*Proceedings of the 2023 Winter Simulation Conference*
*C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.*

# APPROXIMATE DISCRETE-EVENT METHOD FOR SUPERVISORY CONTROL

Maaz Jamal
Gabriel Wainer

Systems and Computer Engineering
Carleton University
1125 Colonel By Dr
Ottawa, ON K1S 5B6, CANADA

2023 Winter Simulation Conference (WSC) | 979-8-3503-6966-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/WSC60868.2023.10408471

## ABSTRACT

Supervisory systems are used to and act when certain events are detected. Studying supervisory using formal Discrete Event Modelling & Simulation allows analyzing an application and then using the model to build the controllers. Supervisors can lead to a state space explosion if the model size increases, thus, reducing the state space complexity can expand the practicality of the model. We present a method based on Discrete Event System Specifications using an approximate method that reduces the state space complexity. The plant models and synthesized controllers can then be deployed on embedded hardware providing model continuity. We discuss the method and present a case study of a supervisory system.

## 1 INTRODUCTION

Digital Control Theory (Fadali and Visioli 2013) translates continuous to digital control systems with a discretization process introduced by a digital clock. Controlling such systems involves sensors, actuators, and a controller that keeps the system between a range of acceptable values. This controller, which can be derived by analyzing models of the system, is usually modeled by differential equations. To guarantee the correct functioning of the devices, supervisory systems are used to evaluate the control system and act when certain events are detected. Since the 1960s, Finite State Machines (FSM) have been used to represent these Discrete Event Dynamic Systems (DEDS) and make them more manageable (Wonham et al, 2018). Supervisory systems' actions can be described by discrete events and the transitions between those events.

These Discrete-Event Supervisory systems (DES) can be studied using a formal model of a physical system and its specifications. The events in the plant are described as alphabets that are either controllable or uncontrollable. The role of the DES is to disable the appropriate events and preventing actuators from responding to those events. To do this, we can use *supervisor synthesis*, a method that involves creating a Global State Space (GSS) that contains every possible path the model can follow for given starting states and conditions, and subsequently extracting the supervisor states from this GSS by removing any state and transition that will cause the model to behave against the control specifications defined by the modeler.

DES were first defined for monolithic models, which caused state space explosion during the GSS definition (Gohari 2000). This limits the feasibility of the method to small use cases. Instead, modular and hierarchical architectures can reduce this state space complexity: modular approaches allow splitting the model into smaller pieces and thus reducing the state space for each module (Wonham 1988). Similarly, hierarchical models use a minimalistic supervisor at the top that controls significant events on the lower level of the plant. The supervisor is then synthesized on the high-level model which imposes high-level specifications. Despite this, reducing the complexity of the state-space remains a challenge.

Among the different methods used to model supervisory control there are different formalisms like Petri Nets as well as computation techniques like binary decision trees (Wonham 2018). In this research we explore the use of DEVS (Zeigler 2000), as its modular and hierarchical nature is well adapted to the problems just discussed. In (Zeigler et al. 1995), a method was presented to synthesize DES using the

Cartesian product. The resulting GSS contains every state and transition, ignoring the hierarchical structure of the DEVS model regardless of whether those states are feasible or not. This creates a GSS that for larger models may be infeasible. In this research we introduce a new method that uses that structural information to reduce the number of transitions in the GSS. We use the coupling information to decide which states and transitions are feasible and which can be safely ignored, thus producing a smaller GSS, from which to extract a supervisory controller. It is an approximate method as there might be safe sequences of events that are ignored. We focus on the state-space explosion problem for DEVS models, and the GSS we generate only considers logical time (timing analysis on the GSS is left for future research). The method is applicable to discrete-event supervisory control systems (continuous systems are excluded). We introduce the method and present a case study showing how to model a system, extract a supervisory controller and execute the controller both in simulation and on embedded hardware targets.

## 2 BACKGROUND / RELATED WORK

Research on DES includes using different modeling formalisms, solving the problem of GSS explosion, observability, reachability, and permissibility. A large state-space can make the process of extracting the controller and performing analyses computationally infeasible. (Ramadge 1982) introduced the idea of deriving supervisory controllers from formally defined DES. The method disables discrete event transitions to enforce control objectives. Modular approaches can isolate subsets of the model with local interactions, reducing overheads and state-space complexity. However, coordination between subsets can lead to race conditions or deadlocks. Solving these problems using global coordination becomes computationally expensive (Teixeira 2011). Hierarchical approaches, instead, use a top-level DES with a limited view of the lower levels. This could cause inconsistent information use or hierarchical inconsistencies (Zhong 1990).

(Brandin 1994) extended the use of supervisory control for temporal models with timed automata, including forcible events. This is akin to passivation of DEVS models, which will be discussed later. More recently (Alves 2017) used a method to generate an untimed model to investigate delays and loss of observation between plant and supervisor and conditions for the existence of supervisors.

These formalisms treat the values of data variables as distinct states. This causes a state space explosion as the values of the variable need to be enumerated for analysis. For such cases (Le Gall et al., 2005) proposed the use of Symbolic Transition Systems where the set of uncontrollable states is abstracted, allowing the model to become decidable and a supervisor to be synthesized. (Forschelen 2012) showed that formal methods for designing controllers are suitable in an industrial setting and are more scalable, maintainable, and dependable as formal methods provide a better structural approach in software design, have fewer ambiguities and are more dependable as the correctness of the model is embedded in the design. (Yamalidou 1996) used Petri Net (PN) place invariants (regions where tokens remain constant for all markings) to compute the DES. Fluid PNs (Silva 2016), where markings can take real negative values, and Hybrid PN (Giua and Silva 2018), which can model a larger class of systems, help to solve the state space explosion problem. However, they are difficult to analyze formally and there is no general methodology to design controllers. For timed PN, the method of discretization of the clock produces more complex models.

DEVS (Zeigler 2000) is better suited to solve these problems, including hybrid systems (Grossman 1993). DEVS is a modular and hierarchical modeling formalism for systems whose state transitions are driven by discrete events. DEVS atomic models define component behavior using states, transitions, and events that trigger the transitions. Coupled models define the hierarchical structure of the model, defining subcomponents and couplings between them. (Maione 2004) defined a genetic event supervisor that can adapt over time to fit the conditions of the system. (Marcosig 2017) showed the use of DEVS to verify, validate and design event-based controllers for hybrid systems through iterative development and model continuity. (Zeigler et al. 1995) introduced the use of DEVS for supervisory control of hybrid systems but research is needed in new methods to gauge the observability, permissiveness, deadlock analysis, reachability and other properties for DES. DEVS allows model continuity across platforms, which is important to ensure the specifications of the model remain intact and therefore ensure models are formally equivalent (Niyonkuru 2022). We can use DEVS Modelling, Simulation, and Real-Time (RT) control on embedded platforms, ensuring model continuity. We achieve this goal using Cadmium (Belolli 2019), a DEVS software stack written as a C++17 header-only library, and RT-Cadmium.

## 3 A METHOD FOR OBTAINING DISCRETE-EVENT SUPERVISORS

We introduce a methodology for obtaining DES presented in Figure 1, an extension of (Zeigler 1995).

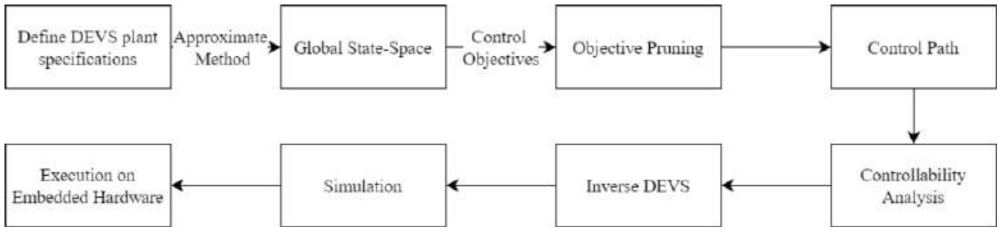

Figure 1: Modified methodology for obtaining the Discrete-Event Supervisor.

The first step is to define a DEVS model of the plant. Then, we use our approximate method (discussed in the next section) and obtain a GSS of reduced complexity that keeps the structure of the coupled plant model. This GSS is a DEVS model represented as a weighted directed graph with information about how the plant evolves from a state by the influence of external inputs and internal transitions. The states form the vertices of the graph, and the transitions make up the edges. Supervisory control objectives can be specified and then applied to the GSS, defining states and transitions that are unwanted. The controller can only disable events and therefore the control objectives include unwanted states and transitions that need to be removed (which represent states and transitions that violate the control goals). In addition, the control objective also specifies if a control path between two states exists. These states define the start and endpoints of the control loop the plant is expected to take. If there is a path between two states, a control loop exists in our reduced GSS. We then apply controllability analysis to show any transitions with the potential to move the controller away from the desired control path and onto an uncontrollable path (weak transitions). If the reduced GSS does not contain weak transitions, we apply Inverse DEVS transformation and obtain a DES (Discrete-Event Supervisor) that satisfies the control objectives. The DES and plant model are coupled together and tested through DEVS simulation, and we can make any necessary changes to the plant or controller to fix any issues found. The models can then be ported over to hardware and executed in real-time, maintaining model continuity between simulation and deployment.

### 1.1 Calculating the Cartesian Product

We calculate the GSS of the plant model using the Cartesian product, which takes the directed graph of each atomic model and then computing the Cartesian product with the previous product. Consider the case of two atomic models G and H in Figure 2(a). $V(G) = \{u_1, u_2\ldots, u_n\}$ and $V(H) = \{v_1, v_2 ,\ldots, v_n\}$ are the sets of vertices of the two graphs. E(G) and E(H) represent the edges of the model. To compute the product, we form a set of ordered pairs of V(G) and V(H), which forms the combined states of the global state-space. They are built as follows: for each vertex $u_i$ form a pair with vertex $v_j$ where j is the index of the vertices of V(H). If we take vertex $u_i$ in Figure 2(a), we form the ordered pairs $(u_i,v_j)$, $(u_i,v_{j+1})$,$(u_i,v_{j+2})$, shown in the first line of Figure 2(b). We repeat the procedure for $u_{i+1}$ and $u_{i+2}$ and get the set of 9 ordered pairs shown in the figure (these are ordered pairs: the order of appearance is important; $(u_i, v_j) \neq (v_j, u_i)$ in the product).

Next, we need to find if two combined states in the global state space are adjacent to each other; in other words, if they can be connected by an edge. Two vertices are adjacent if for an edge $(u_i,v_j)$,$(u_k,v_l)$: i=k and $v_j$,$v_l$ is an edge of H; or j=l and $u_i$, $u_k$ is an edge of G. This means that two combined states in GSS are adjacent if one is fixed in both the combined states, and the unfixed state has an edge in its original graph with the corresponding unfixed state in the combined state (Figure 3(c) and 3(d)). The direction and weights of the edges are preserved as the ordered pairs differ by only one state coming from one atomic model.

Therefore, the original weights still represent the transition condition (full lines for internal transitions and dotted lines for external). Two states can have multiple edges if the weights are different; in this case, the combined states also have multiple edges between them.

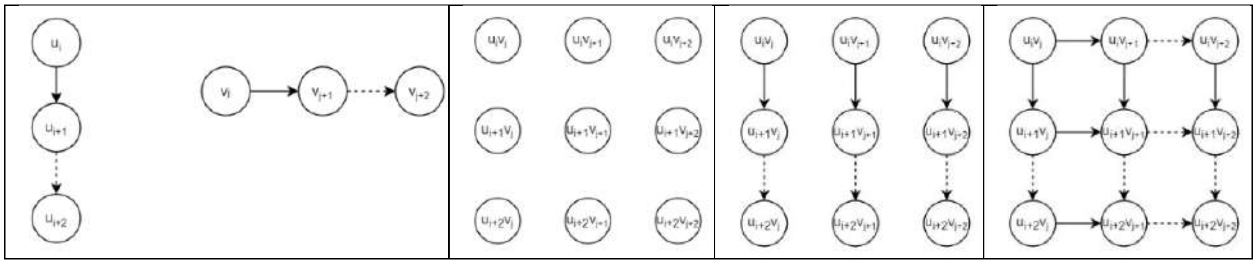

Figure 2: (a) Atomic models G and H; (b) Ordered pair sets of two graphs V(G), V(H); (c) Adjacent vertices due to edges of G; (d) adding vertices due to edges of H.

The edges in 2(c) are due to the edges of G. Notice V(H) does not change between the connected nodes and only V(G) changes. If two vertices are adjacent, we add the edge between the two states, this transition will cause the system to move to the other combined state. The ordered pairs imply that edges (and therefore transitions) can only exist between nodes that have only one state change in their combined states. As the Cartesian product is not commutative, we choose the order of operations based on the *select* function of the coupled model. We can see that the Cartesian product does not use the structural information available in the DEVS coupled model of the plant. This leads to the addition of extra transitions that will never occur. The method presented next uses this information to attempt to reduce the number of transitions.

## 1.2 Approximate Method to Generate the GSS

We use the structural information of the coupled model to generate the GSS. The couplings define the model's structure, which restricts the input events the model can react to, and the output events the model will generate. Feasible transitions for a coupled model are determined by EIC, IC, and EOC sets. If there is no coupling to an input or output port of an atomic model, events cannot input/output the model and certain transitions can be ignored. Further, for the output function $\lambda$, we need the EIC or IC to be coupled with the output ports of the atomic models of interest. The output ports transmit output events; thus, they do not affect the model. Figure 3 shows two models whose structure includes components in which the output ports are not in the IC or EOC sets.

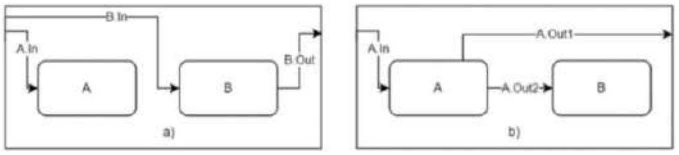

Figure 3: Two coupled models: a) no IC between A and B; b) no EOC with ports of B.

Figure 3(a) shows a model with no IC between two atomic models. Outputs of A cannot reach B (so, we can ignore them in the GSS). Model B can only be influenced by the EIC through *B.In* and only $\lambda$ of B can generate outputs. Therefore, we include it by adding the associated $\delta_{int}$ to the GSS. This is done by finding the two combined states (as explained earlier) and after, $\delta_{int}$ is executed. The coupled model generates information through the $\lambda$ of B, the EOC with port *B.Out*. In Figure 3(b), the $\lambda$ associated with port *A.Out2* influences model B. However, model B has no EOC and only the $\lambda$ associated with *A.Out1* can provide the information. Therefore, the internal transitions of B can be ignored. We utilize this information to only add the transitions that affect the model to the GSS. These transitions can then be added between the combined states of the GSS that have the state change defined in the transition. By doing so we remove impossible transitions from the system and reduce the global state space to a more manageable size.

This is done as follows. For a coupled DEVS model CM = < X , Y, D, {$M_i$}, EIC, IC, EOC, *select* >

```
Order the set D according to the Select function
for each i in D, choose the corresponding instance Mi:
       if i in EIC.component:
           for Mi.external_transition.port in Mi.port:
               if Mi.external_transition.input_type == EIC.port.InputType
                 Add external_transition to global state space
       if i in IC.first_component:       {
           for Mi.internal_transition.port in IC.out_port:
                 Add δint to global state space
       Mi2 = IC.second_component
       for Mi2. δext.port in IC.in_port:
             Add δext to global state space
    }
    if i in EOC.component:
        for Mi. δint.port in EOC.outport:
             Add δint to global state space
```

The algorithm first selects a component from the coupled model using each submodel i from set D and its couplings (the algorithm shows the top model, but this can be applied recursively for lower levels of the hierarchy). For each component i, its associated atomic model is used to obtain the transitions and their weights. The couplings are used to decide which transitions to consider and which to ignore.

The algorithm first searches the EIC and checks if it contains any entry with the component name (remember that not every component may be in the EIC set; this includes components of the same atomic model type where all the components may not be connected to the input port of the coupled model). If it is found, we access the external transitions and search for the port and input type that matches the given EIC entry. We then add this $\delta_{ext}$ to the GSS between the combined states. The $\delta_{int}$ that may be triggered after this external transition is not considered. As discussed, for a $\lambda$ associated with a $\delta_{int}$ to influence the model it must be present in either the IC or EOC.

Figure 4 shows a coupled model with atomic models; A and B (states are rectangles). INT represents internal and EXT external transitions; the names after "/" refer to the ports associated with outputs or inputs.

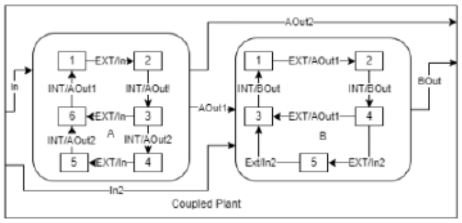

Figure 4: Coupled model with atomic models A (left) and B (right); states, transitions and couplings.

There are two EICs: port In coupled with atomic A and In2 coupled with atomic B. For model A, the $\delta_{ext}$ affected by the input coupling are 1→2, 3→6 and 4→5. These are thus added in the GSS using combined states that differ by these states. For example, the transition 1→2 is added as a link from the combined states that have the A set as 1, to the combined states that have the A set as 2. Similarly, the $\delta_{ext}$ of B affected by the EIC are 4→5 and 5→3 and these are added to the GSS. The $\delta_{int}$ of atomic A or B that are triggered after $\delta_{ext}$ are ignored and are considered in later steps when the IC or EOC are analyzed. Finally, the $\delta_{ext}$ of B that are triggered by input from port AOut are ignored as these are due to the IC between A and B.

Next, we search the list of Internal Couplings. An IC can be used by two transitions (internal and external) across two components. The first is a $\lambda/\delta_{int}$ that generates an output from the first component in the IC list. The output is sent to the second component as an input. This input then triggers $\delta_{ext}$ in the second component. We use a two-step process. First, we match a component *d* with the first component from the IC list. If they match, the internal transitions are filtered, and their output functions are matched with the output port in the second component of the IC. The $\delta_{int}$ is added between the two combined states of the model with the state change specified by $\delta_{int}$ between them. Then, we take the second component for this

IC and find $\delta_{ext}$ in the second component with an input port matching the output of the first component. This transition is added between the combined states that contain the state change, with an added filtering step. We realize that $\delta_{int}$ moves the system from one combined state set $x$ to a combined state set $y$. The $\delta_{ext}$ can only occur from the combined state set $y$ as the GSS must be in that state because, after the time advance, $\delta_{int}$ and $\lambda$ occur simultaneously. The GSS at that time must then be in state $y$ when the $\delta_{ext}$ is executed in the second component. Further, $\delta_{ext}$ also occurs instantaneously and the GSS cannot move away from state set $y$ as no other events can occur at this point. This filtering step ensures that the transitions due to the IC only exist between combined states that can exist at that instant.

The case of the IC can be better explained using the coupled model in Figure 4. The $\delta_{int}$ of A and the associated output functions involved with the IC coupling between the A and B are transitions $2 \rightarrow 3$ and $6 \rightarrow 1$ from A. These are added to the GSS, and B is saved as the second component involved in the IC. For B, the external transitions are matched with the relevant input port; in this case, transitions $1 \rightarrow 2$ and $4 \rightarrow 3$ match AOut, which is added because of the IC. The external transitions due to ports In and In2 were considered in the previous step. The $\delta_{int}$ and the $\lambda$ associated with port AOut2 and BOut are next.

Finally, the EOC list is like the EIC, except that we match the $\lambda$ associated with the $\delta_{int}$ of the component d to the output port specified in the EOC. Referring once again to the Figure 4, the EOC only considers the $\lambda$ associated with the coupled ports AOut2 and BOut and the associated $\delta_{int}$. This means we only add transitions $3 \rightarrow 4$ and $5 \rightarrow 6$ from A and $2 \rightarrow 4$ and $3 \rightarrow 1$ from B. All the other $\delta_{int}$ are ignored.

## 1.3 Obtaining a Reduced GSS

Given a GSS represented as a directed graph G = (V, E), the control objectives can be applied to remove unwanted transitions and states that can move the model onto an uncontrollable path. The controller that we then obtain can disable unwanted transitions. This is done by not having state transitions that would lead to unwanted transitions or illegal states, or not having output functions that would produce outputs to the actuators causing unwanted transitions. Three general rules are applied to the state graph to obtain a reduced GSS. They ensure that the controller that is obtained at a later step from this reduced GSS will comply with our specifications. The three rules are a) no-state, b) no transition, and c) the path exists.

The *no-state* rule removes unwanted states that violate objectives or states whose transitions can move the model away from the control path. This rule can also be used to remove nonsensical states that are in the GSS but whose necessary conditions for being reachable are not feasible. We search the vertices V(G) and find the combined states that match this rule. These states are then removed. The transitions from these states are also removed and only transitions to these states are kept for controllability analysis later.

The *no transition* rule searches the edges and removes transitions between states specified in the rule. There are two reasons for this. The first is to remove unwanted transitions to states that are illegal or would cause the model to stray away from controllable states. The second is to prevent the controller from switching between legal states and creating multiple paths that would make the model uncontrollable. Both the no transition and no state rules can be applied to a subset of a combined state.

Finally, the *path exists* rule establishes that there is a closed path between any two combined states. Since it is a directed graph, the closed loop can have different forward and reverse paths to the two states specified and must be searched in both directions. The uniqueness of the path is not determined and there can be other paths between the starting and ending states. If there is a strongly controllable loop between starting and ending states the controller should be able to keep up with the state of the plant model.

The specifications need to be applied in a specific order as they cannot be applied in parallel. Each rule application changes the connections and paths of the graph, therefore making a parallel computation impossible. We apply the control objectives in such a manner to use the minimum number of operations to obtain the reduced GSS: 1. No State; 2. No Transition; 3. Path exists. Applying the rules in this order results in minimum operations and a reduced GSS in which the determination of path existence from the path-exist rule holds. Applying the no state rule first removes all unwanted states from the system. We then apply the no transition rule and remove all unwanted transitions. These rules change the structure of the GSS, thus, if the path exists rule is applied before them, the result can be invalidated by any subsequent rule application.

## 1.4 Controllability Analysis and Inverse DEVS Transform

The application of the control objectives leaves a reduced GSS that can be used to obtain a supervisory controller that meets the specifications. Before a supervisory controller can be extracted from this graph, controllability analysis needs to ensure the obtained DES keeps the model on the desired path. The objective is to ensure the DES and the reduced GSS only have very-strongly or strongly controllable transitions.

At a given state, if $\delta_{ext}$ executes and the model moves to an illegal state, the transition is weakly controlled and the control path becomes weakly controllable. Since this is an external transition, its execution cannot be prevented. If instead $\delta_{ext}$ moves to an allowed state and the time advance is finite, this transition is strongly controllable: $\delta_{ext}$ may be followed by $\delta_{int}$ and the finite time advance can allow new external inputs to be received, changing the model to a state outside the control path. For $\delta_{int}$, if the time advance is finite and transitions to an allowed state, the transition is strongly controllable ($\delta_{int}$ cannot change states). However, the next $\delta_{int}$ may switch the model to an illegal state. If the transition is to an illegal state, the transition and the path are weakly controllable. The $\delta_{int}$ will take the DES off the control path unless $\delta_{ext}$ moves it back onto the path. Similarly, if the time advance is infinite, $\delta_{int}$ is strongly controllable as the model and DES cannot stray away from the path due to internal transitions. The component passivates after the transition and therefore no other $\delta_{int}$ can shift the component away from the current state. In summary for a strongly controllable path, transitions must only be to allowed states, and for internal transitions, the time advance should passivate the model after the execution of the transition to an allowed state.

Ideally, our path should only contain very strongly controllable transitions. However, strongly controllable paths can still be utilized to make DESs: there must be time before a transition and there must be external transitions on the path that can force the model on the desired control path. In these conditions, a strongly controllable path can be kept on the control path by external transitions of the model. Otherwise strongly controlled transitions run the risk of straying from the path.

The Inverse DEVS transformation is performed on the reduced GSS to obtain a candidate DES. The information about the transition type is stored in the weights of the edges. We can change the type of the edge by modifying the weights and switch internal transitions and external transitions. The I/O ports are switched as well: input ports become output ports for $\lambda$, and output ports become input ports. If the Inverse DEVS of a model, our DES, and the model are coupled together, the output types of the model are the same as the input types of the DES and vice versa; hence, if the two models are coupled and start in the same state, the model and DES stay coordinated.

Once we obtain a DES, we can simulate the DES and the plant model coupled together to test for proper functioning. This is useful to identify any extra transitions and combined states that are not used during the normal operation of the plant-controller coupled model. These transitions do not alter the course of the operation and may be left as is; however, they take up storage resources and should be removed. These extra transitions arise primarily because the control objectives did not identify them as problematic and were not removed during the application of the control objectives.

At this point, it is important to identify missing transitions using the DES and the coupled model. We search for external transitions in the DES with infinite time. For these, we note which state has changed in the combined state (i.e., the component that caused the state change). This is done by comparing the position of the state in the combined state with the *Select* function. Once we identify this atomic model, we know the second component of the IC and from this, we can identify $\delta_{ext}$ that will be triggered in the second atomic model. This $\delta_{ext}$ can now be used to find the inverse $\delta_{int}$ in the DES. We find the $\delta_{int}$ in the DES that corresponds with the state change in $\delta_{ext}$. With the identification of both the transitions involved in the missing transition, $\delta_{ext}$ of the DES is altered such that the transition is to the state after $\delta_{int}$ has been executed.

The DES and the model can be simulated using the DEVS Cadmium tool to identify problems in the model. After testing through simulation, the models are downloaded onto an embedded platform using the RT-Cadmium tool. Since all our models are DEVS models, no changes need to be made. The simulation scenarios defined for the software simulation can now be executed on the embedded platform. This testing can more realistically test the DES in real environmental conditions and identify any further problems. If no problems are found, then we can be assured that the DES and the model work effectively.

## 4 CONTROL OF A ROOM WATER TEMPERATURE MODEL

We show how to apply our method in a case study for a supervisory system for room water temperature control. The water temperature in the room is controlled such that the occupant has access to warm water. For this reason, we require the water should be kept between *Cool* and *Hot* temperature ranges and not dip to *Cold* or rise to *VeryHot* and cause burns (although the control of the temperature heater could be done using traditional continuous control such PID, here we focus on supervisory actions in the system).The boiler affects the water temperature and cycles between *On* and *Off* states. When the room is occupied, the boiler should turn on and ensure the water is warm. For this reason, we also include an occupancy sensor.

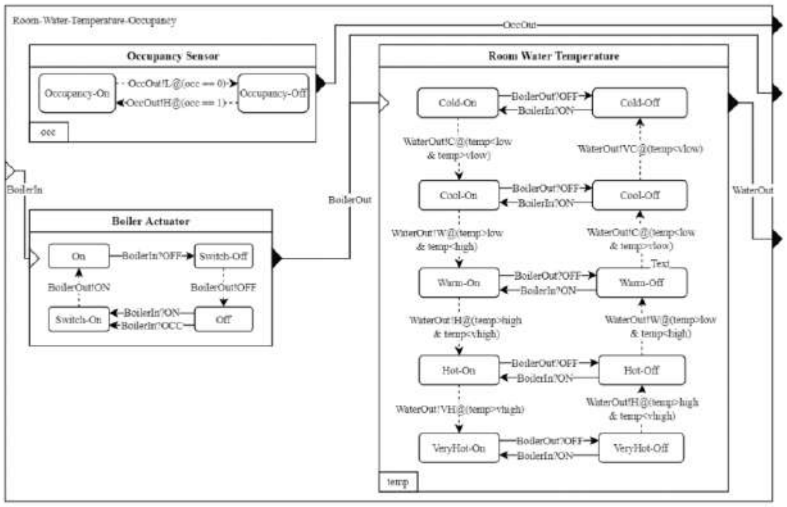

Figure 5: Boiler actuator, occupancy sensor, and the room water temperature model.

Figure 5 shows the top model and its components. The occupancy model checks a sensor, and based on its value $\delta_{int}$ changes the state back and forth between *'Occupancy-On'* and *Occupancy-Off*. The actuator model transition *Off → Switch-On* can be caused by two input events: ON, which triggers the transition due to water temperature rising above the threshold value, and OCC, caused by the room being occupied.

The room water temperature model states represent the water temperature ranges when the boiler is on (suffix *–On*) or off (suffix *–Off*). When the boiler is on, the temperature rises and the internal transitions go from *Cold-On* to *VeryHot-On*. Similarly, when the boiler is off, the water temperature falls, and $\delta_{int}$ moves from the *VeryHot-Off* to *Cold-Off*. The $\lambda$ outputs values for each of the water temperature ranges *Cold*, *Cool*, *Warm*, *Hot* and *VeryHot*. The $\delta_{int}$ checks the value of the sensor and changes the states based on this value. The $\delta_{ext}$ switches between the *-Off* and *-On* versions of the temperature states based on the input.

Once we have defined the model of the plant, we need to extract the GSS from the coupled model using our method. To compute the Cartesian product we use NetworkX, a Python package that provides tools to create graphs and perform operations on them (Hagberg et al 2008). We sort the graphs according to the *Select* function and compute the Cartesian product and note the number of states and transitions in the resulting GSS. We then compute the GSS for each model using the approximate method, by passing it through a python script that returns a file with the vertices and a list of associated edges that are transitioning from this combined state. We note the number of states and transitions for each of the models. In our case, the model needs to keep the water temperature between *Cool* and *Hot* ranges and turn the boiler *On* when there is an occupant in the room. This is defined as follows:

```
path,water.cool-off,boi.off,?,water.hot-on,boi.on,?
no_state,water.cold-on,?,?
```

```
no_state,water.cold-off,boi.on,?
no_state,water.cold-off,boi.switch_on,?
no_state,water.veryhot-off,?,?
no_state,water.veryhot-on,boi.switch_off,?
no_state,water.veryhot-on,boi.off,?
```

The path from *water.cool-off, boi.off* turns the heater *On* when the water temperature is *Cool* until the water is *Hot*, and the boiler is *On*. From this state *water.hot-on, boi.on* we can go back to *water.cool-off, boi.off* by turning the boiler *Off* and letting the water temperature return to *Cool*. The states *Cold-On* and *VeryHot-Off* are removed as they are illegal. The states *water.cold-off, boi.on, ?* and *water.cold-off, boi.switch_on* are removed to force the DES to switch the boiler *On* when the water is *Cool* as then the only transition to switch the boiler *On* will be from *Cool* states. For similar reasons, the states *water.veryhot-on, boi.switch_off, ?* and *water.veryhot-on, boi.off, ?* are removed to turn the boiler *Off* in the state *Hot*.

We apply the rules of controllability to the reduced GSS that we obtain after applying the control specifications. A portion of the controllability analysis for the room water temperature is as follows:

```
water.cool-off,boi.switch_on,occ.occupancy_on,
water.cold-off,boi.switch_on,occ.occupancy_on,
int,water_out,vlow,fin,

water.cold-off,boi.off,occ.occupancy_on,
water.cold-off,boi.switch_on,occ.occupancy_on,
ext,boiler_in,on,fin,
```

The transitions are weak transitions arranged as triplets and separated by an empty line. Each triplet represents one weak transition with the first line representing the starting states and the second line representing the resulting state. The third line represents the weights of the transition.

The first pair moves the state from 'Cool-off' to *Cold-Off*, which is an illegal state. This transition is highly unlikely to occur so long as the range of temperature represented by *Cool* is large enough and the boiler can turn on and start heating the water. The boiler is already in the 'Switch_on' state and therefore the boiler will start heating the water and the water will not reach the *Cold* temperature. The second triplet represents states in which the water temperature is *Cold*. These states are also unlikely to occur as the heater should be *On* and the water should start heating when the water temperature is in the *Cool* range. From this analysis, the control loop is unlikely to transition to those states and therefore we can apply inverse DEVS.

Figure 6 shows the DES obtained from the reduced GSS after applying the Inverse DEVS transformation and resolving the missing transitions. In this case, three missing transitions occur due to the IC between the ports *BoilerOut* of the actuator model and the room water temperature model. The first missing transition occurs when the DES is in the state *Cool-Off, Off, Occupancy-Off* and outputs *ON* to the actuator model and moves to *Cool-Off, Switch-On, Occupancy-Off*. The actuator model executes $\delta_{ext}$ and changes to *Switch-On*, followed by the execution of $\delta_{int}$, moving the actuator to *On* and $\lambda$ generating *On*. The actuator passivates and the room water temperature model moves to *Cool-On*. However, the DES is in *Cool-Off, On, Occupancy-Off* after it receives the input *On* from *BoilerOut* and is passivated. This transition is therefore changed, and we add *Cool-Off, On, Occupancy-Off* → *Cool-On, On, Occupancy-Off* and set the time advance of the room water temperature model. This is shown in Figure 6 as a separate external transition with no port or input event so that this modification is marked. In the implementation, there is only one $\delta_{ext}$ that changes *Cool-Off, Switch-On, Occupancy-Off* → *Cool-On, On, Occupancy-Off*. Similarly, the other two missing transitions (*Hot-On, Switch-Off, Occupancy-Off* → *Hot-Off, Off, Occupancy-Off, and Warm-Off, Switch-On, Occupancy-On* → *Warm-On, On, Occupancy-On*) are resolved.

The execution of the room water temperature, boiler actuator and occupancy models as well as the DES are demonstrated through a scenario where simulated water temperature rises when the boiler is *On* and falls when the boiler is *Off*. In addition, the occupancy sensor model switches between *Occupancy-On* and *-Off* at fixed intervals. The models start in states *Cool-Off*, *Off*, and *Occupancy-Off* for the room water temperature, boiler actuator, and occupancy sensor models, respectively. The DES starts in *Cool-Off, Off, Occupancy-Off*. When the simulation starts the water temperature starts to fall as the boiler is off. The DES outputs *ON* through *BoilerIn* and moves to *Cool-Off, Switch-On, Occupancy-Off*. This is received by

*BoilerIn* of the boiler actuator model causing it to turn *On* after $\delta_{ext}$ and $\delta_{int}$. The boiler actuator model outputs *ON* through *BoilerOut*. The room water temperature model receives this as an input and moves to *Cool-On*. The DES upon receiving this input moves to the state *Cool-On, On, Occupancy-Off*. We observe the water temperature rises due to the model being in one of the -*On* states.

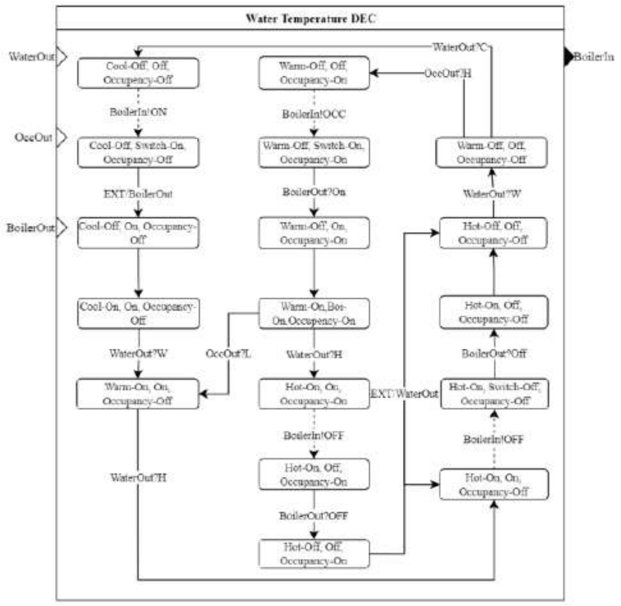
Water Temperature DEC
WaterOut
OccOut
BoilerOut
BoilerIn
Cool-Off, Off, Occupancy-Off
BoilerIn!ON
Cool-Off, Switch-On, Occupancy-Off
EXT/BoilerOut
Cool-Off, On, Occupancy-Off
Cool-On, On, Occupancy-Off
WaterOut?W
Warm-On, On, Occupancy-Off
WaterOut?H
Warm-Off, Off, Occupancy-On
BoilerIn!OCC
Warm-Off, Switch-On, Occupancy-On
BoilerOut?On
Warm-Off, On, Occupancy-On
Warm-On, Boi-On, Occupancy-On
OccOut?L
WaterOut?H
Hot-On, On, Occupancy-On
BoilerIn!OFF
Hot-On, Off, Occupancy-On
BoilerOut?OFF
Hot-Off, Off, Occupancy-On
EXT/WaterOut
WaterOut?C
OccOut?H
Warm-Off, Off, Occupancy-Off
WaterOut?W
Hot-Off, Off, Occupancy-Off
Hot-On, Off, Occupancy-Off
BoilerOut?Off
Hot-On, Switch-Off, Occupancy-Off
BoilerIn!OFF
Hot-On, On, Occupancy-Off

Figure 6: DES for the room water temperature model.

After some time, the occupancy sensor model changes its state to *Occupancy-On* and outputs *H* through the *OccOut* port. The DES receives this and moves to *Cool-On, On, Occupancy-On* and does not output anything as the boiler is already *On*. The water temperature continues to rise and the room temperature model outputs the corresponding event for each of the temperature ranges as the threshold is crossed. The DES changes to match with what the water temperature model outputs and is observed to go through the states *Cool-On, On, Occupancy-Off → Warm-On, On, Occupancy-Off → Hot-On, On, Occupancy-Off*. When the DES reaches the state *Hot-On, On, Occupancy-Off* it signals the boiler actuator to turn off. The boiler is turned *Off*, and the water temperature begins to fall. The DES moves to *Hot-Off, Off, Occupancy-Off*. The DES is observed to move from *Hot-Off, Off, Occupancy-On → Warm-Off, Off, Occupancy-On*. This time the DES generates an output *OCC* through *BoilerIn*. This causes the boiler actuator to turn *On* and the water level begins to rise again until the water temperature reaches *Hot* and the boiler is turned *Off*.

From this simulation, we observe the DES keeps the water temperature between *Cool* and *Hot* temperature ranges and does not let the water temperature reach *VeryHot* or *Cold* thresholds. The DES can keep coordinated with the states of the models. When the occupancy sensor indicates the room is occupied and the boiler is off, the DES signals the boiler to turn on. The boiler is kept on until the water temperature reaches *Hot*. This behavior is per the control objectives set.

Figure 7 shows the hardware setup to test the room and water temperature model. A potentiometer labeled *Setpoint* controls the temperature. An amber LED is turned on by the boiler actuator model when it is in the *On* state. A red LED to the right is turned on by the AC actuator model when it is *On*. The LCD shows the current water and room temperatures. Figure 7 shows various stages in the model execution. Figure 7a) shows the initial states; the water temperature is set at 30 and is falling as the boiler is off. Figure 7b) shows that the temperature falls below the threshold 20 and the DES causes the boiler actuator model to move to *On*, turning on the LED. The temperature then rises. In Figure 7c) the upper threshold of 60 is crossed and the DES responds by turning the boiler actuator *Off*. This causes the LED to turn off and the temperature to fall. In Figure 7d) the temperature once again crosses the lower threshold value and the boiler actuator turns on again. In Figure 7e) we activate the occupancy sensor when the temperature was falling, and the water was *Warm*. The DES moves to *Warm-Off, Off, Occupancy-On*, and it generates an output *ON* to the boiler actuator. The boiler actuator model moves to the *On* state and turns on the LED and we observe the water temperature rise until it reaches the threshold for *Hot* state and turns off again.

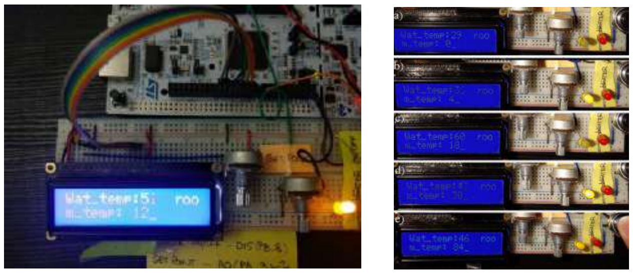

Figure 7: Hardware setup. Right: a) Initial states; b) Boiler on, temperature rises; c) boiler off, water temperature falling; d) boiler on, water temperature rises; e) occupancy sensor, water boiler on.

The DES was able to keep the room water temperature. In addition, we tested the effect of an occupancy sensor on the DES and actuator model and found it turns the boiler actuator on when the room temperature is in the *Warm-Off* state as we intended. This occurred even though the water temperature model did not know the occupancy sensor, showing we can add other sensors outside of the property we are modeling, provided the actuator is modeled to include the effect of the sensor. Finally, the states of the DES and the models stayed coordinated and the DES did not move away from the control path.

Table 1: States and transitions of the GSS: cartesian product and approximate method.

| Model | Cartesian Product States | Cartesian Product Transitions | Approximate Method States | Approximate Method Transitions | Percentage Reduction |
|---|---|---|---|---|---|
| Room Water Temperature | 80 | 304 | 80 | 284 | 6.6 |
| Room Occupancy | 16 | 48 | 16 | 40 | 16.7 |
| Room Smoke Concentration | 16 | 48 | 16 | 40 | 16.7 |
| Room CO2 concentration | 24 | 80 | 24 | 68 | 17.7 |

Table 1 lists the results of our experiment with the room water temperature model that we discussed here and other models not included here for the sake of brevity. The results show the number of states for the Cartesian method and the approximate method is the same in all cases (our method is not missing any combined states). For the room occupancy and room smoke concentration models the percentage difference is approximately 16.7% percent. The room CO2 concentration model has a slightly larger difference at 17.7%. Finally, for the room water temperature model, we observe that our method produces a 6.6% reduction. This is because our method rejects some transitions that are not possible to occur based on the structure of the coupled model. The Cartesian product, on the other hand, still computes all the possible transitions. A reduction of 6.6% is significant and in larger cases can cause a substantial reduction in the total number of transitions, making the next steps easier and more manageable to complete.

## 5 CONCLUSIONS

We used the DEVS formalism and devised an approximate method that replaces the Cartesian product to generate the state-space with a reduction in the number of transitions. The approximate method utilizes the structural information present in a DEVS coupled model to include or exclude transitions. The approximate method produces a GSS with either an equal number of transitions as the Cartesian product or less. We presented a methodology for obtaining a DES from a coupled DEVS model represented as a directed weighted graph and in the process. The method was used in a case study, producing a reduction of 6% to 17% in terms of transitions. The obtained DES and the model were simulated in software and then executed on an embedded platform, achieving the proposed objectives. This work can be extended to incorporate the use of timing analysis and reachability analysis identifying impossible states and transitions, reducing the complexity of the GSS. In addition, the use of timing analysis of GSS for embedded applications can be investigated for timing guarantees. The approximate method algorithm can be further developed to recursively search coupled models in the hierarchy of a DEVS model.

## REFERENCES

Hagberg, A., P. Swart and D. S Chult. 2008. "Exploring Network Structure, Dynamics, and Function Using NetworkX". *SCIPY 08.* Pasadena, CA.

Alves, M., Carvalho, L. and Basilio, J. 2017. "Supervisory Control of Timed Networked Discrete Event Systems". In *Proceedings of IEEE 56th Annual Conference on Decision and Control*, Dec 12th-15th, Melbourne, Australia.

Belloli, L., Vicino, D., Ruiz-Martin C. and Wainer, G. 2019. "Building DEVS Models with the Cadmium Tool". In *Proceedings of 2019 Winter Simulation Conference*, edited by N. Mustafee, M.Rabe, K.Bae, C. Szabo, S. Lazarova, P. Haas, and Y. Son, 45-59. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Brandin B. and Wonham, W. 1994. "Supervisory Control of Timed Discrete-event Systems". *IEEE Transactions on Automatic Control* 39:329–342.

Fadali, M. and A. Visioli. 2013. "Digital Control Engineering". 2nd Edition, Academic Press.

Forschelen, S., van de Mortel-Fronczak, J, Su, R. and Rooda, J. 2012. "Application of Supervisory Control Theory to Theme Park Vehicles". *Discrete Event Dynamic Systems* 22:511–540.

Giua, A. and Silva, M. 2018. "Petri nets and Automatic Control: A Historical Perspective". *Annual Reviews in Control* 45: 223–239.

Gohari P. and W. Wonham. 2000. "On the Complexity of Supervisory Control Design in the RW Framework". *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 30:643–652.

Grossman, R., A. Nerode, A. Ravn and H. Rischel. 1993. *Hybrid Systems*. Berlin: Springer Berlin.

Le Gall, T., B. Jeannet and H. Marchand. 2005."Supervisory Control of Infinite Symbolic Systems Using Abstract Interpretation" in *Proceedings of the 44th IEEE Conference on Decision and Control.* Seville, Spain.

Maione G. and D. Naso. 2004. "Modelling Adaptive Multi-agent Manufacturing Control with Discrete Event System Formalism" *International Journal of Systems Science* 35:591–614.

Marcosig, E., J. Giribet, R. Castro. 2017. "Hybrid Adaptive Control for UAV Data Collection: A Simulation-based Design to Trade-off Resources Between Stability and Communication". In *Proceedings of the Winter Simulation Conference*, edited by W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 1704-1715. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Niyonkuru D. and G. Wainer. 2022. "A DEVS Based Engine for Building Digital Quadruplets". *SIMULATION*. 97: 485-506.

Ramadge P. and W. Wonham. 1982. "*Supervisory Control of Discrete Event Processes*". Berlin: Springer-Verlag.

Silva, M. 2016. "Individuals, Populations and Fluid Approximations: A Petri Net Based Perspective" *Nonlinear Analysis: Hybrid Systems* 22:72–97.

Teixeira, M., J. Cury and M. de Queiroz. 2011. "Local Modular Supervisory Control of DES with Distinguishers". *International Conference on Emerging Technologies and Factory Automation*. Toulouse, France.

Wonham, W. and P. Ramadge. 1988. "Modular Supervisory Control of Discrete-event Systems". *Mathematics of Control, Signals and Systems* 1:13–30.

Wonham, W., K. Cai and K. Rudie. 2018. "Supervisory Control of Discrete-event Systems: A Brief History". *Annual Reviews in Control* 45: 250–256.

Yamalidou, K., J. Moody, M. Lemmon and P. Antsaklis. 1996. "Feedback Control of Petri Nets Based on Place Invariants" *Automatica* 32:15–28.

Zeigler, B., Song, S., Kim, T. and Praehofer, H. 1995. "DEVS Framework for Modelling, Simulation, Analysis, and Design of Hybrid Systems".In *Hybrid Systems II*, 529–55. Heidelberg: Springer Berlin Heidelberg.

Zeigler, B, T. G. Kim and H. Praehofer. 2000. *Theory of Modeling and Simulation*. New York: Academic Press.

Zhong H. and W. M. Wonham. 1990. "On the Consistency of Hierarchical Supervision in Discrete-event Systems". *IEEE Transactions on Automatic Control* 35:1125–1134.

## AUTHOR BIOGRAPHIES

**MAAZ JAMAL** has a Masters degree from the Department of Systems and Computer Engineering at Carleton University (Ottawa, ON, Canada). His email address is maaz.jamal@carleton.ca.

**GABRIEL WAINER** is a Professor at the Department of Systems and Computer Engineering at Carleton University (Ottawa, ON, Canada). He is the head of the Advanced Real-Time Simulation lab at Carleton University. He is a Fellow of SCS. His email address is gwainer@sce.carleton.ca, and his website is http://www.sce.carleton.ca/faculty/wainer.