Proceedings of the International Conference on
Formal Methods and Foundations of Artificial Intelligence
Eszterházy Károly Catholic University
Eger, Hungary, June 5–7, 2025
pp. 201–213

DOI: 10.17048/fmfai.2025.201

Automata-based representation of coordination for distributed reactive systems*

Richárd Szabó, Dóra Cziborová

Budapest University of Technology and Economics
Department of Artificial Intelligence and Systems Engineering
Budapest, Hungary
{szabor,cziborova}@mit.bme.hu

Abstract. Modern cyber-physical systems (CPS) are distributed reactive real-time systems used in many critical application domains, such as automotive or railway systems, so ensuring their correctness is essential. Formal verification can exhaustively explore the behavior of the formal representation of CPS to ensure its reliability. Engineering modeling tools provide separate modeling constructs for the different aspects of the systems, e.g., behavior, architecture, and scheduling, but lack formal composition, making system-level verification difficult. Formal modeling tools, e.g., Lingua Franca or the Gamma Statechart Composition Framework, are efficient in describing component behavior, and they provide formal composition semantics of the subsystems. However, the provided composition patterns in these languages are not general, so incorporating the precise coordination, scheduling, and interaction aspects requires the modification of the component models by encoding the coordination into the components. In this paper, we present a configurable formal description approach for the coordination of distributed critical systems. We extend the well-known timed automata formalism to coordinate the execution of the components by directly reusing the formal models of the components, making the coordination of the system components a first-class citizen.

Keywords: formalization, coordination, distributed systems, system modeling

*The project supported by the Doctoral Excellence Fellowship Programme (DCEP) is funded by the National Research Development and Innovation Fund of the Ministry of Culture and Innovation and the Budapest University of Technology and Economics.

1. Introduction

The modeling and verification of modern cyber-physical systems (CPS) present several unique challenges that arise due to the integration of distributed heterogeneous components. These systems integrate physical systems with HW/SW components across varying platforms and locations. One challenge is to account for the timing of distributed components, which can be influenced by the communication network between the distributed components (messages can be delayed or lost). CPS are often used in critical application domains, e.g., the railway or the automotive industry. One way to ensure the reliability and correctness of these systems is to apply formal verification techniques like model checking, which systematically explores the state space of the system.

Various modeling and formal languages are developed to help engineers use formal verification: engineering modeling tools, such as AUTOSAR, aim for efficient engineering and provide separate modeling constructs for the different aspects of the systems, e.g., behavior, architecture, and scheduling. However, no formal composition is defined, so it is hard to target the system-level behavior with formal verification. On the other hand, formal modeling tools, e.g., Lingua Franca or the Gamma Statechart Composition Framework, are efficient in describing component behavior and they provide formal composition semantics of the subsystems. However, the provided composition patterns in these languages are not general, so incorporating the precise coordination, scheduling, and interaction aspects in these tools requires the modification of the component models and the encoding of the coordination into the components. Additionally, in distributed CPS, the underlying formal models of the communication and the subsystems often vary based on the application domain, network architecture, and system requirements, thus, a configurable solution is needed to model the coordination of the system.

In this paper, we present a configurable formal description approach for the coordination of distributed critical systems. We extend the well-known timed automata formalism to coordinate the execution of the components by directly reusing the formal models of the components, making the coordination of the system components a first-class citizen.

1.1. Motivating example: Steer-by-Wire

In our previous works, we presented an approach to model time-dependent behaviors in complex distributed systems and showed its applicability with a Steer-by-Wire (SbW) system inspired by our industrial partner [5], and we investigated how the coordination of the SbW system can be modeled [10].

In a SbW system to provide steering functionality, the road wheels are actuated via a control loop. Unlike classic Electric Power Assisted Steering (EPAS) systems, in the case of an SbW system, there are no direct mechanical connections between the steering wheel and the road wheels. The angle of the steering wheel is measured by multiple sensors, and the road wheels are actuated by the *Road Wheel Actuator*

(RWA) subsystems based on the measurements. Since actuating the road wheel is a critical function, a Master Selection Protocol (MSP) was developed to control the actuation of the road wheels. The MSP assigns which of the redundant RWAs must be used to control the road wheels based on the availability and correctness of their sensor measurements. In the presented case study, there are two RWAs and four sensors, as depicted in Figure 1.

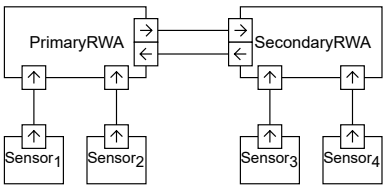

Figure 1. Architecture of the SbW system.

The components of the SbW system are distributed: the components are deployed on separate *electrical/electronic* (E/E) subsystems, and the components communicate over redundant *Controller Area Network* (CAN) buses and private communication buses. The execution of the E/E subsystems is triggered by their clocks, and the clocks can deviate from each other. Furthermore, the CAN buses are used by other components of the system. The combination of these properties can cause delayed messages; in extreme cases, this can lead to comparing sensor measurements originating from different execution cycles, possibly deeming correct measurements as faulty. Selecting both RWAs as master or selecting a failed RWA as the master can lead to accidents. To verify that the deviations of the E/E subsystem clocks cannot cause erroneous master selection, a sufficient formal representation of the coordination of the subsystems is needed. In order to formally model and verify such a complex system, we need a configurable modeling language that allows us to describe the possible execution scenarios, and while the previous works presented in Subsection 2.1 all have their strength, they lack the option to model the possible execution scenarios as first-class citizens.

2. Background and related work

2.1. Modeling complex distributed systems

There are several widely used modeling languages for defining the architecture and behavior of a system. These languages have varying levels of precision in their semantics. To use mathematical tools for the systematic examination of a system's design, a formal model of the system with mathematically precise semantics is needed [3].

Lingua Franca [7] is a coordination language to model concurrent reactive components (reactors). Lingua Franca works with both logical time (discrete ordering of events) and physical time (timestamps). The goal of Lingua Franca is to provide a deterministic model of the system by utilizing the priorities of the reactors, the dependencies between the reactors, and the logical and physical timing of events. Even though the goal of the authors of Lingua Franca is to model deterministic systems, nondeterminism is allowed if explicitly required by the engineer.

The *Gamma Statechart Composition Framework* [8] focuses on bridging the gap between higher-level (engineering) models and lower-level (formal) models and facilitates the modeling and verification of component-based reactive systems. The framework provides semantic-preserving model transformations between higher-level and lower-level modeling languages. The framework supports the hierarchical composition of components, which enables the engineers (1) to break down the system into smaller, reusable subsystems or (2) connect distributed components to provide higher-level system functionalities.

Similarly to the coordination automata formalism presented in Subsection 3.1, Norström et al. [9] present an extension of the timed automata. Their formalism, the *task automata*, enables the schedulability analysis of tasks by performing reachability analysis. In [1], the authors present a timed labeled transition system with invariants to define controlling and priorities of applications. The requirements of the system are mapped to specific transitions, thus requiring a white-box model of the system. Our proposed formalism considers the coordinated subsystems as black-box components since the transitions of the subsystems are not modified.

2.2. The TXSTS modeling formalism

The *timed extended symbolic transition system* (TXSTS) formalism (proposed in [4] as an extension of [6]) is an intermediate modeling formalism with high-level language constructs suitable for representing complex engineering models. Nevertheless, it is compatible with model checking algorithms that are usually based on low-level formal models.

TXSTS models contain *data variables* to represent data-dependent behavior, while timed behavior is modeled by *clock variables*.

Clock variables are continuous, non-negative variables. They are initialized to zero and incremented equally. Most timed analysis techniques allow the comparison of clocks with other clocks or integer constants, and resetting them to integer constants. If there are rational constants in the model, all constants should be multiplied by the least common multiple of denominators [2]. For a set of clocks C, let $\mathcal{G}(C)$ denote the set of clock constraints in the form of $c_i \sim n$ or $c_i - c_j \sim n$, where $c_i, c_j \in C$, $\sim \in \{<, \leq, ==, \geq, >\}$, and $n \in \mathbb{N}_0$. Furthermore, let $\mathcal{A}(C)$ denote the set of clock assignments in the form of $c_i := n$, where $c_i \in C$, and $n \in \mathbb{N}_0$.

A timed extended symbolic transition system is a tuple $TXSTS = \langle V_D, V_C, V_{ctrl}, val^0, init, env, tran \rangle$ where

- V_D and V_C are finite sets of data variables and clock variables, $V_{ctrl} \subseteq V_D$ is a set of *control variables* that may be handled differently by the algorithms;
- val^0 is the initial valuation over V_D that maps each variable $x \in V_D$ to the initial value of the variable, or $\top$ if unknown;
- $init, env, tran \subseteq \mathcal{O}$ are sets of operations representing the *initialization*, *environment* and *internal operation sets*.

The operation sets consist of one or more operations taken from a set of operations $\mathcal{O}$. When executing an operation set, the operation to be executed is selected from the operation set in a nondeterministic manner. The set of operations $\mathcal{O}$ contains the following types of operations:

- *Assumptions* have the form $[\varphi]$, where φ is a Boolean combination of predicates over V_D and clock constraints of $\mathcal{G}(V_C)$;
- *Data assignments* have the form $x := \varphi$, where $x \in V_D$, and φ is an expression of the same type as x, that may contain variables of V_D and clock constraints of $\mathcal{G}(V_C)$;
- *Clock resets* are clock assignments of $\mathcal{A}(V_C)$;
- *Havoc operations* are denoted by $havoc(x)$, which is a nondeterministic assignment to a data variable $x \in V_D$;
- *Delays* are denoted simply by *delay*, it is a nondeterministic but equal incrementation of all clocks in V_C;
- A *no-op* is denoted by *skip*;
- A *sequence* is a composite operation $op_1, op_2, \ldots, op_n$, where $op_i \in \mathcal{O}$ for all $1 \leq i \leq n$, the operations are executed one after the other;
- *Nondeterministic choices* have the form $\{op_1\}$ *or* $\{op_2\}$ *or* ... *or* $\{op_n\}$, they are composite operations, where $op_i \in \mathcal{O}$ for all $1 \leq i \leq n$, from which exactly one operation is executed, chosen in a nondeterministic manner;
- *Conditional operations* of the form *if*(φ) *then* $\{op_1\}$ *else* $\{op_2\}$ are composite operations, where φ is a Boolean combination of predicates over V_D and clock constraints of $\mathcal{G}(V_C)$, if φ holds then $op_1 \in \mathcal{O}$ is executed, otherwise $op_2 \in \mathcal{O}$;
- A *loop* is a composite operation of the form *for* i *from* φ_a *to* φ_b *do* $\{op\}$, where i is an integer variable, φ_a, φ_b are expressions that evaluate to integers, serving as the lower and upper bound for the loop variable i, and $op \in \mathcal{O}$.

A *state* of a TXSTS model is a tuple $\langle val_D, val_C, \tau \rangle$, where val_D is a valuation over V_D, val_C is a valuation over V_C and $\tau \in \{init, env, tran\}$ is an operation set, which is the only operation set that can be executed in this state.

The order of execution of the operation sets is fixed in TXSTS models. The operation set *init* is executed only once, in the initial state. Sets *env* and *tran* are executed in an alternating manner, but only after *init*, starting with *env*.

The semantics of TXSTS operations is straightforward, however, one can refer to [4] for the detailed semantics.

When components of the same system are modeled as separate TXSTS models, we consider them as a *network of TXSTS models*, where time advances equally, as introduced in [10]. We will denote the state of a network $N = \langle TXSTS_1, TXSTS_2, \ldots, TXSTS_n \rangle$ by $S_N = \langle S_1, S_2, \ldots, S_n \rangle$, where S_i is a state of $TXSTS_i$ for each $1 \leq i \leq n$.

3. Formal modeling of coordination

In this section, we propose a new formalism, the *coordination automata*, to ease the challenges of modeling the possible execution orders of the system. We present the semantics of the coordination automata using the TXSTS modeling formalism, which is an extension of the inner modeling formalism used by the Gamma Framework. Finally, we present the coordination automaton of the motivating example.

3.1. Coordination automata

We propose the new coordination automata formalism, which extends timed automata [2] with notations referencing other formal models, e.g., TXSTS models. These models can be referenced on the edges of the coordination automaton, denoting that the given component is scheduled for execution.

A coordination automaton over the set of clocks C disjoint from the sets of clock variables of the referenced formal models and a set of TXSTS models $\mathcal{M} = \{TXSTS_1, TXSTS_2, \ldots, TXSTS_n\}$ is a tuple $CA = \langle L, l_0, E \rangle$, where

- L is a finite set of locations, with initial location $l_0 \in L$;
- $E \subseteq L \times \mathcal{M} \times 2^{\mathcal{G}(C)} \times 2^{\mathcal{A}(C)} \times L$ is a set of directed edges.

The semantics of coordination automata is similar to that of timed automata. With the set of TXSTS models $\mathcal{M}$ forming a network N, the semantics of the corresponding coordination automaton is defined by a transition system with a set of states $\mathcal{S}_{CA} \subseteq L \times Val_C \times \mathcal{S}_N$ where Val_C is the set of clock valuations over C, and $\mathcal{S}_N$ is the set of states of N.

The initial states of the transition system form a set $\{\langle l_0, val_C^{\Delta(S_N)}, S_N \rangle \mid S_N \in \mathcal{S}_N^{init}, \forall c \in C\colon val_C^{\Delta(S_N)}(c) = \Delta(S_N)\}$ where $\mathcal{S}_N^{init}$ is the set of possible states of N after executing the *init* operation set of all TXSTS models of N in the order given by the definition of the network, during which $\Delta(S_N)$ delay takes place if the resulting state of the network is S_N.

In the transition system there are two kinds of transitions:

- An action transition $\langle l, val_C, S_N \rangle \xrightarrow{m,G,A} \langle l', val'_C, S'_N \rangle$, where $m \in \mathcal{M}$, $G \subset \mathcal{G}(C)$ and $A \subset \mathcal{A}(C)$, is enabled iff the following conditions are satisfied:
  - $\langle l, m, G, A, l' \rangle \in E$;
  - val_C satisfies all clock constraints in G;
  - $\forall\, c \in C\colon\ val'_C(c) = z$ if $(c := z) \in A$, otherwise $val'_C(c) = val_C(c) + \Delta$, where $\Delta \in \mathbb{R}_{\geq 0}$ would be the value of a non-resettable clock in S'_N that was set to 0 in S_N;
  - S'_N is a result of executing the *env* and *tran* operation sets of m on S_N.
- A delay transition $\langle l, val_C, S_N \rangle \xrightarrow{\Delta \in \mathbb{R}_{\geq 0}} \langle l', val'_C, S'_N \rangle$ is enabled iff:

- $l' = l$;
- $\forall\ c \in C\colon\ val'_C(c) = val_C(c) + \Delta$; and
- $\forall\ m \in \mathcal{M},\ c \in V_C^m\colon\ val'_m(c) = val_m(c) + \Delta$, where V_C^m is the set of clock variables of m, and val_m, val'_m are the clock valuations of m in S_N and S'_N, respectively.

3.2. Sequential and unordered execution

To allow for more compact representation of the desired coordination of a distributed system, we provide two syntactic constructs for expressing the sequential and unordered execution of multiple components on a single edge of the coordination automaton.

The sequential execution, denoted as $seq\{m_1, m_2, \ldots, m_n\}$ on an edge from l_i to l_j, is equivalent to a path from l_i to l_j of length n, where the edges are annotated with components $m_1, m_2, \ldots, m_n$, respectively, with guards of the original sequential edge repeated on all edges along the path, and the last edge of the path containing the clock assignments of the original edge. I.e., the clock assignments are executed after the sequential execution of all referenced components, while the guards hold invariably until the scheduling of the last component.

Unordered execution is denoted as $unord\{m_1, m_2, \ldots, m_n\}$. Such an unordered execution on an edge from l_i to l_j is equivalent to having sequential paths from l_i to l_j for all permutations of $m_1, m_2, \ldots, m_n$. As with sequential paths, guards of the original unordered edge are repeated on all edges, and the last edges of the paths contain the clock assignments of the original edge. One should note that unordered execution results in $n!$ possible orderings of components, therefore advanced techniques are required for a systematic analysis of the system. Nonetheless, it provides a compact and easy-to-understand way for engineers to represent and communicate the coordination of a complex system.

4. Formal modeling of a distributed reactive system with coordinated components

The systematic examination of a system requires a formal representation. In this section, we describe a transformation of coordination automata and its referenced TXSTS models to a single TXSTS model, which results in a suitable input for model checking algorithms.

The idea of the transformation is mapping the coordination automaton to the environment operation set of an "integrated" TXSTS. The initialization (*init*), environment (*env*) and internal (*tran*) operation sets of the individual components referenced by the coordination automaton are then embedded in the corresponding operation sets of the integrated TXSTS. In a step of this TXSTS, only one of the components is selected for execution, based on a variable that is set by the transformed coordination automaton.

The mapping of the coordination automaton to the TXSTS formalism introduces at least two new control variables. The new variable *scheduled* represents the component that should be executed next by the *env* and *tran* operation sets. The second new variable, *coordState*, represents the current location of the coordination automaton, with initial value l_0. It may also stand for intermediate locations that are introduced by the transformation and represent the ongoing execution of a sequential or unordered edge.

4.1. Operation sets of the integrated TXSTS model

Let $init_m$, env_m and $tran_m$ denote the initialization, environment and internal operation sets of the TXSTS model $m \in \mathcal{M}$. We will also write $init_m$, env_m and $tran_m$ to denote a nondeterministic choice operation (see Subsection 2.2) of all operations in the given operation set. This will be used to represent an operation set as a single (nondeterministic) operation.

The *init* operation set of the integrated TXSTS model is a sequence of the *init* operation sets of all components: $init_{m_1}, init_{m_2}, \ldots, init_{m_{|\mathcal{M}|}}$.

The environment operation set is a sequence of two operations: a step of the coordination automaton and a step of one of the components.

The step of the coordination automaton is a sequence of a *delay* operation and a nondeterministic choice. In this nondeterministic choice, there is a branch for each location (both intermediate locations and locations of the coordination automaton). Each branch starts with the assumption $[coordState == l]$ (thus only one branch can be executed, determined by the value of *coordState*), which is followed by another nondeterministic choice. In this embedded nondeterministic choice, each branch represents an outgoing edge from location l.

I.e., if the outgoing edges of a location l_i are $e_{i,1}, e_{i,2}, \ldots, e_{i,n_i}$ and $(\!|e|\!)$ means the representation of an edge e in the TXSTS formalism, then the mapping of a step of the coordination automaton with locations l_i ($1 \leq i \leq k$, including intermediate locations) is the following:

$$\begin{aligned} &delay,\ \{\ [coordState == l_0],\ (\!|e_{0,1}|\!)\ or\ (\!|e_{0,2}|\!)\ or\ \ldots\ or\ (\!|e_{0,n_0}|\!)\ \} \\ &\quad or\ \{\ [coordState == l_1],\ (\!|e_{1,1}|\!)\ or\ (\!|e_{1,2}|\!)\ or\ \ldots\ or\ (\!|e_{1,n_1}|\!)\ \}\ or\ \ldots \\ &\qquad \ldots\ or\ \{\ [coordState == l_k],\ (\!|e_{k,1}|\!)\ or\ (\!|e_{k,2}|\!)\ or\ \ldots\ or\ (\!|e_{k,n_k}|\!)\ \}. \end{aligned}$$

The mapping of an edge e from l to l' with clock constraints $G \subset \mathcal{G}(C)$ and clock assignments $A \subset \mathcal{A}(C)$ depends on the kind of execution it represents.

If e is annotated by a *single* component $m \in \mathcal{M}$, then it is mapped to

$$[\bigwedge_{g \in G} g],\ A,\ scheduled := m,\ coordState := l',$$

i.e., the clock constraints of G become an assumption, the clock assignments of A are executed, m is marked for scheduling, and *coordState* is set to the target location of e.

If $e \in E$ is a *sequential or unordered edge*, then the domain of *coordState* is extended by a new intermediate location l_e, representing the ongoing execution of e. The location l_e is considered to have only one outgoing edge, to the target location of e, with the same guards, clock assignments and component notations (including *seq* and *unord* notations) as e. The intermediate locations ensure that the execution of sequential and unordered edges cannot be interrupted.

If e is a *sequential edge* annotated by $seq\{m_{s1}, m_{s2}, \ldots, m_{sn}\}$, then a new control variable seq_e is created, representing the number of executed components, and e is mapped to

$$[\underset{g\in G}{\wedge} g],\ coordState := l_e,\ seq_e := seq_e + 1,$$
$$if(seq_e == 1)\ then\ \{scheduled := m_{s1}\},$$
$$if(seq_e == 2)\ then\ \{scheduled := m_{s2}\},$$
$$\ldots,\ if(seq_e == n)\ then\ \{A,\ scheduled := m_{sn},\ coordState := l',\ seq_e := 0\}$$

where the *else* branches of conditional operations were omitted for simplification. The variable seq_e determines the component that should be executed next in the given sequence. When the last component is scheduled, the clock assignments are executed and *coordState* is set to the target location.

If e is an *unordered edge* annotated by $unord\{m_{u1}, m_{u2}, \ldots, m_{un}\}$, then a new control variable $unord_e$ is created (representing the component that should be scheduled next), as well as n Boolean control variables $unord_{e1}, unord_{e2}, \ldots, unord_{en}$ (representing whether $m_{u1}, m_{u2}, \ldots, m_{un}$ has already been executed). The unordered edge e is then mapped to

$$[\underset{g\in G}{\wedge} g],\ coordState := l_e,\ havoc(unord_e),\ \{choose_1\}\ or\ \ldots\ or\ \{choose_n\},$$
$$if(\underset{1\le i\le n}{\wedge} unord_{ei})\ then$$
$$\{A,\ coordState := l',\ unord_{e1} := false,\ \ldots,\ unord_{en} := false\},$$

where $choose_i$ schedules the not yet executed m_{ui} for each $1 \le i \le n$:

$$[unord_e == m_{ui}\ \wedge\ \neg unord_{ei}],\ unord_{ei} := true,\ scheduled := m_{ui}.$$

Since some branch of the nondeterministic $\{choose_1\}$ *or* ... *or* $\{choose_n\}$ operation must be executed and all branches are constrained by assumptions of the form $\neg unord_{ei}$, the only feasible assignments at $havoc(unord_e)$ are those that represent a component that has not yet been executed. So, the component chosen for scheduling is certainly one of the eligible components. Lastly, if all components have already been scheduled, then the clock assignments are executed, and *coordState* is set to the target location.

The second part of the *env* operation set is the execution of the *env* set of one of the components, determined by *scheduled*. More precisely, it is a sequence

of a *delay* operation and conditional operations of the form $if(scheduled == m)\ then\ \{env_m\}$ for each $m \in \mathcal{M}$.

The internal transition set *tran* is constructed similarly. It starts with a *delay* operation, followed by conditional operations of the form $if(scheduled == m)\ then\ \{tran_m\}$ for each $m \in \mathcal{M}$. Therefore, the variable *scheduled* determines both the next environment step and the next internal step of the system.

4.2. Coordination of the SbW system

The coordination automaton of the motivating example is presented in Figure 2. The $q_0 \to q_3$ edge represents that, at first, the sensor components run and process their inputs. The sensor components must be executed in a one second long execution window, representing the possible deviation between their clocks. Since they are deployed separately and cannot influence each other's outputs, this behavior is modeled with the unordered execution. After all the sensors produced outputs, either the PrimaryRWA runs first and the SecondaryRWA runs second ($q_3 \to q_1$ and $q_1 \to q_0$ edges), or vice versa ($q_3 \to q_2$ and $q_2 \to q_0$ edges). This ordering can influence the cross-checking between the RWAs and possibly the MSP.

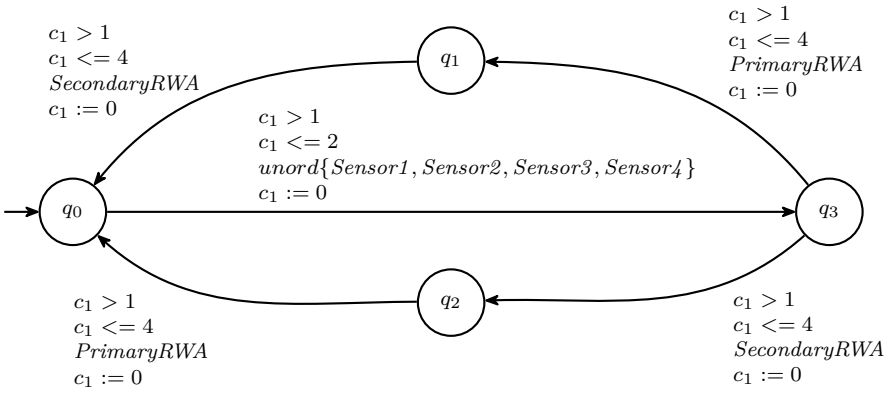

Figure 2. Coordination Automaton of the SbW System.

The TXSTS models representing the components of the system are connected in a single TXSTS model that already implements the coordination described by the given coordination automaton.

In locations q_1 and q_2 of the coordination automaton, there is only one possible execution order; in these cases, the mapping to TXSTS is straightforward, as shown in Listing 1 for location q_1.

In location q_3 of the coordination automaton there are multiple edges to other locations, referencing single TXSTS models. The mapping of this case to TXSTS is done using a nondeterministic choice for the possible outcomes, as shown in Listing 2.

```
assume coordState == q1;
assume c1 > 1 && c1 <= 4;
c1 := 0;
scheduled := SecondaryRWA;
coordState := q0;
```

Listing 1. TXSTS representation of deterministic scheduling of components in location q_1 of the coordination automaton

```
assume coordState == q3;
choice {
    assume c1 > 1 && c1 <= 4;
    c1 := 0;
    scheduled := PrimaryRWA;
    coordState := q1;
} or {
    assume c1 > 1 && c1 <= 4;
    c1 := 0;
    scheduled := SecondaryRWA;
    coordState := q2;
}
```

Listing 2. TXSTS representation of nondeterministically choosing the scheduled component in location q_3

```
assume coordState == q0; // q0q3 in case of line 18 in Listing 4.
assume c1 > 1 && c1 <= 2;
coordState := q0q3;
havoc unord_q0q3;
choice {
    assume unord_q0q3 == Sensor1 && !unord_q0q3_Sensor1;
    unord_q0q3_Sensor1 := true;
    scheduled := Sensor1;
} or {
    assume unord_q0q3 == Sensor2 && !unord_q0q3_Sensor2;
    unord_q0q3_Sensor2 := true;
    scheduled := Sensor2;
} or {
    ... // Sensor3
} or {
    ... // Sensor4
}
if (unord_q0q3_Sensor1 && unord_q0q3_Sensor2 && unord_q0q3_Sensor3 && unord_q0q3_Sensor4) {
    c1 := 0;
    coordState := q3;
    unord_q0q3_Sensor1 := false;
    unord_q0q3_Sensor2 := false;
    ... // Sensor3, Sensor4
}
```

Listing 3. TXSTS representation of unordered scheduling of sensors in location q_0 of the coordination automaton

For the unordered execution starting from location q_0 of the coordination automaton we introduce the variable `unord_q0q3` to represent the next scheduled component, which can take `Sensor1`, `Sensor2`, `Sensor3` or `Sensor4` as its value. We also introduce Boolean variables `unord_q0q3_Sensor1`, `unord_q0q3_Sensor2`, `unord_q0q3_Sensor3` and `unord_q0q3_Sensor4` that represent whether the given component was already scheduled. The scheduled component is chosen by a nondeterministic assignment to `unord_q0q3`, but considering only the components that were not yet scheduled, based on the newly introduced Boolean variables. The coordination automaton completes the transition to q_3 only when all referenced components were already scheduled. The TXSTS mapping of this unordered exe-

cution can be seen in Listing 3. The mapping of the intermediate location q_0q_3 is the same, except for the first line, where q_0 should change to q_0q_3.

The main structure of the resulting TXSTS model can be seen in Listing 4. The above-described mapping of the coordination automaton to the TXSTS formalism is embedded at the beginning of the single operation contained by the *env* operation set, followed by the operations of the individual components. If the *env* operation set of some component contains multiple operations (i.e., $env_{comp} = \{env_1, env_2, \ldots, env_n\}$), then these operations are wrapped in a nondeterministic choice $\{env_1\}$ *or* $\{env_2\}$ *or* ... *or* $\{env_n\}$, to represent them as a single operation without changing the possible behaviours of the model. The *tran* operation set is constructed analogously to the second part of the *env* operation set.

```
ctrl var coordState : enum{q0, q1, q2, q3, q0q3} = q0
ctrl var scheduled : enum{Sensor1, Sensor2, Sensor3,  Sensor4, PrimaryRWA, SecondaryRWA}
ctrl var unord_q0q3 : enum{Sensor1, Sensor2, Sensor3, Sensor4}
ctrl var unord_q0q3_Sensor1, unord_q0q3_Sensor2, unord_q0q3_Sensor3,
        unord_q0q3_Sensor4 : boolean = false
...
init {
    <init of Sensor1>
    <init of Sensor2>
    ... // Sensor3, Sensor4, PrimaryRWA, SecondaryRWA
}
env {
    delay;
    choice { assume coordState == q0; ... } // see Listing 3.
        or { assume coordState == q1; ... } // see Listing 1.
        or { assume coordState == q2; ... } // analogous with q1
        or { assume coordState == q3; ... } // see Listing 2.
        or { assume coordState == q0q3; ... } // see Listing 3.
    delay;
    if (scheduled == Sensor1) { <env of Sensor1> }
    if (scheduled == Sensor2) { <env of Sensor2> }
    ... // Sensor3, Sensor4, PrimaryRWA, SecondaryRWA
}
tran {
    delay;
    if (scheduled == Sensor1) { <tran of Sensor1> }
    if (scheduled == Sensor2) { <tran of Sensor2> }
    ... // Sensor3, Sensor4, PrimaryRWA, SecondaryRWA
}
```

Listing 4. Structure of a TXSTS model that schedules multiple TXSTS components based on a coordination automaton

5. Conclusion and future work

In this paper, we presented the coordination automata formalism as an extension of the timed automata formalism to model the possible interactions of the distributed subsystems. The presented coordination automata formalism is flexible and configurable: the coordinated subsystems can be modeled using arbitrary formal or engineering modeling languages.

As a continuation of our work, we investigate the possibilities of extending the coordination automata with more complex parallelism, to define coordination with overlapping component executions.

References

[1] K. Altisen, G. Gössler, J. Sifakis: *Scheduler modeling based on the controller synthesis paradigm*, Real-Time Systems 23 (2002), pp. 55–84, doi: 10.1023/A:1015346419267.

[2] R. Alur: *Timed automata*, in: Computer Aided Verification: 11th International Conference, CAV'99 Trento, Italy, July 6–10, 1999 Proceedings 11, Springer, 1999, pp. 8–22, doi: 10.1007/3-540-48683-6_3.

[3] C. Baier, J. Katoen: *Principles of model checking*, MIT Press, 2008.

[4] D. Cziborová: *Abstraction-based model checking for real-time software-intensive system models*, Scientific Students' Association Report, Budapest University of Technology and Economics, 2023.

[5] D. Cziborová, R. Szabó: *Modeling of Time-Dependent Behavior in Fault-Tolerant Systems*, in: 31th PhD Minisymposium of the Department of Measurement and Information Systems, Budapest University of Technology and Economics, 2024, doi: 10.3311/MINISY2024-011.

[6] B. Graics, M. Mondok, V. Molnár, I. Majzik: *Model-Based Testing of Asynchronously Communicating Distributed Controllers*, in: Formal Aspects of Component Software, ed. by J. Cámara, S.-S. Jongmans, 2024, pp. 23–44, isbn: 978-3-031-52183-6, doi: 10.1007/978-3-031-52183-6_2.

[7] M. Lohstroh, C. Menard, S. Bateni, E. A. Lee: *Toward a lingua franca for deterministic concurrent systems*, ACM Tran. Embedded Comput. Syst. 20.4 (2021), pp. 1–27, doi: 10.1145/3448128.

[8] V. Molnár, B. Graics, A. Vörös, I. Majzik, D. Varró: *The Gamma Statechart Composition Framework: design, verification and code generation for component-based reactive systems*, in: ICSE '18, ACM, 2018, pp. 113–116, doi: 10.1145/3183440.3183489.

[9] C. Norstrom, A. Wall, W. Yi: *Timed automata as task models for event-driven systems*, in: Proceedings Sixth International Conf. on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No. PR00306), IEEE, 1999, pp. 182–189, doi: 10.1109/RTCSA.1999.811218.

[10] R. Szabó, D. Cziborová, A. Vörös: *Towards Configurable Coordination for Distributed Reactive Systems*, in: 32th PhD Minisymposium of the Dept. of AI and Systems Engineering, Budapest University of Technology and Economics, 2025, doi: 10.3311/MINISY2025-009.