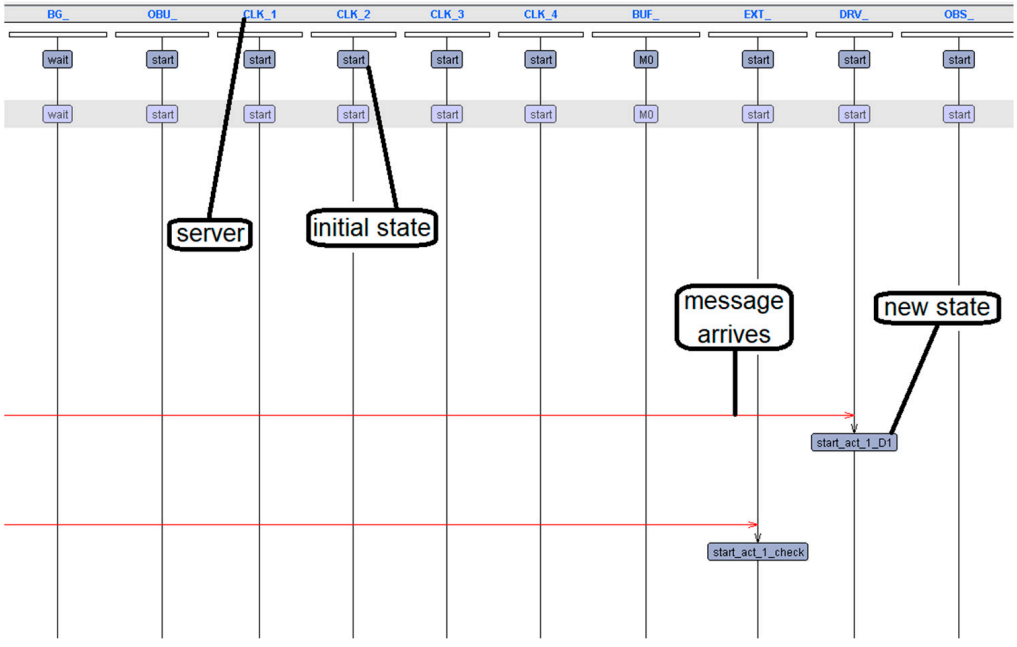

Figure 15. The beginning part of the counterexample, as shown by the Uppaal verifier. The balloons point to specific elements of the counterexample.

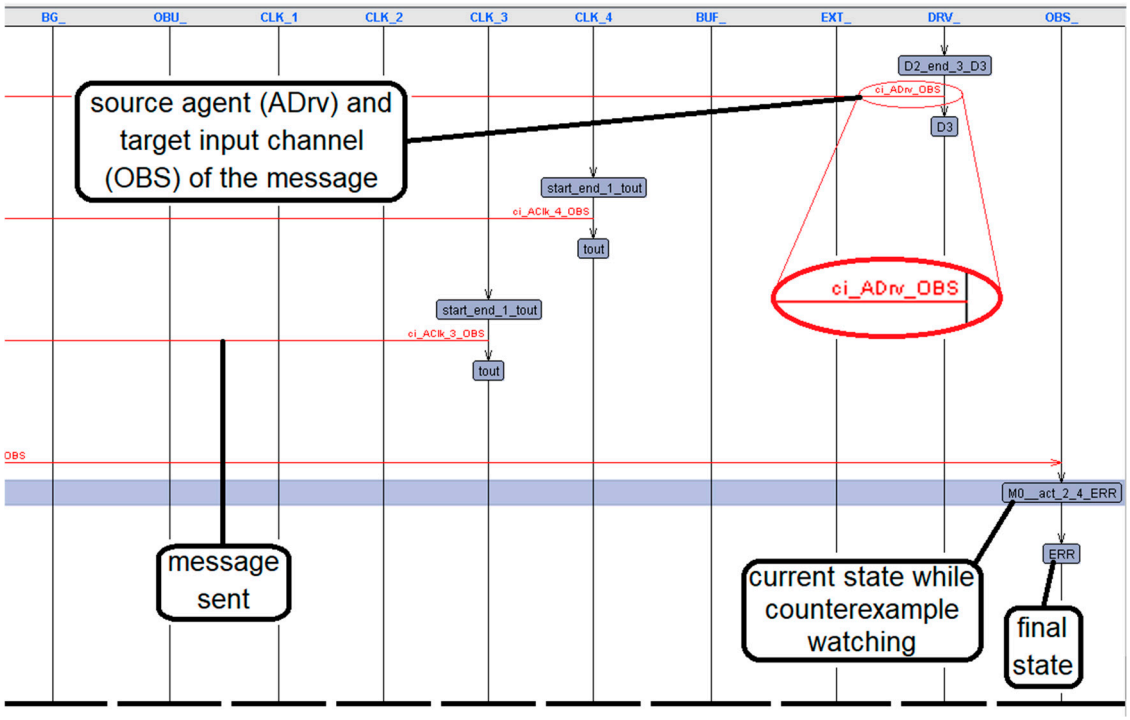

Figure 16. The end part of the counterexample, as shown by the Uppaal verifier. The balloons point to specific elements of the counterexample. The message identifier is enlarged for readability.

8. Conclusions and Further Work

The ETCS system is being implemented on many thousands of kilometers of railway lines throughout Europe. The applications developed are characterized by geographical dispersion and system complexity. Due to the necessity of a smooth run without unwanted braking, it is required that the system operates correctly under any combination of operating conditions. The current application of verification techniques using dynamic tests with actual rolling stock cannot verify all cases. This paper presents a methodology and environment for time-based verification of ETCS systems. Its capabilities allow for verification of the defined conditions by checking all system runs simultaneously. The properties of the formal model and the tools used are specially tailored to distributed systems and automated verification of real-time-dependent systems. These properties are asynchronous and timed verification, graphical modeling based on the local features of distributed components without knowledge of the global state, and automated verification of total and partial deadlocks and termination.

Asynchronous modeling reflects the very nature of distributed systems, where the operation of a device is dependent only on its current state and the external signals acquired, rather than on the states of other devices and their synchronous actions with the device under consideration. This property, connected with real-time analysis, allows for checking of the actual behavior of interconnected devices. For example, we identified the unsafe operation of the ETCS under conditions of accumulated text telegrams, which the driver must acknowledge. In addition to verifying the correctness of the system, thanks to timed verification, we determined the range of train velocities allowing the safe operation of the ETCS. Such an identification of time circumstances can be applied for every safety property. This confirms the thesis set out in the introduction to this article.

In the future, we plan to build a library of typical components of ETCS systems, which would allow for the verification of large cases. We plan to incorporate timed verification into the Dedan tool, for a better presentation of counterexamples, and for the interpretation of results directly on the graphical scheme of a railway line during counterexample simulation.

Discussions are currently underway with ETCS application designers and contractors on the requirements for implementing the proposed solution. The proposed solution is very promising. The information gained from the research carried out shows that, for the structures that ETCS creates, mapping models can be built to form the aforementioned library. Using a library of ready-made models, it will be relatively simple to build a model of an entire application. Another strand of the authors' work supports this approach, focusing on a formal approach to railway infrastructure design. When describing a design using this specification, the temporal model described in this article can be generated algorithmically through transformations. This theme will be the subject of further research and publications.

Another area where the potential value of the proposed solution is evident is in the verification of techniques for optimal control of the running of several consecutive trains in order to improve the energy efficiency of the entire transportation process. Partial results of this work were described in [52]. Model verification will also be investigated from this point of view.

Author Contributions: Conceptualization, J.K., W.B.D. and A.K.; methodology, J.K., W.B.D. and W.G.; software, W.B.D. and W.G.; validation, J.K., W.B.D., W.G. and A.K.; formal analysis, J.K., W.B.D. and W.G.; investigation, J.K. and W.B.D.; resources, J.K. and W.G.; data curation, J.K.; writing—original draft preparation, J.K. and W.B.D.; writing—review and editing, J.K., W.B.D. and W.G.; visualization, W.B.D.; supervision, W.B.D. and A.K.; project administration, A.K.; funding acquisition, A.K. All authors have read and agreed to the published version of the manuscript.

Funding: This paper was co-financed under the research grant of the Warsaw University of Technology supporting the scientific activity in the discipline of Civil Engineering, Geodesy and Transport.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. *32016R0919-Commission Regulation (EU) 2016/919 of 27 May 2016 on the Technical Specification for Interoperability Relating to the 'Control-Command and Signalling' Subsystems of the Rail System in the European Union (Text with EEA Relevance)*; European Comission: Brussels, Belgium, 2016; Available online: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv%3AOJ.L_.2016.158.01.0001.01.ENG&amp%3Btoc=OJ%3AL%3A2016%3A158%3ATOC (accessed on 19 February 2023).
2. *Unisig SUBSET-026 System Requirements Specification*; Issue 3.6.0; European Railway Agency: Valenciennes, France, 2016; Available online: http://webpages.iust.ac.ir/sandidzadeh/Courses/Signalling%202/spec3%20ETCS%20baseline%203%20and%20GSM-R%20baseline%201/Index04%20SUBSET-026%20v360/SUBSET-026-2%20v360.pdf (accessed on 19 February 2023).
3. Munawir, T.I.T.; Samah, A.A.A.; Rosle, M.A.A.; Azlis-Sani, J.; Hasnan, K.; Sabri, S.M.; Ismail, S.M.; Mohd Yunos, M.N.A.; Bin, T.Y. A Comparison Study on the Assessment of Ride Comfort for LRT Passengers. In *Proceedings of the IOP Conference Series: Materials Science and Engineering, Melaka, Malaysia, 6–7 May 2017*; IOP Publishing: Bristol, UK, 2017; Volume 226, pp. 12–39. [CrossRef]
4. Koper, E.; Kochan, A. Testing the Smooth Driving of a Train Using a Neural Network. *Sustainability* **2020**, *12*, 4622. [CrossRef]
5. Kwaśnikowski, J.; Gramza, G. Analiza wybranych zakłóceń w ruchu kolejowym (in Polish). *Probl. Eksploat.* **2007**, *2*, 89–96. Available online: https://bibliotekanauki.pl/articles/257256 (accessed on 19 February 2023).
6. Kwaśnikowski, J.; Gramza, G. Wpływ zakłóceń ruchu i profilu trasy na zużycie energii przez lokomotywę elektryczną EU07 prowadzącą pociąg pasażerski (in Polish). In Proceedings of the 9th TransComp Conference, Zakopane, Poland, 5–8 December 2005; Prace Naukowe Politechniki Radomskiej–Elektryka: Radom, Poland, 2005; pp. 131–136.
7. Nolte, R.; Würtenberger, F. *EVENT-Evaluation of Energy Efficiency Technologies for Rolling Stock and Train Operation of Railways*; Institute for Futures Studies and Technology Assessment: Berlin, Germany, 2003; Available online: https://www.forschungsinformationssystem.de/servlet/is/117125 (accessed on 19 February 2023).
8. Dąbrowa-Bajon, M. Podstawy sterowania ruchem kolejowym. In *Funkcje, Wymagania, Zarys Technik*; Oficyna Wydawnicza Politechniki Warszawskiej: Warsaw, Poland, 2015; ISBN 978-83-7814-320-8.
9. Sabatier, D. Using Formal Proof and B Method at System Level for Industrial Projects. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, Proceedings of the RSSRail 2016, Paris, France, 28–30 June 2016*; Lecomte, T., Pinger, R., Romanovsky, A., Eds.; Springer: Cham, Switzerland, 2016; LNPSE Volume 9707, pp. 20–31. [CrossRef]
10. Comptier, M.; Deharbe, D.; Perez, J.M.; Mussat, L.; Pierre, T.; Sabatier, D. Safety Analysis of a CBTC System: A Rigorous Approach with Event-B. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, Proceedings of the RSSRail 2017, Pistoia, Italy, 14–16 November 2017*; Fantechi, A., Lecomte, T., Romanovsky, A., Eds.; Springer: Cham, Switzerland, 2017; LNCS Volume 10598, pp. 148–159. [CrossRef]

11. James, P.; Moller, F.; Nguyen, H.N.; Roggenbach, M.; Schneider, S.; Treharne, H. Techniques for Modelling and Verifying Railway Interlockings. *Int. J. Softw. Tools Technol. Transf.* **2014**, *16*, 685–711. [CrossRef]
12. Idani, A.; Ledru, Y.; Ait Wakrime, A.; Ben Ayed, R.; Bon, P. Towards a Tool-Based Domain Specific Approach for Railway Systems Modeling and Validation. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, Proceedings of the RSSRail 2019, Lille, France, 4–6 June 2019*; Collart-Dutilleul, S., Lecomte, T., Romanovsky, A., Eds.; Springer: Cham, Switzerland, 2019; LNPSE Volume 11495, pp. 23–40. [CrossRef]
13. Fantechi, A.; Gori, G.; Haxthausen, A.E.; Limbrée, C. Compositional Verification of Railway Interlockings: Comparison of Two Methods. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, RSSRail 2022, Proceedings of the RSSRail 2022, Paris, France, 1–2 June 2022*; Collart-Dutilleul, S., Haxthausen, A.E., Lecomte, T., Eds.; Springer: Cham, Switzerland, 2022; LNCS Volume 13294, pp. 3–19. [CrossRef]
14. Ghosh, S.; Das, A.; Basak, N.; Dasgupta, P.; Katiyar, A. Formal Methods for Validation and Test Point Prioritization in Railway Signaling Logic. *IEEE Trans. Intell. Transp. Syst.* **2017**, *18*, 678–689. [CrossRef]
15. Iliasov, A.; Laibinis, L.; Taylor, D.; Lopatkin, I.; Romanovsky, A. Safety Invariant Verification That Meets Engineers' Expectations. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, Proceedings of the RSSRail 2022, Paris, France, 1–2 June 2022*; Collart-Dutilleul, S., Haxthausen, A.E., Lecomte, T., Eds.; Springer: Cham, Switzerland, 2022; LNCS Volume 13294, pp. 20–31. [CrossRef]
16. Kamburjan, E.; Hähnle, R. Deductive Verification of Railway Operations. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, Proceedings of the RSSRail 2017, Pistoia, Italy, 14–16 November 2017*; Fantechi, A., Lecomte, T., Romanovsky, A., Eds.; Springer: Cham, Switzerland, 2017; LNCS Volume 10598, pp. 131–147. [CrossRef]
17. Carrasquel, J.C.; Morales, A.; Villapol, M.E. Prosega/CPN: An Extension of CPN Tools for Automata-Based Analysis and System Verification. *Proc. Inst. Syst. Program. RAS* **2018**, *30*, 107–128. [CrossRef] [PubMed]
18. Sun, P.; Collart-dutilleul, S.; Bon, P. A Model Pattern of Railway Interlocking System by Petri Nets. In Proceedings of the 2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), Budapest, Hungary, 3–5 June 2015; pp. 442–449. [CrossRef]
19. Parillaud, C.; Fonteneau, Y.; Belmonte, F. Interlocking Formal Verification at Alstom Signalling. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, Proceedings of the RSSRail 2019, Lille, France, 4–6 June 2019*; Collart-Dutilleul, S., Lecomte, T., Romanovsky, A., Eds.; Springer: Cham, Switzerland, 2019; LNCS Volume 11495, pp. 215–225. [CrossRef]
20. Salierno, G.; Morvillo, S.; Leonardi, L.; Cabri, G. Specification and Verification of Railway Safety-Critical Systems Using TLA +: A Case Study. In Proceedings of the 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Virtual Conference, 4–6 November 2020; pp. 207–212. [CrossRef]
21. Ghazel, M. Formalizing a Subset of ERTMS/ETCS Specifications for Verification Purposes. *Transp. Res. Part C Emerg. Technol.* **2014**, *42*, 60–75. [CrossRef]
22. Mammar, A.; Frappier, M.; Tueno Fotso, S.J.; Laleau, R. A Formal Refinement-Based Analysis of the Hybrid ERTMS/ETCS Level 3 Standard. *Int. J. Softw. Tools Technol. Transf.* **2020**, *22*, 333–347. [CrossRef]
23. Limbrée, C.; Cappart, Q.; Pecheur, C.; Tonetta, S. Verification of Railway Interlocking-Compositional Approach with OCRA. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, Proceedings of the RSSRail 2016, Paris, France, 28–30 June 2016*; Lecomte, T., Pinger, R., Romanovsky, A., Eds.; Springer: Cham, Switzerland, 2016; LNPSE Volume 9707, pp. 134–149. [CrossRef]
24. Halchin, A.; Feliachi, A.; Singh, N.K.; Ait-Ameur, Y.; Ordioni, J. B-PERFect. Applying the PERF Approach to B Based System Developments. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, Proceedings of the RSSRail 2017, Pistoia, Italy, 14–16 November 2017*; Fantechi, A., Lecomte, T., Romanovsky, A., Eds.; Springer: Cham, Switzerland, 2017; LNCS Volume 10598, pp. 160–172. [CrossRef]
25. Platzer, A.; Quesel, J.-D. Logical Verification and Systematic Parametric Analysis in Train Control. In *Hybrid Systems: Computation and Control, Proceedings of the HSC2008, St. Louis, MO, USA, 22–24 April 2008*; Egerstedt, M., Mishra, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; LNCS Volume 4981, pp. 646–649. [CrossRef]
26. Han, X.; Tang, T.; Lv, J.; Wang, H. Failure Analysis of Chinese Train Control System Level 3 Based on Model Checking. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification, Proceedings of the RSSRail 2016, Paris, France, 28–30 June 2016*; Lecomte, T., Pinge, R., Romanovsky, A., Eds.; Springer: Cham, Switzerland, 2016; LNPSE Volume 9707, pp. 95–105. [CrossRef]
27. Larsen, K.G.; Lorber, F.; Nielsen, B. 20 Years of UPPAAL Enabled Industrial Model-Based Validation and Beyond. In *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice, Proceedings of the ISoLA 2018, Limassol, Cyprus, 5–9 November 2018*; Margaria, T., Steffen, B., Eds.; Springer: Cham, Switzerland, 2018; LNCS Volume 11247, pp. 212–229. [CrossRef]
28. Kim, J.H.; Larsen, K.G.; Nielsen, B.; Mikučionis, M.; Olsen, P. Formal Analysis and Testing of Real-Time Automotive Systems Using UPPAAL Tools. In *Formal Methods for Industrial Critical Systems, Proceedings of the FMICS 2015, Oslo, Norway, 22–23 June 2015*; Núñez, M., Güdemann, M., Eds.; Springer: Cham, Switzerland, 2015; LNPSE Volume 9128, pp. 47–61. [CrossRef]
29. Ben-Rayana, S.; Bozga, M.; Bensalem, S.; Combaz, J. RTD-Finder: A Tool for Compositional Verification of Real-Time Component-Based Systems. In *Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of the TACAS 2016,*

Eindhoven, The Netherlands, 2–8 April 2016; Chechik, M., Raskin, J.-F., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; LNCS Volume 9636, pp. 394–406. [CrossRef]
30. Daskaya, I.; Huhn, M.; Milius, S. Formal Safety Analysis in Industrial Practice. In *Formal Methods for Industrial Critical Systems, Proceedings of the FMICS 2011, Trento, Italy, 29–30 August 2011*; Salaün, G., Schätz, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 68–84. [CrossRef]
31. Lahtine, J. *Model Checking Timed Safety Instrumented Systems*; Helsinki University of Technology, Department of Information and Computer Science: Helsinki, Finland, 2008; 68p, Available online: https://aaltodoc.aalto.fi/handle/123456789/874 (accessed on 19 February 2023).
32. Holzmann, G.J. The Model Checker SPIN. *IEEE Trans. Softw. Eng.* **1997**, *23*, 279–295. [CrossRef]
33. Alur, R.; Dill, D.L. A Theory of Timed Automata. *Theor. Comput. Sci.* **1994**, *126*, 183–235. [CrossRef]
34. Mazuelo, C.L. Automatic Model Checking of UML Models. Master's Thesis, Bern University, Informatics and Applied Mathematics Institute, Bern, Switzerland, 2008. Available online: http://www.iam.unibe.ch/tilpub/2008/lar08.pdf (accessed on 19 February 2023).
35. Fahland, D.; Favre, C.; Koehler, J.; Lohmann, N.; Völzer, H.; Wolf, K. Analysis on Demand: Instantaneous Soundness Checking of Industrial Business Process Models. *Data Knowl. Eng.* **2011**, *70*, 448–466. [CrossRef]
36. Joosten, S.J.C.; Julien, F.V.; Schmaltz, J. WickedXmas: Designing and Verifying on-Chip Communication Fabrics. In Proceedings of the 3rd International Workshop on Design and Implementation of Formal Tools and Systems, DIFTS'14, Lausanne, Switzerland, 20 October 2014; Technische Universiteit Eindhoven: Eindhoven, The Netherlands, 2014; pp. 1–8. Available online: https://pure.tue.nl/ws/files/3916267/889737443709527.pdf (accessed on 19 February 2023).
37. Yousefian, R.; Rafe, V.; Rahmani, M. A Heuristic Solution for Model Checking Graph Transformation Systems. *Appl. Soft Comput.* **2014**, *24*, 169–180. [CrossRef]
38. Daszczuk, W.B. Static and Dynamic Verification of Space Systems Using Asynchronous Observer Agents. *Sensors* **2021**, *21*, 4541. [CrossRef] [PubMed]
39. Lutz, M.J. Modeling Software the Alloy Way. In Proceedings of the 2013 IEEE Frontiers in Education Conference (FIE), Oklahoma City, OK, USA, 23–26 October 2013; p. 3. [CrossRef]
40. Krystosik, A. Embedded Systems Modeling Language. In Proceedings of the 2006 International Conference on Dependability of Computer Systems, DepCos-RELCOMEX '06, Szklarska Poręba, Poland, 25–27 May 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 27–34. [CrossRef]
41. Daszczuk, W.B. Specification and Verification in Integrated Model of Distributed Systems (IMDS). *Computers* **2018**, *7*, 65. [CrossRef]
42. Daszczuk, W.B. Communication and Resource Deadlock Analysis Using IMDS Formalism and Model Checking. *Comput. J.* **2017**, *60*, 729–750. [CrossRef]
43. Karolak, J.; Daszczuk, W.B.; Grabski, W.; Kochan, A. Temporal Verification of Relay-Based Railway Traffic Control Systems Using the Integrated Model of Distributed Systems. *Energie* **2022**, *15*, 9041. [CrossRef]
44. Daszczuk, W.B. Asynchronous Specification of Production Cell Benchmark in Integrated Model of Distributed Systems. In Proceedings of the 23rd International Symposium on Methodologies for Intelligent Systems, ISMIS 2017, Warsaw, Poland, 26–29 June 2017; Studies in Big Data, Volume 40. Bembenik, R., Skonieczny, L., Protaziuk, G., Kryszkiewicz, M., Rybinski, H., Eds.; Springer International Publishing: Cham, Switzerland, 2019; pp. 115–129. [CrossRef]
45. Mazzanti, F.; Ferrari, A.; Spagnolo, G.O. Towards Formal Methods Diversity in Railways: An Experience Report with Seven Frameworks. *Int. J. Softw. Tools Technol. Transf.* **2018**, *20*, 263–288. [CrossRef]
46. Daszczuk, W.B. Modeling and Verification of Asynchronous Systems Using Timed Integrated Model of Distributed Systems. *Sensors* **2022**, *22*, 1157. [CrossRef] [PubMed]
47. Reniers, M.A.; Willemse, T.A.C. Folk Theorems on the Correspondence between State-Based and Event-Based Systems. In Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, 22–28 January 2011; Springer: Berlin/Heidelberg, Germany, 2011; LNCS Volume 6543, pp. 494–505. [CrossRef]
48. Daszczuk, W.B. Using the Dedan Program. In *Integrated Model of Distributed Systems*; Springer Nature: Cham, Switzerland, 2020; pp. 87–97. [CrossRef]
49. Balan, M.S. Serializing the Parallelism in Parallel Communicating Pushdown Automata Systems. *Electron. Proc. Theor. Comput. Sci.* **2009**, *3*, 59–68. [CrossRef]
50. Bollig, B.; Leucker, M. Message-Passing Automata Are Expressively Equivalent to EMSO Logic. In Proceedings of the 15th International Conference CONCUR 2004-Concurrency Theory, London, UK, 31 August–3 September 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 146–160. [CrossRef]
51. Daszczuk, W.B. Graphic Modeling in Distributed Autonomous and Asynchronous Automata (DA^3). *Softw. Syst. Model.* **2021**, *20*, 363–398. [CrossRef]
52. Szkopiński, J.; Kochan, A. Energy Efficiency and Smooth Running of a Train on the Route While Approaching Another Train. *Energies* **2021**, *14*, 7593. [CrossRef]