Supervisory Control Using DEVS with Approximate Method and Hybrid Layer

by

Maaz Jamal

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Electrical & Computer Engineering

Carleton University
Ottawa, Ontario

Abstract

Supervisory control is a formal method for the control of Discrete Event Systems (DES). The benefit of using supervisory control is that it allows the use of Modelling & Simulation (M&S) techniques to model an application and then use the model to create controllers. The use of supervisory control is currently restricted to robotics due to the issue of state space explosion as model size increases. Reduction in the state space complexity can expand the practicality of the model.

We use the Discrete Event System Specifications (DEVS) formalism to implement supervisory control with an approximate method that reduces the state space complexity of the model. We also investigate the use of the hybrid layer to incorporate human interaction with a model and show that for certain cases a general approach can be used to reduce the complexity of the controller.

Acknowledgments

I would like to thank my supervisor, Dr. Gabriel Wainer, for his support, guidance and patience throughout the past two years. Without his help completing this work would not have been possible. In addition, I would also thank my colleagues Joseph Boi-Ukeme and Rishabh Jiresal for their help in technical matters and general comradery.

To my brother and parents, I thank them for their support and for helping me stay on track while I worked on this thesis.

Finally, to the folks at Paul Menton Centre and Carleton Health & Counselling, I thank them for their guidance, while I navigated through the available supports for my disabilities.

Table of Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

Abbreviation	Definition
AC	Air Conditioning
BFS	Breadth First Search
CPN	Continuous Petri Net
CPS	Cyber Physical System
DCS	Distributed Control System
DCT	Discrete Control Theory
DEC	Discrete Event Controller
DEMES	Discrete-Event Methodology for Embedded Systems
DES	Discrete Event System
DESS	Differential Equation Specified Systems
DEVS	Discrete Event System Specification
EIC	External Input Coupling
EOC	External Output Coupling
EXT	External
FD-DEVS	Finite and Deterministic DEVS
FSM	Finite State Machine
GMEC	Generalized Mutual Exclusion Constraints
GSS	Global State Space
HPN	Hybrid Petri Net
IC	Internal Coupling
INT	Internal
LAN	Local Area Network

LCD	Liquid Crystal Display
LED	Light Emitting Diode
OS	Operating System
PLC	Programmable Logic Controller
PN	Petri Net
RT	Real Time
RTDEVS	Real Time DEVS
SCADA	Supervisory Control And Data Acquisition
SCT	Supervisory Control Theory
STS	Symbolic Transition Systems
TCPN	Timed Continuous Petri Nets
TDES	Time Discrete Event System
UAV	Unmanned Aerial Vehicle

Chapter 1: Introduction

Supervisory Control Theory (SCT) [1] is a branch of Control systems theory [2] applied to Discrete Event Systems (DES) [3]. Control systems have existed since at least 300 BC; specifically, feedback control dates to the Greek civilization where a float was used to control the water level in a tank and measure time [4]. The formal definitions of control systems were developed during the industrial revolution in the 17^{th} and 18^{th} centuries [4]. The machines of the industrial revolution were regulated using a variety of mechanical control devices that controlled temperature, pressure, etc. The simplest example of a control system, to this date, is the water level control in a tank using float (plastic air ball) which rises with the rising water level and eventually closes the valve [3].The major driving force in Control theory has always been industrial processes and process control. With electronics taking over mechanical control, new techniques came along that were more suited to the new digital sensors and controllers and with electrical actuators. Digital Control Theory [5] translates non-digital continuous linear control system to digital control systems with discretization introduced by the tick of a digital clock.

The complexity of a digital system has been increasing over time with automation and continuing research in robotics and autonomous driving. These systems can no longer be modeled by differential equations only. Since the 1960s, this was tackled using Finite State Machines (FSM) and automatons to represent the some of the subsystems as Discrete Event Systems (DES) and make them more manageable [6]. In a DES, the system is described by discrete events and transitions between those events.

Control of systems revolves around the concepts of feedback, stability, controllability, observability, and optimality, each of which has a significant importance when it comes to controlling a physical system [6]. Feedback provides the state of the system at any time allowing the controller to decide the next control response to achieve stability. Similarly, controllability and observability are the properties of the physical system. Controllability is dependent upon access to actuators defining what parts a controller can influence directly. Observability is dependent upon the sensors, as these sensors define what properties of the system the controller knows of. Controllability and observability can therefore be increased by adding more actuators and sensors, respectively. There is always a trade-off between price, complexity, and value gain [4]. Quantitative optimality is achieved by tweaking the controller to get the response we require. SCT which is also referred to as DES control theory [3] implicitly incorporates all these concepts.

There is a stress on enhanced testing, verification, and synthesis of controllers to make systems safe and reliable. Testing involves passing the system through a series of scenarios to confirm that it works within parameters. For complex systems, the scenarios can become prohibitively large to test, in addition searching for all these scenarios becomes impractical [7]. In contrast, formal verification is a better mechanism to check for safety and reliability as it uses formal methods to show that the system works as designed, without the need rigorous testing at the cost of scalability. Control synthesis formally utilizes the plant equation, output, and expected response to generate a new controller that fulfills the required specifications. Synthesis and verification are better than testing, at making a

reliable system. Since SCT synthesizes a controller based on formal specifications of the system, it embeds reliability into the system design [8].

SCT involves modeling the system, called a plant, which is viewed as a DES. DES models systems that cannot be described completely using closed-form expressions [3] i.e. those systems that cannot be described by an exact equation and that can be computed in a finite number of steps. DES can be formally modeled using Finite State Machines (FSM) [9], Petri nets [10], Discrete Event System Specification (DEVS) [11], and other formal techniques.

Following a formal model, control specifications are formally defined. The event of the plant can be described as alphabets that are either controllable or uncontrollable. The role of the supervisor is to disable the appropriate amount of controllable alphabet so that the maximum number of the control specifications are met [1]. In control theory, the controller acts as a forcing agent and forces the control output over the plant while in SCT the supervisor acts as a disabling agent and generally fulfills its operation by disabling controllable parts of the system [6].

SCT was first defined for monolithic models, with a single supervisor controlling the entire system. When this approach is applied to practical examples, the state space explodes exponentially [12] with each additional model of a system component added. Thus, synthesizing a supervisor from the state-space of DES models can quickly become infeasible. Reducing the complexity of the state-space remains a challenge for SCT and is the focus of this thesis.

Modular and hierarchical architectures were presented to reduce state space complexity. Modular approach splits the model into smaller modules and thus reducing the state space for each module and a supervisor is synthesized for each one of them [13].

Hierarchical systems have a minimalistic supervisor at the top that is only interested in very significant events of the plants on the lower level. This is done by creating a smaller model at the higher layer than the underlying low-level model. The supervisor is then synthesized on the high-level model which imposes high-level specifications.

To broaden the applications and computational techniques that can be applied to the models, researchers extended the use of SCT to other formalisms like extended FSM, Petri Nets state model, and Vector DES, as well as computation techniques like combining state charts and state tree structures and using binary decision trees for computation [6].

In this thesis, we define an approximate modified method to build modular supervisors based on DEVS and show that such an approach allows for a manageable state-space and practical implementation using Discrete-Event Methodology for Embedded Systems (DEMES) [14] of a supervisor and system. The DEVS formalism is a modular and hierarchical modeling system consisting of atomics models and bigger structural components called coupled models composed of atomics and other coupled models. Our method is based on the methodology used by Zeigler et al. [15], who showed a method exists that can be used to synthesize controllers for a DES system and hybrid systems.

DEMES allows model continuity between software and embedded environments. Thus, allowing models to be defined using the DEVS formalism, extensively tested in software simulations and finally the models can be ported and executed on embedded hardware. This then gives us a complete method where we can model the system formally

using the DEVS formalism, synthesize a supervisor, and then simulate the model on software and execute the models on hardware.

Furthermore, we use the hybrid layer [16] to introduce human interaction to a DEVS modeled system and utilize the hierarchical approach to make a simpler top level supervisor and use the hybrid layer to hide the complexity of the models at the lower level.

There are two major contributions of this thesis. The first and the principal contribution is the approximate method to generate the GSS of a given DEVS coupled model in the process of obtaining a supervisory Discrete Event Controller (DEC). The approximate method utilizes the structural information of the coupled model to generate the GSS and in doing so reduces the number of transitions and therefore state-space complexity of the GSS.

The state space explosion issue occurs during the calculation of the Global State Space (GSS), thus making the remaining steps of the method unfeasible. By reducing the number of transition during the generation of the GSS, the remaining steps become more feasible.

We show how the method works, the justification for the steps in the method, and the process of applying the method given a DEVS model of a system from start to finish. Since we utilize DEVS, we can model a large class of hybrid systems and with Real-Time (RT) DEVS [17] extension and DEMES, the DEVS model can be ported to and executed on embedded platforms with model continuity. Model continuity is the property that the model has the same formal definition and implementation across different simulation environments and platforms. Thus, we can apply Modelling and Simulation (M&S) techniques throughout the development process. To show these aspects, we introduce a

case study of a room in a smart building application with various sensor and actuators for room temperature control etc., with the goal of showing the method produces a less complex GSS and a working DEC.

The second contribution is investigating the use of the hybrid layer to reduce the number of transitions of a system with human interaction. We show how human interaction with the system can increase the number of transitions and then devise a general approach to implement the hybrid layer such that it hides and removes the complexity of the human interaction from the GSS. The resulting DEVS model is a hierarchical structure with the supervisory DEC on top, the hybrid layer in the middle, and the sensors, actuators, and human interaction models in the bottom layers.

By using an abstract model the procedure to obtain the supervisor becomes easier as the source model is less complex and therefore less states and transitions need to be considered by the approximate method or the cartesian product. The resulting controller is also smaller and simpler and therefore easier to analyse and execute on embedded platforms.

Minor contributions of the thesis include utilizing the RT-DEVS and DEMES to execute the supervisory DEC and models on embedded platforms and experimentally validating the models given by Zeigler [15]. Additionally, we identified some issues that can occur when approximating time in a GSS and steps to resolve them.

The thesis makes a few assumptions and simplifications in achieving the goals. For the approximate method, zero time advance is not considered as it introduces uncertainties about the order of execution of the transitions. In addition, we describe the atomic models as directed weighted graphs where the nodes are the states and edges are transitions, with

the weights describing the input/outputs ,ports and time advance associated with the transition, this allows us to compute the GSS easily. Finally, the TOP coupled model is assumed to have only atomic models as components.

For the use of the hybrid layer, we define the necessary requirements that would allow an abstract model to be created. We also assume in our case study that when the setpoint changes the system does not carry out any time consuming operations and therefore the time advance duration can be reduced, while the hybrid layer reaches the desired state. Also, the change of setpoint is not a time critical change and can be delayed letting the hybrid layer reach equalize the desired state with the current state.

The thesis is organized as follows: Chapter 2 details the history of supervisory control, the state of the art in the Petri nets, and examples of applications proposed and implemented in literature for SCT. Finally, the chapter ends with the discussion on DEVS and hybrid layer and the proposed techniques for using SCT with DEVS, with a focus on the methodology of Zeigler et al. [15] on which this thesis is based.

Chapter 3 defines the scope and objectives of the thesis. Chapter 4 explains the methodology in detail, explaining the cartesian product, the approximate method, and the rest of the methodology for a directed weighted graph. The chapter closes with a section on the hybrid layer and the problems with introducing human interaction in the system. It also specifies the criteria for extraction an abstract DEVS model from the DEVS system model and defines the general algorithm to cater for human interaction.

Chapter 5 presents a case study of a room controller in a smart building on which the methodology defined in Chapter 4 is applied. Comparisons with the cartesian product

are carried out and the models and obtained DECs are passed through software simulations and executed on an embedded platform for experimental testing.

Chapter 6 ends the thesis with a conclusion, outlining the results, and presents a few avenues for further research in this topic area.

Chapter 2: Background

The state-space explosion problem made SCT impractical in its original form. The theory however started gaining traction as the number of systems modeled as Discrete Event Systems (DES) started to grow. DES applies to many systems that are discrete such as computer systems, communication, and software systems [6]. DES is also applicable at a higher level of abstraction to hybrid systems which include both discrete and continuous components, such as automation systems, manufacturing and process control systems, transportation systems, and other autonomous systems.

In this literature review, we will go through the general development of the theory in the early years to some of the work done in Petri Nets, its challenges, and then eventually to the recent adoption into various fields, particularly robotics. Though still in its niche stage, the theory has been applied to and developed for many versatile applications. Finally, we will discuss the work done in DEVS and study DEVS and the hybrid layer in detail.

2.1 Early Years of Supervisory Control

Ramadge and Wonham [1] introduced the concept of deriving supervisory controllers from a formally defined DES. The conceptual framework of SCT devises a control method that disables discrete event transitions to enforce a control loop that satisfies control objectives. The synthesis of supervisory controllers suffers from space and computational complexity issues, and this led to the introduction of modular and hierarchical supervisors.

The modular approach utilizes the modularity of the plant to isolate subsets of control problems of a plant and fulfill a single specification. The subset of the plant has

interactions locally thus we can ignore the other subsets and reduce computational overheads and state-space complexity [18]. However, the resulting system has issues of its own. The coordination between supervisors becomes a very difficult task as there is a risk of race conditions and deadlocks. Attempting to solve these problems using global coordination becomes computationally unfeasible [19].

The hierarchical approach creates a hierarchy where a top-level supervisor has a limited view of the system on lower levels. The supervisor assumes that its control instructions will be implemented at the lower level faithfully. However that is not the case because the high-level model is made using inconsistent information. This issue is also known as a hierarchical inconsistency [20].

The automata considered so far, in the modular and hierarchical approach do not consider time. Therefore, Brandin and Wonham [21] presented the use of SCT for timed automata systems The work expanded the use of SCT to temporal systems. A class of systems with timing delays between transitions. They also introduce the use of forcible events and disabling as means of supervisory control. The former forces an event before the global tick of the clock and ensure the system adheres to specifications. The latter extends the time delay of an event indefinitely and thereby ensuring the event does not occur. This is akin to passivation of DEVS models, which will be discussed later.

Lin and Wonham [22] generalize the Time Discrete Event System (TDES) for the case of partial observability, i.e., conditions where the supervisor does not have full knowledge of all the states of the system and present a definition for coherent languages. The work presents the necessary conditions for the existence of the supervisor and methods for synthesizing the supervisor. Takai and Ushio [23] further extends the work by

modifying the supervisor such that it has greater control over which forceable events can be enabled or disabled. By doing so they can synthesize supervisors for non-coherent languages at the cost of exponential increase in computation time. Le Gall, Jeannet, and Marchand [24] define an all-inclusive controller that can be used to synthesize maximally permissive supervisors for TDES under partial observation. The controller satisfies safety conditions. More recently, Alves, Carvalho, and Basilio [25] investigated delays and loss of observation between plant and supervisor (as is the case for practical physical systems). The work proposes a method to generate an untimed model of the system and then presents conditions for the existence of supervisors and controllers. However, timing delays from control actions were not considered.

DES formalisms generally treat the values of data variables as distinct states. This causes state space explosion as the values of the variable need to be enumerated for analysis. If the variable can take values from an infinite domain, then computation becomes infeasible. For such cases, Le Gall, Jeannet, and Marchand [24] investigated the use of Symbolic Transition Systems (STS) and proposed the use of abstract interpretations where the set of uncontrollable states is over-approximated. This then allows the system to become decidable and a supervisor can be synthesized.

2.2 Petri Nets for Supervisory Control

Petri Nets (PNs) were defined to describe systems that have concurrency and need synchronization; and therefore their use for supervisory control was considered. They also allow algebraic techniques to be applied for synthesis and analysis [26]. Early works in PNs focused on defining the classes of PNs for which supervisory controllers could be synthesized. In [27] SCT for a class of controlled PNs called Cyclic controlled Marked

Graph (CMG) was defined. CMGs allow external binary inputs to the PN and disable transitions of the PN. The controller in this method can be derived without an exhaustive search of the state-space. Yamalidou, et al. [28] used the PN property of 'place invariants' to compute the supervisory controller, which obeys a linear equation involving place invariant vectors. The 'place invariants' are regions of the net where the number of tokens remains constant for all markings. The supervisor size is proportional to the number of constraints and is a PN consisting of only arcs and places. This method applies to a more general class of specification for PNs called Generalized Mutual Exclusion Constraints (GMECs).

GMECS has issues where uncontrollable transitions exist. In these cases, the supervisor might need to disable these transitions. This creates a situation where the system becomes uncontrollable as the transitions are uncontrollable and therefore the supervisor cannot control these transitions. While solutions have been proposed resolving this issue is still an active research topic [29].

As model size increased the state-space complexity of the PNs increased as well. To combat the state space explosion problem fluid PNs were introduced. Fluid approximations relax the requirements of Discrete PNs by allowing markings to take on real negative values. This can allow control techniques and algorithms from the continuous-state system domain to be applied to the DES [30]. The fluid approximation can work for Timed Continuous Petri Nets (TCPN) and untimed Continuous Petri Nets (CPN). If all the markings and transitions are fluid, then the model is considered continuous otherwise if only some are fluid, then the model is considered a Hybrid Petri Net (HPN). A major concern here is the accuracy of the fluid PN approximation as compared to the

DES model. As of now, a general fluid approximation does not exist that would alleviate discrepancies between discrete and continuous behaviors [29].

The controllability of Timed Continuous Petri Nets (TCPN) is dependent upon the number of controllable transitions. Control actions involve temporarily blocking enabled transitions from firing. If all transitions are controllable then the system is controllable independent of initial marking or timing, otherwise, the system controllability is dependent on initial marking, timing, and net structure [31].

The problem of observability for TCPN and fault diagnosis for CPN is discussed in [32] and [33]. A TCPN is observable if its initial marking is computable at any point. Initial marking is the starting placement of tokens in the Petri Net. Fault detection in CPN does not require the condition that a cycle of unobservable transitions must not exist and additionally CPN computational complexity for diagnosis is much smaller than that of DES.

Another approach to combat state space explosion is the use of HPNs. HPNs can model a larger class of systems as compared to ordinary Petri nets. However, this comes at the cost of complex models that are difficult to analyze formally. HPNs can take different forms, one of the earliest forms is HPNs with continuous places, where the discrete and continuous parts are modeled using Discrete PN or continuous PN. The two discrete and continuous nets are connected through arcs as shown in Figure 1. there are two nets; one continuous and one discrete, and the arc is connected from the transition of one to the place of the other [34].

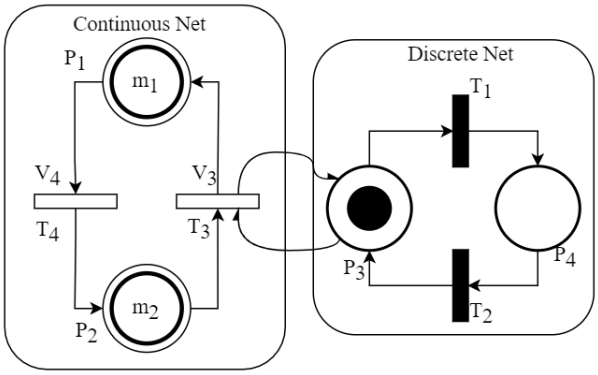

Figure 1. Hybrid PN with continuous and discrete nodes and arcs [34] .

More recently Markovian Hybrid PN were defined adding an approximation of Markovian PNs to the discrete component of the system by fluidifying the transition [35]. Fluidifying the transitions involves changing a discrete transition to an approximate continuous one.

Another approach has been to use a classifier to distinguish between safe and unsafe states. The classifiers are constructed from linear inequalities through which the state vector is passed, obtaining a set of safe and unsafe states. The supervisor can then be extracted such that it only transitions to safe states of the system [36]. By doing so the state space of the entire PN can be avoided and only safe states can be focused on.

PNs have some limitations. The efficient techniques for synthesizing the supervisor require restrictions to be placed on the PN to allow for supervisor synthesis. The different approaches are suited to a specific class of specifications that arise in real-world applications. However, these do not cover all applications and, therefore, PNs do not currently have a general methodology to design controllers with more general specifications. In addition, for timed PN the current method of discretization of the clock produces more complex models [29].

With DEVS the methodology for obtaining a supervisor applies to any hybrid system that can be modeled as a DEVS. In addition, DEVS uses a continuous-time base for the representation of time, this may create fewer complex models as compared to timed PNs.

2.3 Implementations of Supervisory Control in Literature

The purpose of this section is to provide examples of the use of supervisory control in the literature and some of the techniques used to model practical systems and simplify control problems. Some of the techniques used here were also used in the case study in Chapter 5, for example, the manual removal of impossible states is mentioned in this section.

2.3.1 Use of Supervisory Control in Process Control

Process control is a discipline that is used to control industrial processes such that the system achieves a consistent level of production. Process control has been developed mainly around traditional control theory. With the discretization and computerization of the industrial sector, the theory was adopted into the discrete world thus giving rise to Discrete Control Theory (DCT) and Distributed Control Systems (DCS). Supervisory Control And Data Acquisition (SCADA) systems are mainly used in process control. The supervisor in SCADA is to be distinguished from the supervisor synthesized in SCT. In SCADA the supervisor is a human operator or a computer that has limited control over the plant; setting the setpoints of the control loops and acquiring data and monitoring the plant distributed systems. The term supervisor was historically used in process control before SCT was proposed and has remained in use.

Adopting SCT to process control creates issues due to differences in the SCT approach and the current physical process control systems available [37]. For example, Programmable Logic Controllers (PLCs) are one of the most common controllers used in process control. The nature of PLCs does not lend itself to working well with SCT in practical implementations. Fabian and Hellgren [37] list some of the issues PLCs and other controllers used in process control have with SCT.

1. PLCs work with continuous signals whereas SCT works with discrete events.
2. A Supervisor cannot generate an event rather it fulfills its task by disabling events. PLCs do need to generate events for normal working of the plants.
3. A supervisor must be minimally restrictive, meaning there need to be open paths all the time for the system to take thus requiring changes to software structure and overall system to work asynchronously as opposed to being synchronous.

Basile, Chiacchio, and Gerbasio [38] show these issues can be resolved with good programming and plant design. Furthermore, by employing the benefits of modularity of manufacturing plants and industrial processes, local modular models can be created avoiding the state space explosion issue that arises due to the scale of manufacturing plants [39].

Practical programming of the PLC based on the supervisory model is difficult, to make physical implementation easier supervisor reduction can be employed before PLC programming [40]. In [39, 41] methods for implementing PLC in SCT are discussed. There has also been an effort to create methods to obtain programs and PLC code from the synthesized supervisors [42].

Safety is of prime importance in process control. Advancements in industrial hardware with time have now embedded safety with PLCs with safety PLCs [8]. To avoid undue complexity, the specifications are split into safety specifications and regular specifications followed by splitting the supervisory controllers into safety and regular supervisory controllers.

Operators are sometimes needed to arrange the setpoints, overlook the sensor readings for safety, resolve machine conflict, etc. Setpoints are the values of a property that the controller needs to maintain or reach. In the case of supervisory control, operator intervention needs to follow certain rules so that they don't violate the system specifications. In anticipation of any operator intervention, the synthesized supervisory can be embedded with the ability to guide the operator and avoid blocking and deadlocks and prevent the control specifications from being violated [43].

2.3.2 Use of Supervisory Control in Robotics

Control of robots has previously been dependent on traditional control theory; however, SCT has been adopted in robotics research due to the use of automaton models in robotics. As with process control and manufacturing, SCT must be distinguished from 'supervisory control' done in various literary works such as [44].

In [45] the state-space explosion problem was addressed for single-purpose autonomous robots. Their state machine design of a 'sumobot' (a robot that pushes other robots in a sumo competition) has some extra steps which greatly reduces the number of states and transitions. First, they only considered states from their sensors that they were interested in. Secondly, they removed the states that were physically impossible to occur from the combined state space of their robot model. For example, in this case, a state where

the sensors on the opposite side of the robot become high is impossible to occur as the other robot in the dojo can physically only appear on one side. Using this approach of physical infeasibility removal other single operation robots can be designed with reduced state-space complexity.

However, most practical robots do not operate in a controlled environment and there is always some degree of uncertainty. In addition, there is also uncertainty in sensor reading and actuator functioning. SCT in these cases can be applied using Fuzzy DES that can model the uncertainty of the sensor behaviors [46].

A rather interesting application of robotics is swarm robotics [47]. In this, multiple simple robots are programmed to coordinate and work together to form a large single system. Most of the work done in swarm robotics is ad hoc and therefore is unable to scale and deploy to real-world systems and is hard to maintain. Lopes et al. [48] proposed the use of SCT for swarm robotics in a bid to reduce ad hoc development. Additionally, they tested for controller reusability across different platforms so that time to market can be reduced.

They tried three supervisor synthesis approaches, monolithic synthesis, modular synthesis, and experimented with an approach called local modular synthesis [39] which explores modular behavior along with reducing the number of free behavior models being used for synthesis. Free behavior models describe every capability of the system i.e., whatever a system can do. Therefore, the size and number of free behavior models can be reduced as not all the capabilities of the system are needed for every scenario. Surprisingly in [47], the modular approach gave twice the size of the state-space model for robot

grouping and segregation experiments than a monolithic supervisor. The local modular approach had the best result in terms of model size in each test category.

Autonomous driving is yet another area of robotics. Its goals include improving safety on the road by removing human error, enhancing performance, and reducing the need for human labor. SCT has been applied to autonomous vehicles in [49], where warehouse robots were automated. A comparison was done of global control of the robots versus localized control of the robots where the robot takes control decisions on its own. The performance of localized control was the same as global control, however, localizing control has the benefit of being easily scalable. They also used a software package called TCT [50] to create an automaton and synthesize the supervisor. The paper demonstrated the design procedure and applied it to three actual robots with satisfactory results.

In [51] SCT was applied to a vehicle called a multicover which could be used in theme parks or museums to carry tourists. The purpose of the paper was to prove that formal methods for designing controllers are suited in an industrial setting as well, and are more scalable, maintainable, and reliable. They were successful in modeling, synthesis, verification, and implementation of the multicover and were successful in reducing the time to develop and the reusability of code. The main priority of the supervisor in this vehicle is safety. The vehicle functionality is distributed so that the controls can be applied locally.

The model used was restricted (partially controlled) as opposed to unrestricted (uncontrolled). The supervisor can be synthesized easily for the restricted model as compared to the uncontrolled model (having all the physical capabilities of the vehicle). To synthesize the DEC both centralized and distributed approaches were tested. For the

centralized supervisory controller, the event-based model was hard to deal with due to the problem of scale, so a state-based approach was used [51]. For distributed supervisory control they used two different event-based approaches [52] and [53]. In the state-based approach, the supervisor only has access to state information and in the event-based approach, the supervisor only sees the sequence of observable events.

The results of [51] conclude that formal methods provide a better structural approach in the design of control software. They have fewer ambiguities and are more reliable as the correctness of the model is embedded in the design. It is also seen that the product is easily changeable (with changing requirements) when the formal method is used compared to traditional engineering processes.

Chen et al. [44] used SCT at the behavior level and operation level to control complex autonomous vehicles with multiple modules. There was latency between the modules as they were working across different computers and were connected through a LAN connection. The vehicles can be commanded by a human teleoperator. The result showed improved performance of the vehicles in all the different behavior scenarios they tested. This was adopted in autonomous military vehicles as well where the vehicle can act independently or under human supervision. Aigner and McCarragher [54] discussed human integration into autonomous systems by modeling humans as a DES.

For supervisory control of hybrid or highly complex systems, traditional single controller and supervisory methodologies can sometimes fail. In these cases, switched control systems [55, 56] were devised. In switched systems, a bank of controllers is available. These controllers are obtained by having different control requirements for each controller. Based on the condition of the system, the supervisor switches to the appropriate

controller for the current measurements of the system. The benefit is the supervisory controller is derived from a smaller state space, where its purpose is only to enable and disable the controllers based on the conditions of the system model. This is yet another way of SCT introduced in literature.

2.3.3 Use of Supervisory Control in Software

SCT can also be applied to software programs to make them fault free and controlled by a supervisory controller without any structural modifications to the application or the OS [57]. By modelling the program and restricting the OS actions on the program, the programs execution can be controlled. In case of a fault like hardware or memory faults, the supervisor points the software to a safe state. More recently there has been an effort to combine SCT and reactive synthesis [58] and apply it to terminating [59] and non-terminating [60] software processes.

We have already seen the use of TCT [50] for modeling automata for supervisory control in this section. There are additional tools to help with the modeling systems as automatons like Supremica, libFAUDES, and IDES. Of these Supremica is highly scalable due to its use of binary decision diagrams to store the transition functions of automata [61]. Supremica has been used to synthesize supervisors for practical implementations [62], comparative studies [63], and analysis purposes [64, 65].

2.4. The DEVS Formalism

In this section, we give a brief overview of the DEVS formalism and some of the definitions and notations used in this thesis. DEVS is a modular and hierarchical modeling formalism for systems whose state transitions are driven by discrete events [11]. The systems, in

general, can be DES and continuous state systems. The continuous state systems must be Differential Equation Specified Systems (DESS) as they have piece-wise constant input and output segments. The discrete events, in this case, are the jumps between these piece-wise constant segments. The DEV&DESS extension to the DEVS formalism can then model continuous state as well as DES systems as shown by Zeigler et al. [15]. Combining both, DEVS can model hybrid systems with continuous and discrete components [66].

DEVS models consist of two components, atomic models and coupled models. Atomic models are modular and model a component using states, transitions, and events that trigger the transitions. An atomic model is defined as [67]:

$AM = <X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta>$

Where 'X' is the set of input events from an input port. 'Y' is the set of output events from an output port. 'S' is the set of states of the models. 'δ_{int}' is the internal transition function and moves states from S to another state in S. 'δ_{ext}' is the external transition function and moves the states of S from one state to another based on input event from 'X'. 'λ' is the output function and based on the current state from S generates an output.

'ta' is the time advance function and can be updated by the internal or external transition functions. The 'ta' can be set to a finite or infinite time period, after a transition the elapsed time 'e' starts at zero and increases. When 'ta' equals 'e' the internal transition function and the output function, in theory, are executed simultaneously. The external transition function only executes when input arrives at an input port. A model can be 'passivated' when the time advance function is set to an infinite time, as then the model will wait and indefinitely until the internal transition function is executed again. A

passivated model can only execute external transition due to input events. In addition, variables can be defined in the internal and external transition functions that can set the conditions for the triggering of a state change by the transition functions.

Coupled models define the hierarchical structure of the model and are composed of coupled or atomic models, defining the couplings between the ports. A coupled model CM is defined as [67]:

$CM = \langle X, Y, D, M_d, EIC, EOC, IC, \textit{select} \rangle$

Where ‘X’ and ‘Y’ are the set of input and output events of the model. ‘D’ is the set of component names, where ‘d’ ∈ ‘D’. ‘M_d’ is the set of atomic or coupled models in the top coupled model. The External Input Coupling (EIC) defines the coupling between input ports in ‘X’ and the input ports of components of ‘D’. The External Output Coupling (EOC) defines the coupling between output ports in ‘Y’ and the output ports of components of ‘D’. The Internal Coupling (IC) defines the input couplings between the output and input ports of two components from ‘D’. The *select* function acts as a tiebreaker in case more than one component needs to execute its transition functions at the same time. Coupled models are closed under coupling, meaning that a coupled model can be represented as an atomic model.

In this thesis, we define the atomic models as a directed weighted graph and use the DEVS graph notation [68] as shown in Figure 2 to represent the models graphically.

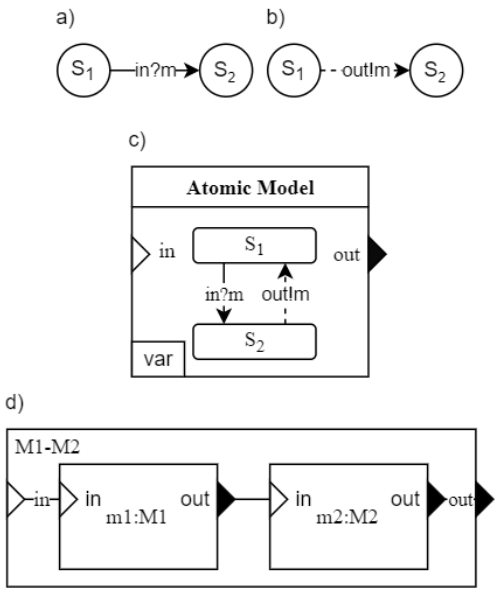

Figure 2. DEVS graphical notations. a) External transition notation; b) Internal transition notation; c) Atomic model notation; d) Coupled model notation. [68]

In Figure 2a) an external transition between two states 'S_1' → 'S_2' is shown. The external transition is represented by a solid arrow and the label 'in?m' represents the input port 'in' and 'm' is the message or event that triggers this transition. A question mark is always used '?' to separate the input port and input event. In Figure 2b) an internal transition is shown, here the transition is represented by a dashed line, and '!' separates the output port 'out' and the generated output 'm'. It should be noted that the output is generated by the output function that executes simultaneously with the internal transition and is hence represented with the internal transition. Furthermore, the '@' symbol can be added after 'out!m' and it can be used to specify conditions for triggering a state change in the internal transition function.

In Figure 2c) an atomic model is shown, named 'Atomic Model' here. The input port is a triangle that is on the inner line of the model and labeled 'in', the output port is a filled triangle on the outer line of the model and is labeled 'out' here. The states 'S_1' and

'S_2' and their internal and external transitions are shown. The small rectangle in the bottom left defines the name of the variable, 'var' in this case, that this model contains and is used to specify conditions for the internal transitions.

Finally, Figure 2d) shows a coupled model with the name 'M1-M2' in the top left corner. The components are inside the larger rectangle of 'M1-M2' showing the hierarchy and can be either atomic or coupled models. For a component 'm1:M1' 'm1' is the named component '*d*' and 'M1' is the base DEVS model belonging to 'M'. The EIC is defined between the input port of the coupled model 'in' and the component 'm1' port 'in'. Similarly, the EOC is shown between the output port of component 'm2' 'out' and coupled model 'out', and the IC is defined between the 'm1' output port 'out' and 'm2' input port 'in'. We will use this notation throughout the rest of this thesis unless otherwise specified.

Model continuity across platforms is important to ensure the specifications of the model remain intact and therefore ensure models are formally equivalent on any platform. Thus, results from model testing across the different platforms can be compared with one another. For this reason, in [14] and [69] Discrete Event Modelling of Embedded Systems (DEMES) is defined to allow the use of Modelling and Simulation (M&S) techniques based on DEVS on software as well as embedded platforms, ensuring model continuity. Using this method, the models can be simulated in software and then be ported and executed on hardware and we can be assured the models are the same in both implementations.

For simulations, we will define the DEVS models using the Cadmium tool [70]. Cadmium is a DEVS simulator written as a C++17 header-only library [71]. Due to its header-only nature, it can be added to a variety of applications easily and allows an easy

method to define DEVS models. It also implements model checks to ensure the models are written correctly and do not feature any bugs.

Real-Time (RT) DEVS [17] is an extension to the DEVS formalism that adds a time interval function and activity mapping function. Unlike DEVS, in RT-DEVS the time between '*e*' = 0 and '*e*' = ta is filled with activities using the activity mapping function, and the time interval function defines the timing bounds for these activities. Time, therefore, proceeds according to the execution or real-time of the system due to these activities. Ben et al. [72] developed an RT kernel for the Cadmium tool based on DEMES. This kernel allows models written using the Cadmium tool to be ported and executed on Mbed-OS [73] or Linux [74] based embedded hardware in real-time. The kernel measures the execution time of the embedded hardware and therefore runs in real-time. We use this RT-Cadmium tool to write the DECs and models in Chapter 5 and use a Mbed-OS based Nucleo F207ZG board [75] to execute the models on embedded hardware.

2.5. Supervisory Control in DEVS

DEVS lends itself to modelling hybrid systems [76], a class of systems that feature both discrete and continuous models of a system. Hybrid systems are present in a wide array of practical applications that cannot be adequately modelled by timed automata.

As previously mentioned, our method is based on the work of Zeigler et al. [15]. From our literature review this work forms the basis of the work done in DEVS for supervisory control of hybrid systems. However, in our search we find that the literature is lacking in terms of works related to SCT for DEVS. Instead, we find DEVS is used to model and control hybrid systems and utilizes other techniques to design or derive controllers.

In this section, we go over these works in general that are closely related to control of hybrid systems and discuss the hybrid layer.

2.5.1. DEVS Formalism for Control of Hybrid Systems and the Hybrid Layer

Starting with the control of hybrid systems, in [77], the authors define multiple Intelligent DEVS models that can be used as controllers for autonomous agents. The control models use soft computing approach to control the models and include Fuzzy DEVS, Genetic Algorithm DEVS, Stochastic Learning Automata DEVS and Neural Network DEVS. These controllers are intelligent in that they react to the inputs from the system and can change their behavior over time. In [78], the Fuzzy-DEVS controller is used to derive controllers for the application of autonomous agents, showing simulations in virtual environments can be transferred to the real world.

Maione and Naso [79] create a genetic event supervisor that can adapt over time to fit the conditions of the system. By using DEVS to model the parts and machines, the evolution of a system can be computed faster than real-world dynamics would allow, and thus the controller can react much more quickly. DEVS also allows the authors to translate their models directly to software thus saving developmental time and costs. Later in [80], the method is adapted for agile manufacturing.

Koutsoukos, et al. [81] provide an in-depth introduction and discussion on supervisory control of hybrid systems. The basic model shown in Figure 3, is useful for the control of hybrid system plants and has a three-layered structure with the DES controller on top, a hybrid layer interface in the middle, and a continuous layer at the bottom. This model is useful for a large swath of control problems involving hybrid systems. However, certain hybrid systems may not have a clear separation between continuous and discrete

components of the plant. Having this model, a DES model of the plant can be created that can approximate the continuous plant and a supervisory controller can be synthesized.

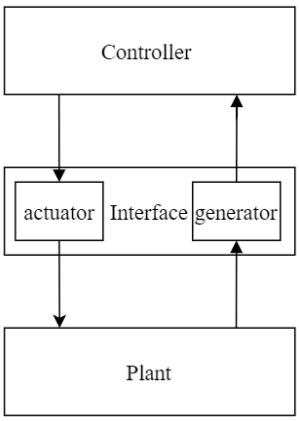

Figure 3. The hybrid layer approach [81] .

This paper assumes the continuous generator and actuator are already defined and can adequately translate between discrete and continuous signals. More importantly, it does not cater to a sudden jump in the system and the hypersurface (boundaries of a thermostat e.g.) do not change. We use the hybrid layer to cater to sudden jumps caused by changes in the setpoints of the system.

Marcosig, Giribet, and Castro [82] demonstrate the use of DEVS to verify, validate and design event-based controllers for hybrid systems. The use of DEVS allows iterative development and model continuity that ensures the hybrid systems work as intended. Further, in [83] the hybrid layer is used to permit complex interactions with a continuous environment using a discrete controller. However, in these cases, an iterator-based controller modeled as an FSM is used instead of a supervisory controller.

Abstraction is a useful method to reduce the complexity of a hybrid system and involves creating an abstract view of the system. In [84] the process for computing

abstractions for continuous and hybrid plant systems is described. In addition, they introduce a compositional method that allows smaller hybrid systems to be combined into a larger complex hybrid system. In [85], error detection in the automation logic of a hybrid system is presented. The finite abstraction of an infinite-state hybrid system allows a supervisory controller to be extracted that can then show a specification is guaranteed to fail. Through this proof by falsification, the correctness of the logic system can be gauged, and we can avoid the computationally expensive process of using the reachability of hybrid systems to gauge correctness. We have used the ideas of abstraction to introduce human interaction with the system.

Formal analysis of DEVS models is done in [86], the authors demonstrate a method of generating a finite reachability graph of Finite and Deterministic DEVS (FD-DEVS) models. FD-DEVS are a subset of DEVS whose states and events are finite, and the behavior of the functions is deterministic, with the added restriction the time advance functions are rational numbered.

The finite reachability graph can be used to determine liveliness or for safety checking. The FD-DEVS models are less expressive however they can still be used to model a variety of control problems. The work is based on time-zone equivalence described by Dill [87], where using a Buchi automaton Dill shows that timing properties of a concurrent automaton can be verified to prove speed independent specification can be met. Those specifications that do not depend upon the execution speed of and delays between the components.

Utilizing DEVS and RTDEVS the authors in [88] show safety-critical applications, a railroad crossing in this case, can be implemented using the method described by Zeigler

et al. [15]. An algorithm for the generation of reachability graphs of the model is proposed that can be used for the safety analysis of the system.

While we have used directed weighted graphs to represent our atomic and coupled model, the use of Higraphs to draw and model atomic DEVS phase transition is proposed by [89]. The phase transitions consist of states as vertex and transitions as the edges. The edges in this case are complex code for transitions and the conditions required for the transitions. Higraphs, offer the benefit that the area of the graph is used to represent the cartesian product, set enclosure, and exclusion, this allows for a reduction in the complexity of the diagrams.

DEVS remains underutilized for supervisory control of hybrid systems. More work needs to be done to formally define methods to gauge the observability, permissiveness, deadlock analysis, reachability and other properties related to supervisory control. Without this the only method to determine these properties is to perform vigorous testing through simulations. As we have discussed for complex systems this is not feasible. State space explosion remains a major challenge in DEVS for the purpose of supervisory control and we attempt to tackle this problem in the thesis.

2.5.2. The Use of DEVS for Supervisory Control

In this section, we discuss the methodology devised by Zeigler et.al [15] in detail. The major steps in the methodology are shown in Figure 4.

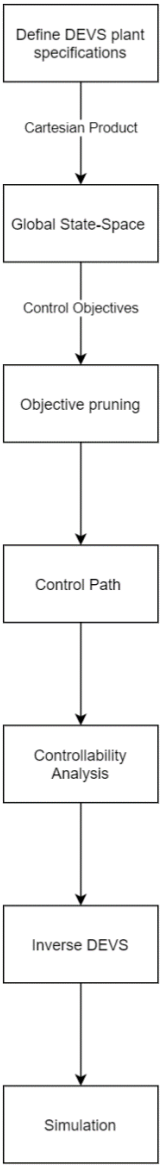

Figure 4. Original methodology for obtaining a DEC [15].

The first step involves defining a DEVS model of a system. This system can be continuous or discrete with the condition that the input and output trajectories are piece-wise constant for continuous event dynamical systems. Plants are generally represented by Differential Equation Specified Systems (DESS), these systems have piece-wise constant input, output trajectories and therefore can be represented by the DEVS formalism [66].

For the given DEVS coupled model of a plant, we take the cartesian product of the coupled model. A detailed explanation of the cartesian product as applied in our case for comparison with this methodology is given in Chapter 4. Briefly, the cartesian product is a non-commutative operation that can be applied between two sets. In this case, the two sets are two components atomic or coupled model of the top coupled model, represented by the set of states and transitions. The result of this is the set of all possible combinations of states from the two state sets and the set of allowed transitions between the two state sets. This resulting Global State Space (GSS) contains every path the model can take given starting conditions. The state space explosion issue occurs at this step of generating the GSS. After this step the GSS is only reduced and simplified to obtain the controller. In the next step, the GSS is reduced by applying the control objectives. Two types of control objectives are applied in the objective pruning step. One all unwanted states are removed from the GSS and two, all the unwanted transitions are removed from the GSS. The purpose of this step is to find a control path in the GSS that if the model follows will fulfill the objectives set by the modeler. By removing all unwanted states and transitions the model can only follow the control loop required by the modeler. Before moving to the next step, the algorithm checks if a closed path exists between two states in the GSS. This control path is the desired control loop in the GSS.

Before the controller can be extracted from this control path. Controllability analysis needs to be performed on the control path. The purpose here is to ensure the control path does not have any transitions or states that may, given the right conditions, move the model away from the desired path. Three types of controllability are defined in this method.

1. Weakly controllable paths.

2. Strongly controllable paths.

3. Very strongly controllable paths.

Weakly controllable paths are those where for a given state 'S_i' on the control path, such that 'S_{i+1}' is the next state on the control path. The path from 'S_i' consists of internal and external transitions to the state 'S_{i+1}', i.e., there are states between 'S_i' and 'S_{i+1}' that are not on the control path. The transitions from these states not on the path are not guaranteed to move the current state back onto the control path and can further move the model away from the control path.

Strongly controllable paths are those for which two states 'S_i' and 'S_{i+1}' have direct internal or external transitions, with no states between them that aren't on the path. The model can remain active for all transitions. This means the internal transition can move the model away from the control path, however, in this case, an external input exists that the controller can schedule that will force the model back to the control path.

Very strongly controllable paths are like strongly controllable paths with the added requirement that internal transitions passivate the path. This, unlike the strongly controlled path case, completely prevents the internal transition function from moving the model away from the control path and the controller does not need to issue an external input to force the model back onto the path.

The purpose of this step is to identify any part of the desired control path that can potentially cause the model to stray. If such a weakly controllable path is found, the modeler can add more sensors or change design parameters so that the path becomes strongly controllable. If a path is very strongly controllable, further state reduction is

applied to reduce the number of states and transitions from the controller to obtain a small number of states and transitions. The exact method for state reduction is not specified however for smaller applications state reduction can be carried out manually. State reduction provides little benefit at this step as no further analysis needs to be done on this path. State reduction is more useful before the cartesian product is applied as it then massively reduces the number of states and as a result the generated GSS is considerably less complex.. However, it can still make the controller smaller and easier to implement for simulation later.

Inverse DEVS transform on this control path can now be applied to give us the controller for this model. The Inverse DEVS operation is defined as, given a plant DEVS coupled model 'M', and input events 'E_{INPUT}' and output events 'E_{OUTPUT}'. The inverse DEVS transform obtains a coupled model 'I'. Where 'I' has the properties:

1. S (M) → S (I). The states of the original DEVS model have a one-to-one correlation with the states of the inverse model. We have state bijection.
2. $I_X = \{E_{INPUT} \mid E_{INPUT} \in M_Y\}$, $I_Y = \{E_{INPUT} \mid E_{OUTPUT} \in M_X\}$.

   The input and output event sets are the same and the input event set of one is the output event set of the other and vice versa.
3. $\delta I_{INT} = \{ (q, E_{OUTPUT}, r) \mid (q, E_{INPUT}, r) \in \delta M_{EXT} \}$

   $\delta I_{EXT} = \{ (q, E_{INPUT}, r) \mid (q, E_{OUTPUT}, r) \in \delta M_{INT} \}$.

   The internal transitions of model M become the external transitions of I and the external transitions of M become the internal transitions of I.

The Inverse DEVS transform in essence inverts the transitions and I/O ports while keeping the states the same. For every internal transition and output event in the model

there is an external transition and input event in the inverse model, in this case, our DES controller, and vice versa. After the application of the Inverse DEVS on our control path we obtain the controller for our model.

Since the controller and the model are both DEVS models they can be coupled together. The internal ports of the model are coupled with the corresponding inverse output ports of the controller and vice versa. Through a simulation in software, the controller can now be tested for proper functioning. From this testing, any required adjustments can be made to the model to make it more accurate, thus ensuring a proper controller is obtained. In our methodology in Chapter 4, we replace the cartesian product with the approximate method for creating a state space and describe the methodology for directed weighted graphs. In addition, we also execute the model on hardware using DEMES.

Chapter 3: Problem Statement

In this chapter we describe the problem that we solve in this thesis considering the works done in the literature as discussed in chapter 2 and the scope of this thesis.

In this work, two problems are tackled. The first problem is a practical methodology for the implementation of a supervisory controller using DEVS as a Modeling and Simulation (M&S) methodology. Our method involves the use of an approximate algorithm to obtain a global state-space by ignoring timing information, from which controller can be obtained. Using the RT-DEVS formalism the models can be tested on hardware surrogates. The objective here is to reduce the global state-space complexity so that practical implementations are possible. The state space complexity is dependent upon the method used to generate the GSS and by changing the method to generate the GSS, the GSS can be reduced.

The second problem is that of adding human interaction to a supervisory control system. Particularly allowing a human to change the setpoints of a system without introducing extra transitions that would add complexity to the design of the supervisory controller. We use the hybrid layer to mask the human user to the supervisory controller and as a result keep the controller design simple. The objective here is to introduce human interaction to a system and keep the supervisory controller design simple and utilise the methodology described in this thesis without modifications.

We restrict the scope of this thesis to defining and demonstration a practical methodology for the implementation of SCT and introducing human interaction to a system. We do not derive formal definitions or perform formal proof to determine the

properties of the controller, like reachability, liveliness, and observability, as our approach is based on previous works with a sound basis in formal languages and therefore carries its properties from those works. The reduction in the GSS does not invalidate the method as any unreachable state arising due the reduced GSS are impossible to occur in simulations. Therefore, they do not affect the simulation and can be safely removed and in the original methodology are removed at a later step. The use of the hybrid layer is restricted to introduce human interaction with the system and does not delve deeper into fine control of actuators or other applications of hybrid layer.

Our work is based on the work of Zeigler et al. [15] as described in Section 2. The use of cartesian product to obtain the global state-space of a DEVS model leads to a computationally prohibitive state-space on which further analysis is not practically impossible. Using the approximation, the global state-space can be reduced to a manageable size.

We also found as discussed in chapter 2, the work done in relation to DEVS as a M&S methodology for SCT is lacking. DEVS has been used to model FSMs and other automata to utilise supervisory control as defined in other methodologies, or it has been used to define intelligent computing models that are primarily utilized in autonomous robotics. The use of RT-DEVS as a method to have model continuity across software and hardware simulations in relation to SCT has not been studied.

The use of hybrid layer to allow human interaction with the system has to our knowledge not been explored in the literature. Previous works as discussed in the chapter 2, model humans as a sensor in the DES, this intuitively simple approach is useful for many applications. However, this approach introduces complexity in the extraction of a

supervisory controller and may not be applicable to all applications. We define a component in the hybrid layer that allows a human controller to interact with the system. The component is designed in a way to encapsulate human interaction with the system so that the number of transitions presented to the supervisor is reduced and the system appears as a DES.

In the following chapters we will present our modified methodology and the approximate algorithm to be used to extract a controller from a DEVS model of a plant and the limitations of our approach. The definition of the component to incorporate human interaction with the system is also discussed. We demonstrate the practicality of our approach by implementing the case study of a smart room controller using the methodology described and simulating the models using Cadmium simulation environment in a software and an embedded hardware environment.

Chapter 4: Supervisory Control with DEVS

The operating principle of a supervisory controller is the disabling of events or transitions of a plant model as is discussed in chapter 2. For this reason, the generation of the Global State Space (GSS) is an important step in the process of synthesizing a supervisory controller. The GSS contains all the events and states that a given plant model can have and the supervisor can then decide what events to disable from the GSS. For DEVS-based supervisory control, the cartesian product has been used to generate the GSS as is discussed in chapter 2 and the problems associated with it were reviewed in chapter 3. In this chapter, we introduce our alternative approximate method to generate the GSS, discussing its benefits and limitations.

DEVS coupled models contain information about the structure of the plant that can be exploited to eliminate events and their associated transitions from the GSS that are unable to occur due to the nature of the plant. The cartesian product ignores this structure of the plant and therefore produces a GSS that contains transitions to states that are not possible. This leads to a bloated GSS that can be reduced if the structure of the plant model is considered during the generation process. The approximations method addresses this limitation and incorporates information about the structure of the plant from the coupled model to generate a GSS that only contains events that can occur.

This chapter also discusses a method to introduce human interaction in the system using the hybrid layer. We discuss how a human changing the set-points of a plant introduces transitions and therefore added complexity to the plant model and how the hybrid layer can be introduced to reduce this complexity. The necessary conditions to

introduce the hybrid layer to the plant model and its drawbacks are also discussed. We first provide an overview of the modified process to obtain the Discrete Event Controller (DEC), followed by a brief description of the cartesian product and a detailed description of the approximate method to obtain the GSS. The rest of the steps in the process are then explained as applied in our implementation. Finally, the introduction of human interaction to the system using the hybrid layer is described and discussed at length.

4.1 Supervisory Control Methodology Overview

Obtaining a supervisory controller for a DEVS plant model follows a series of steps, each of which achieves a specific goal. In chapter 2, we outlined the original methodology defined by Zeigler et al. [15]. Based on this methodology, Figure 5 shows our modified process. The most notable change in the methodology is how the GSS is obtained. Instead of using the cartesian product as in the original method, we use the approximate method to generate the GSS. This change reduces the complexity of the GSS at the point of generation, by maintaining the structural information of the coupled plant model. The second change is the addition of testing the plant models and synthesized controllers on an embedded hardware platform using the RT-DEVS formalism. This added step increases confidence in the ability of the controller to perform correctly in real-time environments. The methodology is explained and implemented for a directed weighted graph structure representing the DEVS plant model.

4.1.1 A Modified Methodology to Obtain a DEVS Controller

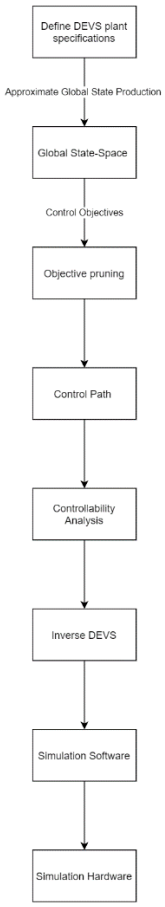

Figure 5. Modified methodology for obtaining the DEC.

The first step in the procedure is to define a DEVS model of the plant. As discussed in chapter 2, DEVS can model both hybrid and DES systems. Once a DEVS model for the plant is defined, we can apply our approximate method to get the GSS of the plant DEVS model. The GSS in our case is a weighted directed graph that contains information about how a plant model can evolve from a given 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 of the graph. The weight of the edges defines the transition type, port, message, and time advance duration to be set after the transition.

The objectives can now be specified and applied to the GSS. The control objectives define the states and transitions that are unwanted in the plant: the supervisory controller can only disable events and therefore the control objectives include unwanted states and transitions that need to be removed (i.e., disabled). These represent states and transitions that violate the control goal of the plant by either being illegal states that the modeler does not want the plant to enter or transitions that move the plant away from the desired control loop. By applying the control objectives, we remove these unwanted states and transitions from the GSS. In addition, the control objective can also specify if a path exists between two states. These states define the start and endpoints of the control loop the plant is expected to take. If a path exists between two states specified by the modeler, then we can conclude a control loop exists in our reduced GSS. We then apply controllability analysis on our reduced GSS. The result of this analysis shows any transitions in the state-space that has the potential to move the controller away from the desired control path and onto an uncontrollable path. These transitions are referred to as weak transitions.

If our reduced GSS does not contain any weak transitions that can disrupt the control loop, we can apply the Inverse DEVS transformation on the GSS and obtain a DEC for our plant that would satisfy the control objectives. The DEC is analyzed, and we remove any transitions and states that are unused to reduce the number of transitions and states.

The DEC and plant model can now be coupled together and simulated for testing. Using the Cadmium tool and RT-DEVS formalism the DEC and plant model are modeled and simulated in software. We can provide test inputs to observe the behavior of the plant and verify proper functioning. If the observed behavior does not match the expected outcome, we can make the necessary changes to the plant or controller to fix the issue. The

models can then be ported over to hardware and simulated in real-time. Because we use the RT-DEVS formalism the models maintain continuity between software and hardware implementations. The tests can be repeated to observe the behavior in a real-time environment. These tests can show the obtained controllers can be expected to always keep the plant on the control path. Using this procedure, we have obtained and tested a DEVS-based supervisory controller from a DEVS model of a plant.

4.2 Generating the GSS

We use a directed graph to represent the DEVS atomic plant models, as shown in Figure 6. Here, we show two states and a transition linking the states. A solid line represents external transition, and the dashed line represents an internal transition [68].

Figure 6. Graphical representation of atomic models as states and transitions.

The weighted edge stores information about the transition including the transition type, the time advance, the input/output port, and the message. The transition type specifies if the transition is internal or external. The time advance here represents what the time advance is set to after the state change occurs. It is simplified to either a finite transition time or an infinite transition time which represents the passivation of the model. In this representation, zero-time advance is not allowed. Transitions with zero-time advance introduce complications in the modeling and simulation of the plant and therefore obtaining a controller for the plant. For instance, an internal transition with zero-time advance has the potential to start a non-terminating loop of the zero-time advances. The simulation can

no longer proceed as the time does not advance and no further scheduled events can occur that would break the loop. We avoid these complications by not allowing zero-time advance to exist in the GSS [66].

The transition type specifies the port type that is represented by the input/output port field. In the case of external transition, the port represents an input port from which a message is received, and in the case of the internal transition, the port represents an output port from which an output is produced. It is important to point out the output occurs because of the output function, however, both functions theoretically execute at the same time and therefore we store the output information in the weighted edge for internal transitions.

4.2.1 The Cartesian Product

To compare our method with the original methodology, we calculate the GSS of the plant model using the cartesian product. We use the NetworkX [90] python package to compute the model. NetworkX stores the atomic models as a directed multigraph. The procedure involves taking the directed graph of each atomic model and then computing the cartesian product with the previous product. For example, consider three graphs G, H, and J, to compute the cartesian product C, we compute the product K = G x H, and then compute the product C = K x J. This can be repeated for N instances of atomic models. If there is more than one instance of an atomic model, then each instance is computed with the product. Consider the case of the two atomic models G and H shown in Figure 7. V(G) and V(H) are the set of vertices of the two graphs, with V(G) = $\{u_1, u_2,\ldots, u_n\}$, V(H) = $\{v_1, v_2 ,\ldots, v_n\}$. E(G) and E(H) represent the edges of the model.

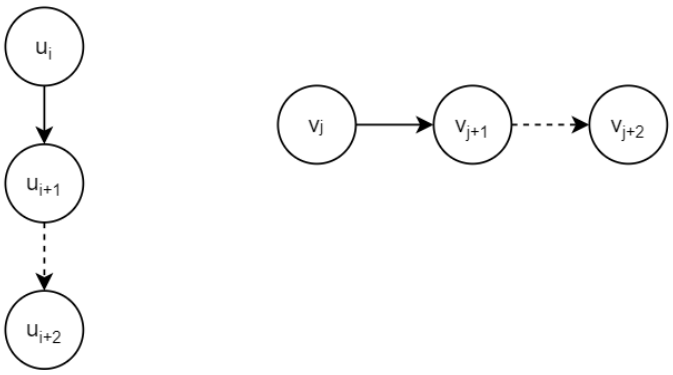

Figure 7. Graphical representation of atomic models G and H.

To compute the product, we form the set of ordered pairs of V(G) and V(H). This set of ordered pairs forms the combined states of the global state-space. The ordered pairs are formed by the following procedure.

- For each vertex u_i form pairs with vertex v_j where:
    - j is the index of vertices of V(H) from 1 to n, and n is the total number of vertices of V(H).

Taking the vertex u_i shown in Figure 7, we form the ordered pairs (u_i,v_j), (u_i,v_{j+1}),(u_i,v_{j+2}) .We repeat for u_{i+1} and u_{i+2} to get a set of 9 ordered pairs as shown in Figure 8. They are called ordered pairs because the order of appearance is important and (u_i, v_j) != (v_j, u_i).

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.

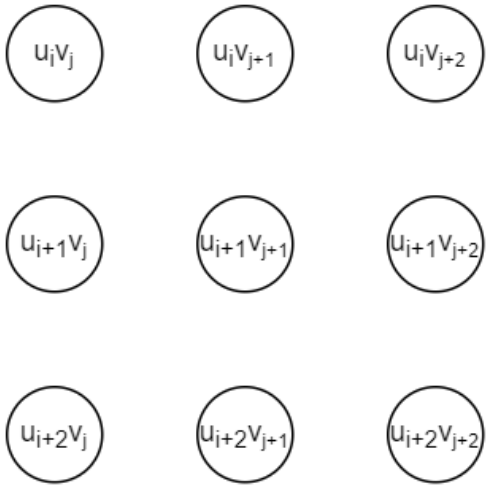

Figure 8. The ordered pair sets of two graphs V(G), V(H).

This means that two combined states in the GSS are adjacent if one state 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 as shown in Figure 9a, and Figure 9b. The directionality of the edges is preserved. The weights of the edges are also preserved, this can be done as the ordered pairs differ by only one state and this state change is from one atomic model. Therefore, the original weights still represent the transition condition and can be copied without modification to the edge. Two states can have multiple edges if the weights are different, in this case, the combined states also have multiple edges between them. The edges in Figure 9a are due to the edges of graph G. Notice the V(H) do not change between the connected nodes and only the V(G) change. If two vertices are adjacent, we can add the edge between the two states, this transition will cause the system to move to the other combined state.

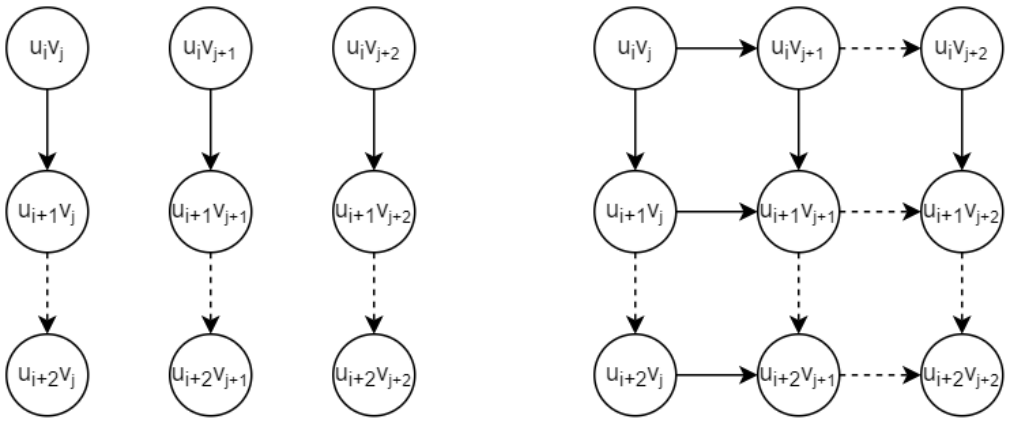

Figure 9 a) Adjacent vertices due to edges of G. 5b) Adjacent vertices due to edges of H.

The cartesian product and the resultant product have some properties of these two are of interest. First, the use of ordered pairs implies that edges and therefore transitions can only exist between nodes that have only one state change in their combined states. Second, the cartesian product is not commutative. We select the order of operations based on the *select* function of the coupled model. We choose to use the *select* function to fix the order of combined states.

With the description of the cartesian product provided in this section, we can see it does not utilize the structural information present in the DEVS coupled model of the plant. This leads to the addition of extra transitions that are not possible to ever occur and can cause the state space to explode. Our approximate method uses this information to attempt to reduce the number of transitions.

Our method needs to share the properties of the cartesian product and therefore it must not be commutative and must only allow transitions between combined states that have one state change between them. The *select* function is utilized to set the order of

combined states and maintain parity with the results due to the cartesian product. This is to ensure the rest of the methodology stays consistent with the original methodology.

4.2.2 The Approximate Method

Our approximate method uses the structural information of the DEVS coupled model to generate the GSS. The basic principle is recognizing that the possible transitions for a given coupled model “CM” are determined by the EIC, IC, and EOC definitions in the coupled model, and by analyzing these transitions we can generate the GSS. The coupled model of a plant contains couplings that link input and output ports. If coupling to an input or output port of an atomic model does not exist in the coupled model, events will not reach or be outputted from the atomic model. In consequence, certain transitions in the atomic models will not have an effect on the plant and can be ignored in the GSS.

In the case of the output function, the EIC or IC must define couplings with the output ports of the atomic models. The output ports are defined in the atomic model and are used by the output function to direct the output events. The generated output events are only accessible through the output ports designated by the output function. Otherwise, the output events generated will not be able to reach another component in the model or be outputted from the coupled model. Thus, it does not affect the model or components connected to the model. Figure 10 shows two coupled models where the model structure includes some components in which the output ports are not in the IC or EOC sets.

The model in Figure 10a, represents a case with no IC between two atomic models. Model ‘A’ cannot influence atomic model ‘B’, as there is no coupling between them and therefore the output of ‘A’ cannot reach ‘B’ and we can ignore it for the GSS.

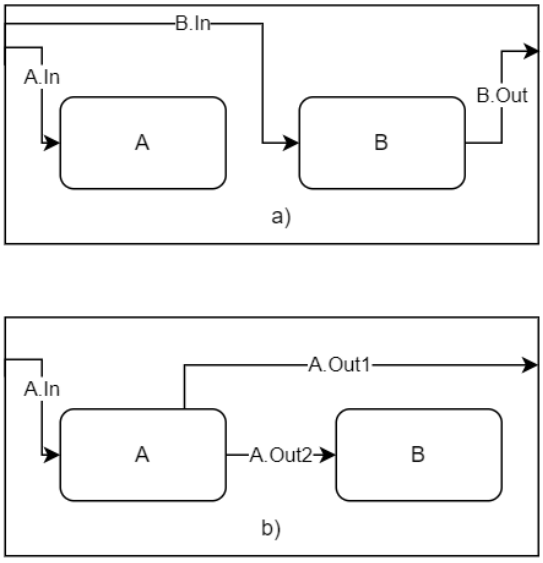

Figure 10. Two coupled models: a) Coupled model with no IC between 'A' and 'B'; b) no EOC with ports of 'B'.

Model 'B' can only be influenced by the EIC through the 'B.In' port and only the output function of 'B' can provide an output of the coupled model. Therefore, we include it by adding the associated internal transition to the GSS. This is done by finding the two combined states that differ by the states that exist before and after the internal transition is executed. Section 4.2.2.5 further explains how the two combined states are chosen.

The plant coupled model generates information about the model through the output function of atomic 'B' the EOC with the 'B.Out' port. For the coupled model in Figure 10b, the output function associated with the port 'A.Out2' influences the model 'B'. However, model B has no EOC and thus its output function cannot provide information about the coupled model to the outside, in this case, only the output function associated with the 'A.Out1' port can provide the information and therefore the internal transitions of B can be ignored and only the internal transitions of A are considered. In the case of an

external transition, the input events will not be able to reach the atomic model as there are no couplings from either EIC or IC.

Figure 11 shows two examples of coupled models where one component has no EIC defined and in the second the two components are not internally coupled.

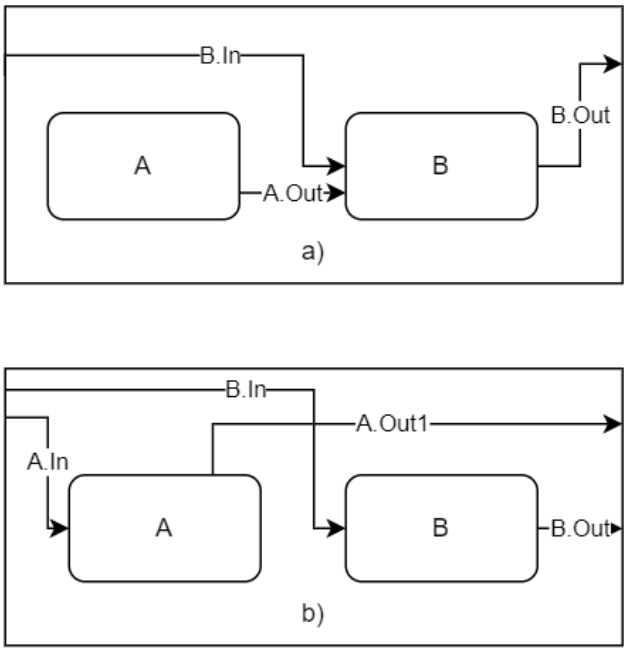

Figure 11. Coupled model a) no EIC to 'A' and b) no IC between 'A' and 'B'.

In Figure 11a, there are no coupled inputs to the atomic model A. The external transitions of A will never execute and can therefore be ignored in the GSS. The external transitions of 'B' are triggered due to the EIC with port 'B.In' and IC 'A.Out' and these external transitions are included in the GSS. In Figure 11b, the coupled model features no IC, therefore the external transitions of 'B' are only triggered by the EIC with 'B.In'. Any external transitions not associated with 'B.In' can be ignored in our GSS.

We can see that the couplings defined for a model form its structure and this structure restricts the input events the model can react to, and the output events the model

will generate. We utilize this information to only add the transitions that affect the model. 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.

4.2.2.1 The Approximate Method Algorithm

We now present the algorithm for generating the GSS 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

Select a component name 'd' from D:

for 'd' select corresponding instance m from Mi:
{

    If d in EIC.component:
    {
        // Section 4.2.2.2: Analyzing the External Input Coupling
        for d.external_transition.port in m.port:
        {
            if d.external_transition.input_type == EIC.port.InputType
            {
                Add external_transition to global state space.
            }
        }
    }

    if d in IC.first_component:
    {
        // Section 4.2.2.3 Analyzing the Internal Couplings
        for d.internal_transition.port in IC.out_port:
        {
            Add internal transition to global state space.
        }
        d2 = IC.second_component
        for d2.external_transition.port in IC.in_port:
        {
            Add external transition to global state space.
        }
    }

    if d in EOC.component:
    {
        // Section 4.2.2.4 Analyzing the External Output Coupling
        for d.internal_transition.port in EOC.outport:
        {
            Add internal transition to global state space.
        }
    }
}
```

The algorithm first selects a component ‘d’ from the coupled model. Each atomic model in the coupled model can be reused and have different couplings. To differentiate between different components that are instances of the same model type, we name each component uniquely in set D. Each component ‘d’ from set D represents an atomic or a coupled model and its unique couplings in the top model. The top model is the outermost coupled model in the hierarchy. The coupled models contained in the top model can be similarly composed of atomic and coupled models. The method defined here requires that the top model must only be composed of atomic models. Otherwise, the coupled models need to be recursively searched until a coupled model with only atomic components is found. This coupled model is at the lowest level in the hierarchy. For this coupled model the GSS can be computed. This GSS and the coupled model’s couplings in the next higher coupled model in the hierarchy is the same as an atomic model and its state space and couplings. We can therefore move up the hierarchy until we reach the top model and compute the GSS for the top model. In the scope of this thesis, we only implement the method for a top model with only atomic components.

For a component ‘d’ its associated atomic model is used to obtain the transitions and their weights. The couplings of the coupled model with component ‘d’ inform the algorithm of what transitions to consider and what to ignore.

In Figure 12, the coupled model consists of atomic model’s ‘A’, ‘B’, and ‘C’. The component names of ‘d’ are specified by the dot after the atomic model type, so we have components named ‘1’, ‘2’, ‘3’ and ‘4’.

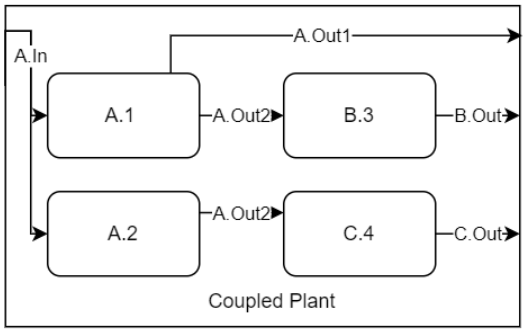

Figure 12. Coupled model with multiple instances of atomic model 'A'.

There are two instances of the atomic model 'A', named '1' and '2' (each of them having different couplings). Component '1' has links in the EIC and EOC of the coupled model and a link in the IC with the component '3', which is of the atomic model type 'B'. Component '2' on the other hand has only an EIC and an IC with the component '4', which is of the atomic model type C.

An atomic model may have parameter variables defined in it, such as threshold values, sensitivity, output values, etc., and therefore different components of the same atomic type may use different parameter values. However, the behavior of the atomic model in terms of states and transitions does not change due to differences in parameters. If a parameter changes the condition of a transition then that affects when the transition occur, however, it cannot alter the state change caused by the transition. If a parameter alters the state change caused by the transition then each separate state change will be represented by a separate transition. These parameters can be safely ignored, and the atomic model is defined completely in terms of only the states and transitions and thus allowing the reuse of the atomic model without alteration for each unique named component 'd' of the same atomic model type.

4.2.2.2 Analyzing the External Input Coupling

We search the EIC and check if it contains any entry with the component name ‘d’ (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 the component name 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 external transition to the GSS between the combined states that have the corresponding state change between them. In other words, between all the combined states that contain the starting state of the transition and the ending state of the transition. The internal transition that may be triggered after this external transition is not considered. As we reasoned earlier in this section, for an output function associated with an internal transition to influence the model it must be present in either the IC or EOC. Otherwise, the internal transition will not affect the model.

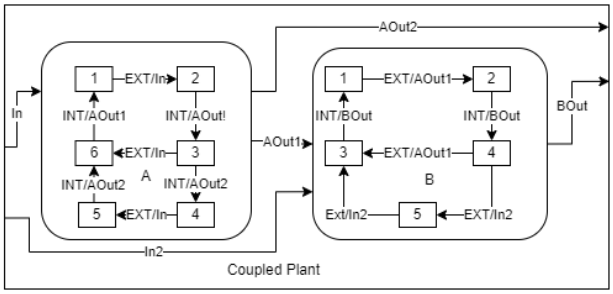

Figure 13. Coupled model with atomic model’s ‘A’ and ‘B’ and their states, transitions, and couplings.

Figure 13 shows a coupled model with two atomics ‘A’ with the states ‘1’ to ‘6’ and ‘B’ with states ‘1’ to ‘5’ (represented by rectangles). The transitions between the states are shown by marked directional arrows with the ‘INT’ referring to internal transition and

‘EXT’ referring to external transitions, the name after the ‘/’ refers to the port associated with the output or input event. There, are two EICs in this coupled model, one with port ‘In’ coupled with atomic ‘A’ and port ‘In2’ coupled with atomic ‘B’. For the atomic ‘A’ the external transitions that are affected by these couplings are ‘1’ → ‘2’, ‘3’ → ‘6’ and ‘4’ → ‘5’. These are added in the GSS between combined states that differ by these states, for example, the transition ‘1’ → ‘2’ is added as a directed edge from the combined states that have the atomic ‘A’ set as ‘1’ to the combined states that has the atomic ‘A’ set as ‘2’. Similarly, the external transitions of atomic ‘B’ affected by the EIC are ‘4’ → ‘5’ and ‘5’ → ‘3’ and these are added to the GSS. The internal transitions of atomic ‘A’ or ‘B’ that are triggered after the external transition are ignored and are considered in later steps when the IC or EOC are analyzed. Finally, the external transitions of atomic ‘B’ that are triggered by input from port ‘AOut’ are ignored as these are due to the IC between ‘A’ and ‘B’.

4.2.2.3 Analyzing the Internal Couplings

Next, we search the list of Internal Couplings. An IC can be used by two transitions, an internal and an external transition across two components. The first is an internal transition that has an associated output function that generates an output for this state, this internal transition is from the first component in the IC list. The output generated is sent to the coupled second component in the coupled model as an input. This input then triggers an external transition in the second component.

The method we employ here involves a two-step process. In the first step, we match the component ‘d’ with the first component from the IC list. If a match is found, the internal transitions are filtered, and the output functions associated with it are matched with the output port and type required by the second component of the IC. The internal transition is

then added between the two combined states of the model that have the state change specified by the internal transition between them. We take note of the second combined state as this will be used to filter out states for the next step.

In the second step, we take the second component for this IC entry and find the external transition in the second component that has the input port and type that matches the output of the first component. This external transition is then added between the relevant combined states that contain the state change between them, with an added filtering step. We realize that the internal transition moves the system from one combined state set ‘A’ to combined state set ‘B’. The external transition can only occur from the combined state set ‘B’ as the GSS must be in that state. This is due to the fact, theoretically, after a time advance period is complete, the internal transition and output function occur simultaneously. The GSS at this instance of time must then be in state set ‘B’ when the external transition is executed in the second component in the IC. Furthermore, the external transition also occurs instantaneously and therefore the GSS cannot move away from state set ‘B’ as no other events can occur at this point. This filtering step ensures the transitions in the GSS due to the IC only exist, between combined states that can exist at that instant of time.

The case of the IC can be better explained by referring to the coupled model shown in Figure 13. The internal transitions of atomic ‘A’, and the associated output functions involved with the IC coupling between the atomics ‘A’ and ‘B’ are the transitions ‘2’ → ‘3’ and ‘6’ → ‘1’ from atomic ‘A’. These are added to the GSS, and the atomic ‘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, the transitions ‘1’ → ‘2’ and ‘4’ → ‘3’ match the

port ‘AOut’, this port is associated with the output of atomic ‘A’ and is added to the GSS because of the IC. The external transitions due to ports ‘In’ and ‘In2’ are considered in the previous step where the EICs are analyzed. The internal transitions and the output function associated with port ‘AOut2’, and ‘BOut’ are considered in the next step, where EOCs are analyzed.

It is important to mention that if a component exists in the coupled model with a zero-time advance transition, it can potentially occur at the same instant as the internal transition, output function, and external transition function of the IC and move the coupled model away from the state set ‘B’. This is another reason within the scope of this thesis we avoid zero-time advance in the plant models and we do not allow it in our approach.

4.2.2.4 Analyzing the External Output Coupling

Finally, we then move on to the EOC list. The EOC follows similar steps to the EIC, except that in this case, we match the output function associated with the internal transition of the component ‘d’ to the output port and type specified in the EOC. We note that the output function can produce an output without a change of state due to the internal transition function. However, such an output creates self-loops in GSS. If these self-loops require a controlling action the supervisory controller cannot take corrective action. This is because the state of the model does not change, therefore, the state of the controller does not change and thus it cannot generate a unique output for such self-loops.

There are a vast number of scenarios where self-loops can occur and each scenario will require its workaround in the controller, making a generalized approach improbable. We, therefore, do not consider the case where there are self-loops in the model that require

a controlling action without the state changing. One potential method to resolve the issue is by adding additional states with an internal transition that removes self-loops and allows the controller to register a state change and produce a unique output if required. The price to pay for this approach is increased complexity in state-space due to added states and therefore transitions.

Referring once again to the example coupled model shown in Figure 13, the EOC only considers the output function that is associated with the coupled ports "AOut2" and "BOut" and the associated internal transition. This means this step only adds the transitions '3' → '4' and '5' → '6' from 'A' and '2' → '4' and '3' → '1' from 'B'. All the other internal transitions are ignored by this step.

4.2.2.5 Simplifying the GSS By Considering Only One State Change

We can reduce the size of the GSS if we recognize that each internal or external transition can only make one state change in the combined state. For example, suppose we have two components 'A' and 'B' with transitions that change the state from 'a' to 'b', and '1' to '2' respectively. If we start with the combined state {1, a} and a transition due to component 'A' occurs it will cause the combined state to change from {1, a} → {1, b}. The component 'B' cannot change its state as its transition has not been triggered and must therefore be in state '1'. In the event, that both components have their transitions triggered simultaneously, the *select* function establishes the order in which the components are allowed to execute the transition. This forces the transition of one component to occur before the other and the combined state change would then be one of the combinations of combined states from {1, a} to {2, b}. This applies to more than two components having their transitions occur

simultaneously, as the *select* function ensures there is a strict execution order for the transitions from multiple components.

Therefore, we can add the restriction that transitions can only be added in the GSS between combined states with no more than one state change between them. For example, no transition can be added between the combined states {1, a} and {2, b} as there are two state changes between them. By restricting the transitions allowed in such a manner, we reduce the total number of transitions in the GSS.

By incorporating this in the approximate method and utilizing the structure of the coupled model we can prevent the state space from exploding during the calculation of the GSS. The remaining steps are dependent upon the GSS and if the GSS is small then the remaining steps become easier to carry out.

Analyzing concurrent transitions and states can reduce the GSS by identifying impossible transitions and states that cannot occur physically. It can also identify multiple transitions that can be combined into one. However, such an analysis is difficult in practice to do formally and in a generalized manner. The modeler can conduct model inspections of the GSS and identify these issues. Similarly, we manually identify multiple transitions that can be combined into one to resolve the issue of "missing transitions" that we will discuss next.

4.2.2.6 Missing Transitions

One issue with our method is that in some cases during model simulation, the transitions involved in an IC can have "missing transitions". This is due to two reasons. The first is the restriction we place that only one atomic can change its state at a time. Secondly, we

do not consider timing information and approximate it as either a finite or infinite transition.

The “missing transitions” problem occurs when the first of two connected atomic component passivate the component. The passivation of the component is reflected as passivation of the controller as the inverse DEVS transformation makes the internal transition an external transition. The controller should not be passivated as the external transition of the second component is independent of the passivation of the first component. Therefore, this external transition is executed, and the GSS moves onto the next state regardless of the passivation of the first atomic. The controller, however, is passivated due to the external transition and cannot execute its internal transition to keep up with the current state of the GSS. The GSS does not contain a transition that would take it from the combined state before the internal transition to the combined state after the external transition, as such a transition would require two state changes for one transition, The approximation of time in the GSS does not completely capture the time independence of the components from each other. The controller, therefore, does not have a transition that would let it skip the passivation and move to the current state of the GSS.

Only 2 time advances are considered: either finite or infinite time. A finite time advance is one where the time advance function takes a non-zero finite real value. In this case, the model waits for a finite duration and then executes the internal transition function. The model is said to be ‘active’ for a finite time advance. For an infinite time advance, the time advance function takes an ‘infinite’ value. The model in this case will wait indefinitely and will not execute its internal transition function. The model is said to be ‘passive’ and

will remain in this state until an external transition causes the time advance function to take a finite value again.

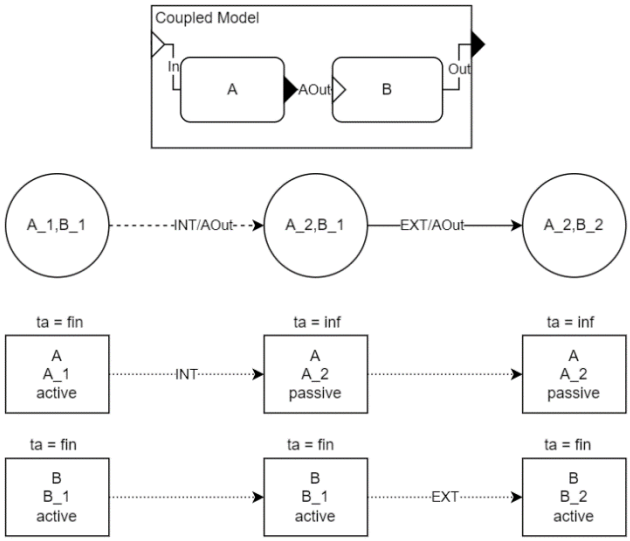

Figure 14. Coupled model composed of atomics models 'A' and 'B' and their combined state transitions (circles represent the states; rectangles represent the atomic model current state)

Consider the two atomic models 'A' and 'B' in Figure 14. The diagrams below the coupled model show the state change over time. The combined states are represented by the circles and are connected by transitions. The rectangles show the model's name, then its current state and finally its active or passive status. The current time advance of the atomic model is written on top of the model and is represented by 'ta = fin' or 'ta= inf', for finite and infinite advance respectively. The dotted lines between the atomic models show the advancement of time in the simulation and if a transition occurred in the atomic model during that time. The sequence of events that occur in Figure 10 is as follow:

- After a finite time advance in atomic model 'A' an internal transition is executed in component 'A' that causes the state to change from 'A_1' to

‘A_2’. The time advance is updated and set to infinity. This causes the atomic ‘A’ to passivate.

- The internal coupling between ‘A’ and ‘B’ as shown in Figure 10 causes the atomic model ‘B’ to have an external transition with the state change ‘B_1’ to ‘B_2’. The time advance of atomic ‘B’ is set to a finite advance.

The complete transition sequence that occurs in the coupled model is then as follows, an internal transition from ‘A_1, B_1’ → ‘A_2, B_1’ and then an external transition from ‘A_2, B_1’ → ‘A_2, B_2’ as shown in Figure 10. The result is that atomic model ‘A’ is passivated, and the atomic model ‘B’ is active. We apply the Inverse DEVS transformation to our coupled model and obtain a controller. The internal transition becomes external transitions and the external transitions become internal transitions, as seen in Figure 15.

The resultant controller is coupled with the plant model as shown in Figure 15. The state transitions are represented as in Figure 14. In this scenario, the state change ‘A_1, B_1’ → ‘A_2, B_1’ is accompanied by the time advance being set to infinity and passivates our controller. This passivation of the controller prevents any internal transitions from executing and therefore the state change ‘A_2, B_1’ → ‘A_2, B_2’ cannot occur in the controller, as shown in Figure 15 this transition is crossed out.

We know that this transition ‘A_2, B_1’ → ‘A_2, B_2’ occurs in the plant model and our actual GSS has moved to ‘A_2, B_2’ due to the second component ‘B’ executing its external transition, as shown in Figure 14.

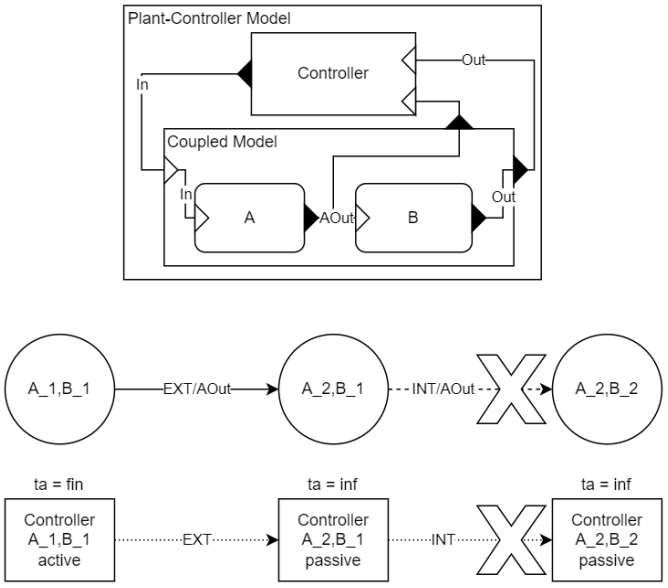

Figure 15. Coupled model of the plant and controller model. State transitions of the controller displayed below the coupled model diagram. The circles represent the states, and the rectangles represent the atomic model and their current mode, active or passive.

This is how transitions can be "missed" in our approximate method. The transition exists as an internal transition of the controller; however, it becomes inexecutable as the controller is passivated. This is a problem that can occur with the cartesian product approach as well provided there are internal couplings whose state changes passivate the first component.

To resolve this, we can analyze our controller and plant models to easily identify the transitions that are involved in an IC, whose associated time advance passivates the controller. These transitions can then be replaced in the controller with one transition that does not passivate the controller. We show how this is done in Section 4.5.2.

This method shares two properties with the cartesian product. The method is limited to only allowing transitions between the combined states that have only one state change between them. Secondly, this method is not commutative, the *select* function is used to

order the components and therefore the order of the states in the GSS. The approximate method generates the same number of combined states as the cartesian product. However, depending upon the structure of the coupled model, it generates transitions that are less than or equal to the cartesian product. This equivalency in properties with the cartesian product allows the approximate method to be used with the rest of the methodology.

4.3 Defining the Control Objectives

Given a GSS represented as a directed graph G = (V, E), the control objectives can be applied to this graph 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 no state, no transition, and the path exists.

4.3.1.1 The No-State Rule

The no-state rule searches the set of vertices of the graph, V(G), and finds the combined states that match this rule. These states are then removed from the GSS so that the controller does not have unwanted states. The transitions originating from these states are also removed and only transitions to these states are kept for controllability analysis later. The purpose of this rule is to remove unwanted states that are dangerous and violate our objectives or states whose transitions can move the model away from the control path.

However, this rule can also be used to remove non-sensical states that are present in the GSS but whose necessary conditions for being reachable are not possible to occur in real-world applications. Figure 16 shows an example of removing a state from the GSS.

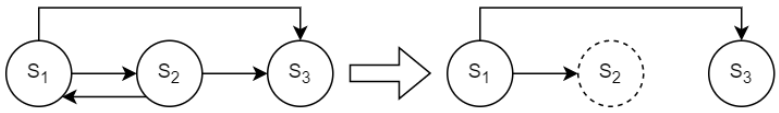

Figure 16. Application of the no state rule on 'S2'. The arrow points to the diagram after the rule is applied and the removed 'S2' is represented by a dotted line.

In Figure 16, the state 'S2' has been marked for removal by the control objectives. The state 'S2' is removed and its transition 'S2' → 'S3' is also removed. The transition from 'S1' → 'S2' is kept, this is because the state 'S1' is still part of the GSS, and it has the potential to transition to 'S2' given the necessary conditions. However, the transition 'S2' → 'S3' is dependent on the existence of the 'S2', removing this from the controller means it cannot act on this state and disables any transitions from it. It is important to mention, the state 'S2' exists in the model's GSS and can be reached, if the path is weakly-controllable (Section 4.4).

4.3.1.2 The No -Transition Rule

The no transition rule searches the set of edges E(G) of the graph and removes transitions between states specified in the no transition rule. There are two reasons to do this. The first is to remove unwanted transitions to states that would cause the model to stray away from controllable states or are illegal states themselves. The second reason is to prevent the controller from switching between legal states and creating multiple paths that would

disrupt the control loop and make the model uncontrollable. Figure 17 shows the application of the no transition rule.

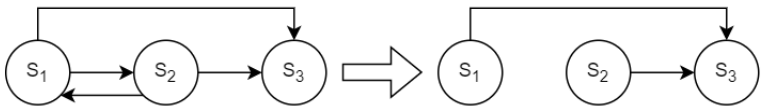

Figure 17. Application of no transition rule between 'S1' and 'S2'. The arrow point to the graph after the application of the rule.

The transition between 'S1' → 'S2' has been marked for removal. This transition is removed from the GSS. The transition between 'S1' → 'S3' and 'S2' → 'S3' are kept. In this process, no states are removed from the GSS. As before, the transitions exist in the individual components of the coupled model, however, by removing them here the obtained controller does not have these transitions and can therefore not trigger these transitions and as a result, disables them.

It should be noted that both the no transition and no state rule can be applied to a sub-set of a combined state. Given a state graph whose combined states are composed of states from atomics 'A', 'B' and 'C'. We can specify that certain compositions for states from atomics 'A' and 'C' are unwanted and therefore require removal for example consider the no state rule that specifies this state {A.1,? , C.1} needs to be removed. This means that all combined states that feature 'A.1' and 'C.1' together are removed, regardless of the current state of the atomic B. This can be done as the state of atomic 'B' cannot change the violation of control objectives caused by the combination 'A.1' and 'C.1'. If in case, the state of atomic 'B' does cause a violation of objectives in that state, then we can include the states of B that violate the control objectives.

4.3.1.3 The Path Exists Rule

Finally, the path exists rule establishes that a closed path exists between any two combined states in the system. The existence of the closed path shows that a control loop exists in our path. Since the graph is a directed graph, the closed-loop can have different forward and reverse paths to the two states specified and therefore must be searched in both directions as shown in Figure 18.

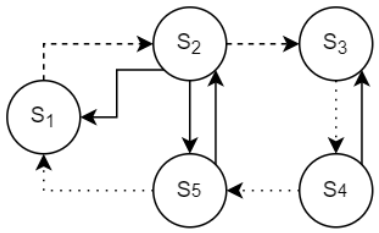

Figure 18. Paths between the states 'S1' and 'S3'. The forward path is shown by dashed lines and the return path is represented by the dotted line.

Figure 18 shows a graph with multiple paths between the starting state 'S1' and ending state 'S3'. The path is established by starting from a specified starting state 'S1' and then following transitions (external or internal) to the ending state 'S3' using a Breadth-First Search (BFS) traversal algorithm [91] forming the path shown by the dashed lines. In a BFS, from a starting node, all connected nodes are searched and added to a queue, then for each visited node, its connected nodes are searched and added to the queue, and so on until the ending state is found. In case, all the nodes are searched, and the ending state is not found then a path does not exist.

Similarly, the return path is established with a BFS traversal with the ending combined state 'S3' as the starting point and the starting state 'S1' as the ending state, forming the path shown by the dotted line.

In our approach, the uniqueness of the path is not determined and there can be other paths between the starting and ending states, for example, there is a second return path ‘S5’ → ‘S2’ → ‘S1’. This is not a major concern, so long as there is a strongly controllable control loop between our starting and ending state the controller should be able to keep up with the state of the plant model. The multiple paths on the control loop are reduced by using more refined specifications for the no-transition rule. In this case, the removal of transition ‘S5’ → ‘S2’ can disrupt the second return path.

4.3.2 Order of Application for the Control Objective Rules

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. The order we use is as follows.

1. No State rules.
2. No Transition rules.
3. The path exists rule.

Applying the rules in this order results in minimum operations and a valid reduced GSS. By valid we mean the determination of path existence from the path-exist rule hold. Applying the no state rule first removes all unwanted states from the system. The dropped states remove the transitions that they contribute to the GSS. If instead, we apply the no transition rule first then we are left with transitions to and from illegal states that will subsequently have to be removed.

We then apply the no transition rule and remove all unwanted transitions. If the path exists rule is applied before any of the other two rules, the result obtained can be invalidated by any subsequent rule application. The no state rule can remove a state and its transitions, this removal can disrupt a path discovered by the path exists rule. Similarly, the no transition rule can remove a transition to a previously accessible state on the computed path that is no longer accessible. Therefore, the path exists rule is applied last, and we calculate if a path exists between two states.

It may be feasible to apply the path-exists rule with the no-state and no-transition rule. In this approach, the search is carried out such that at each combined state the transitions and the connected states are compared against the rules. Only those transitions and states are included that comply with the specifications. This approach is complicated as it applies all the rules in one traversal of the GSS. This may cause issues in case the control objectives require that there be multiple control loops. In that case, there will be a path for each path-exist rule, and combining those paths is another issue. We, therefore, apply each rule separately and, in the sequence, defined in this section.

4.4 Controllability Analysis of the Reduced GSS

The application of the control objectives leaves a reduced GSS that can be used to obtain a supervisory controller that meets the modelers' specifications. the supervisory controller is a DEC. However, before a supervisory controller can be extracted from this reduced graph, controllability analysis needs to be performed to ensure the obtained DEC will keep the model on the desired path. As discussed in Chapter 2, the objective of this step is to ensure the DEC and the reduced GSS only have either very-strongly controllable transitions or strongly controllable transitions.

Since we have defined the time advance in terms of either finite time or infinite time (passivation of the model). We explain the criteria we follow to determine if a transition is very strongly, strongly, or weakly controllable. The time advance is set after the transition has occurred. The criteria for the controllability of an external transition are discussed first and then we talk about the internal transition.

For a state, with an external transition, regardless of time advance type, if the transition moves the model to an illegal state, then the transition is weakly controlled, and the control path becomes weakly controllable. Since this is an external transition, its execution cannot be prevented, this is because if the model is in this state then an external triggering event can at any time cause the transition to execute.

If instead, the external transition is to an allowed state and the time advance is set to a finite time, this transition is considered strongly controllable. This is because in this case the external transition may be followed by an internal transition, the finite time advance duration can allow external input to be received by the component before the time advance ends and move the model to another state, not on the control path.

For an internal transition, if the time advance is finite and the transition is to an allowed state then the transition is strongly controllable as the current internal transition cannot move the model to any other state. However, the next internal transition may move the model to an illegal state. If the transition is to an illegal state, then the transition and the path are weakly controllable. The internal transition will take the DEC off the control path unless an external transition moves it back onto the path.

Similarly, if the time advance is infinite. The internal transition is very strongly controllable as the model and DEC cannot stray away from the path due to internal

transitions. The component passivates after the transition and therefore no other internal transition can move the component away from the current state.

In summary for a very strongly or a strongly controllable path, the transitions must only be to allowed states, and additionally, for internal transitions, the time advance should passivate the model after the execution of the transition to an allowed state.

If in our model, we find that there are weakly-controlled transitions then we resolve this, as discussed in Chapter 2, by either changing or adding more sensors. Adding more sensors helps controllability by increasing observability and creating more paths in the GSS that may be more strongly controllable than the currently available paths in the GSS. Changing the sensor in some way, for example, increasing/decreasing sensitivity or changing its location can help make the path more controllable by providing more prompt information and giving more time to the DEC to force the model back to a controllable path.

Ideally, our path should only contain very strongly controllable transitions. However, strongly controllable paths can still be utilized to make DECs under some conditions. That is there must be time before a transition occurs between two states 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. Fortunately, in many practical applications there exist external transitions in the model that can keep the model on the desired control path, and the period is large enough that the DEC can generate output events to trigger the external transitions and keep the model on the control path.

4.5 Obtaining a DEC Using the Inverse DEVS Transform

As we stated in chapter 2, a very strongly controllable path can be used to extract a DEC using the Inverse DEVS transformation. The Inverse DEVS transformation is performed on the reduced GSS to obtain our candidate DEC. The information about the transition type is stored in the weights of the edge of the graph. Therefore, we can simply change the type of the edge by modifying the weights and changing the internal transitions to external transitions and external transitions to internal transitions. Due to the change in the transition type, the I/O ports are switched as well. The input port becomes the output port for the output function and the output port becomes the input port.

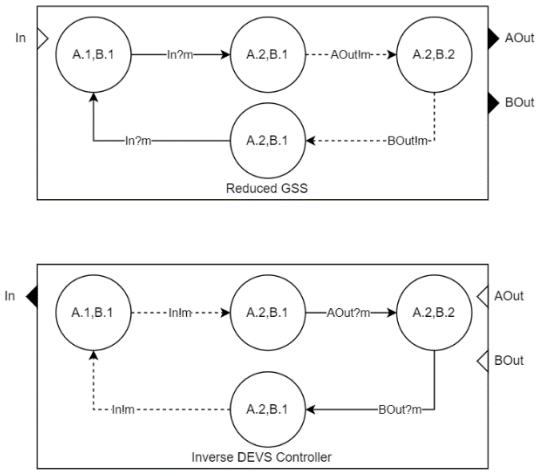

Figure 19. A reduced GSS shows the control loop and the Inverse DEVS of the loop.

Figure 19 shows a reduced GSS with its combined states and transitions and the input port ‘In’ and output ports ‘AOut’, ‘BOut’ of the coupled model. The reduced GSS is the control loop desired by the modeler and was obtained by applying the control objectives. The Inverse DEVS takes this reduced GSS and the coupled model information and swaps the internal and external transitions. We can see Inverse DEVS transform maintains the directionality of the transition but changes the transition type, in addition,

now the input ports are ‘AOut’, ‘BOut’ and the output port is ‘In’. By using the GSS and the coupled model information we obtain a DEVS atomic DEC that can now be coupled with the plant model.

The output types of the model are the same as the input types of the DEC and vice versa, hence if the two models are coupled and started in the same state, the model and DEC stay coordinated. An output event from the model triggers an external transition in the DEC, which causes the same state change in the DEC as the model, thus keeping them coordinated. The DEC may now execute an internal transition from this state to the next state and the output function generates an output. This output event triggers an external transition in the model and this time the model’s combined state is changed to the combined state of the DEC.

We can now see how a weakly controllable path can cause the DEC to not stay coordinated with the model. If the model moves to a state, not on the path, the DEC will not have matching transitions to handle this case and therefore will not change its state to match the state of the model. Unless the model returns to the state the DEC is in, the model and DEC will veer away from each other and not have the same combined states.

4.5.1 Further Simplifying the DEC Manually

Once we obtain a DEC, we can run simulations of the DEC and the plant model coupled together to test for proper functioning. The plant and DEC models are coupled as shown in Figure 20. The DEC input port is coupled with its namesake output port of the model, the output port of the DEC is similarly coupled with the input port of the model. This is also useful to identify any extra transitions that are not used during the normal operation of the plant-controller coupled model.

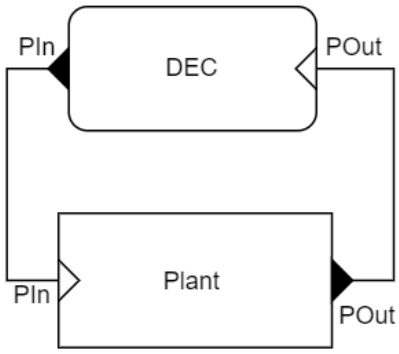

Figure 20. A coupled model of the plant model and the obtained DEC.

These transitions do not alter the course of the operation and may be left as is, however, they will unnecessarily take up memory and storage resources during implementation and should be removed. These extra transitions arise primarily because the control objectives do not identify them as problematic and are therefore not removed during the application of the control objectives. They are transitions between combined states that theoretically may occur during operation but in practice cannot occur as the sequence of events required to reach those states and execute the transitions does not occur.

We can also remove combined states that are not utilized during operation either because they are impossible to physically occur or as described in the previous paragraph may not be physically reachable on the current control path of the system. Refining the control objectives can help reduce these transitions and states at the expense of making the objectives more complicated. In practice, it is difficult to identify all the unused states and transitions and ideally should not be the focus of the control objectives. The control objectives primarily should be restricted to ensuring illegal states and transitions do not occur and a control loop exists in the model that fulfills the objectives. Unused and

impossible states and transitions can be easily removed manually from the DEC as we have demonstrated in Chapter 5 of this thesis.

4.5.2 Dealing With 'Missing' Transitions and Simulating the Plant

At this point, it is important to identify the "missing transitions" that we discussed in Section 4.2.2.6. The "missing transitions" are trivial to find given the DEC and coupled model. We start by searching for external transitions in the DEC whose associated time advance function is infinite. For these transitions, we note which state has changed in the combined state, this provides us with 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. This component is the first component of the IC. We, therefore, search the list of IC and find this atomic model. Once we identify this atomic model, we know the second component of the IC and from this, we can identify the external transition in the second atomic model that will be executed because of the first atomic component.

This external transition can now be used to find the inverse internal transition in the DEC. We find the internal transition in the DEC that corresponds with the state change in the external transition. With the identification of both the transitions involved in the "missing transition" the external transition of the DEC is altered such that the transition is to the state after the internal transition has been executed. If we refer back to the example in Figure 14 and Figure 15, the two transitions, an external transition from 'A_1, B_1' → 'A_2, B_1' and then an internal transition from 'A_2, B_1' → 'A_2, B_2', are replaced by one external transition from 'A_1, B_1' → 'A_2, B_2'. We set the time advance after the external transition to the time advance after the internal transition. By doing this the

external transition of DEC results in the same combined state and has the same time advance as the external transition of the second component of the IC.

The DEC and the model are now simulated using the Cadmium tool to identify any problems in the model by providing test inputs. If any problems are found their cause can be determined and fixed by altering the model or the control objectives. After this testing is done the models are ported onto an embedded platform using RT-DEVS formalism and the RT-Cadmium tool. Since all of our models are defined in terms of DEVS models, no changes need to be made to the models.

The simulation scenarios defined for the software simulation can now be executed on the embedded platform. This testing can more realistically test the DEC in real environmental conditions and identify any further problems. If no problems are found, then we can be assured that the DEC and the model work effectively.

4.6 Consideration for Inclusion of Human Interaction with A Plant

The hybrid layer as described in Chapter 2 is a layer inserted between the continuous and discrete models of the plant in a CPS. Its acts as an interface between the two domains and helps ease the modeling and simulation process by dividing and simplifying the design of the two distinct models. In our thesis, we use the hybrid layer to introduce human interaction with the plant model and reduce the transitions in the model. The hybrid layer presents an abstract view of the model to the DEC. How this model is obtained is discussed in Section 4.6.2.

In addition, the use of the hybrid layer in this manner allows us to use the methodology described in this Chapter to obtain the DEC without any modifications. The

DEC is obtained by carrying out the process described in this chapter on the abstract plant model.

4.6.1 The Problem with Representing Human Interaction as a Sensor

Human interaction with the plant model can be modeled as a sensor. The human sensor can produce changes to the model by triggering internal transitions. In this thesis, for comparison with the hybrid layer, we implement the non-hybrid layer with the human interaction as a sensor triggering internal transitions of the room model in Chapter 5. Alternatively, we can model the human as an external input to the model, in this case, it will generate external triggering events for the model.

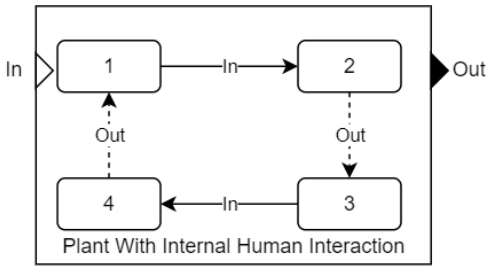

Figure 21. Plant atomic model with human interaction modeled as internal transitions.

Figure 21 shows a plant model where the effect of the human interaction has been modeled as a sensor causing state changes using internal transitions. The model receives some external triggering events through the port 'In'. The internal transition function is defined such that it checks a special variable, whose value is controlled by the human operator. The internal transition function causes a state change when a pre-defined threshold value is crossed. This also causes the output function to generate an output event. A disadvantage with this approach is that the model cannot be passivated, otherwise the human entity cannot update the value of the variable and a state change cannot occur.

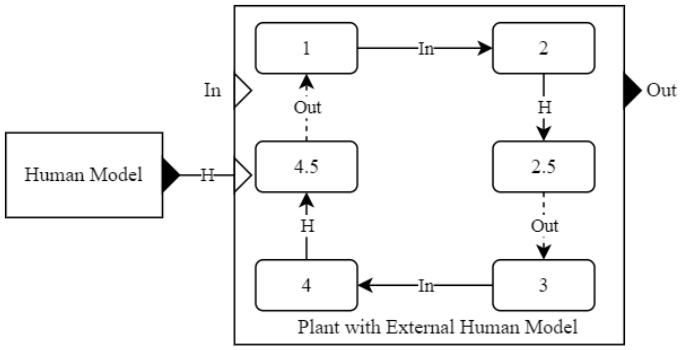

Figure 22. Plant atomic model with the human interaction modeled as a separate model.

Figure 22 shows the other method in which we model the human entity separate from the plant model. This model is modeled to achieve the same effect as the model in Figure 21. Therefore, the external transitions associated with the 'In' port are the same. The input port 'H' of the plant model is coupled with the output port of the human model. The external transitions execute on the same states as in the previous model, therefore, the external transitions cause the state change from states '2' and '4'. As discussed in this chapter, the output function can produce an output without an associated state change in the internal transition function. However, we do not allow self-loops in the model and thus do not allow the output function to produce an output without the internal transition function causing a state change. Therefore, the external transition caused by the human entity cannot directly move the plant model to the next state, we need two additional states '2.5' and '4.5', which can then through an internal transition move to the next state as in the original model.

The model does not need to keep track of a special variable and can be passivated, as an external transition can always be triggered regardless of the current time advance of the model. The human model still models human interaction using internal transition as

was done for the previous model. So, in effect, we have moved this internal transition from the plant model to another model. We, therefore, chose to model human interaction as a sensor in the plant model.

In general, the implementation of humans as a sensor adds complexity to the model due to the extra states and transitions that must be included to cater to human interaction. This is reflected as added complexity in the DEC. This is especially true in the case of a human entity changing the setpoints of a model, the plant model must have transitions from each state to every other state such that a change in the setpoint can be catered by the model as shown in Figure 23.

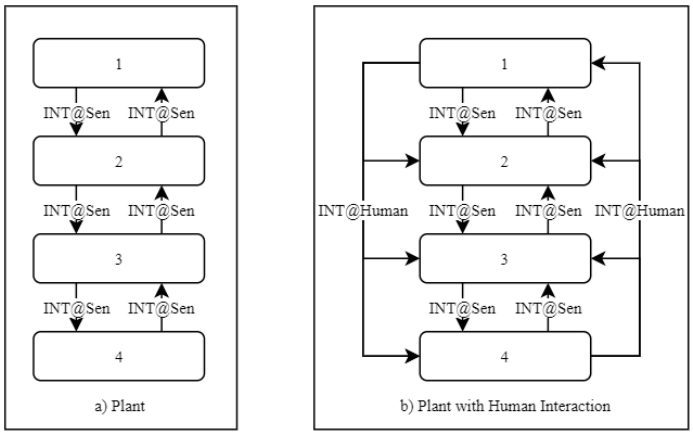

Figure 23. Two plant models: a) Model without human interaction; b) Model with human interaction.

Figure 23a represents a plant model that models some physical quantity with each state representing a different threshold, for example, temperature ranges. The temperature is reported by a sensor that triggers internal transitions once a threshold is crossed represented by 'INT@Sen'. It is reasonable to model the transitions in sequential steps from '1' to '4' and back from '4' to '1' as most physical properties can be expected to

move from one threshold to the next. The model in Figure 23b includes human interaction with the model such that the human is changing the setpoint of the system, i.e., changing the threshold values of the states. In this case, each of the states needs to have an internal transition, 'INT@Human', with all the other states. This is because changing the set-point can move the threshold values such that the new threshold value can be in any of the other states. Thus, the number of transitions increases due to the addition of human interaction with the model.

4.6.2 Utilizing the Hybrid Layer to Introduce Human Interaction to the Plant

We utilize the hybrid layer to provide an abstract view of the plant model to the supervisory controller. The abstract view removes the human entity and its effect on the model from the view of the supervisor, instead, the supervisor sees a simpler DES as shown in Figure 24.

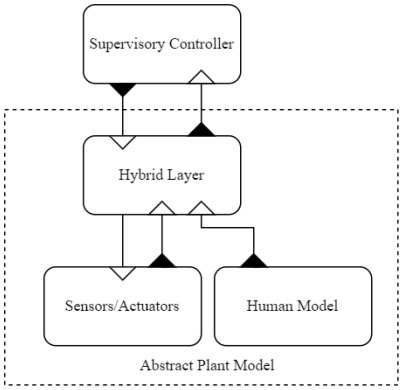

Figure 24. Abstract plant model view for the Supervisory controller.

In Figure 24 we see the supervisory controller only communicates with the hybrid layer. This allows the hybrid layer to present an abstract view of the plant model that only provides the necessary minimum information to the supervisory DEC. The hybrid layer

updates the state of this abstract model with the information from the sensors and the human model. It also passes on the control signals of the DEC to the actuators.

To create this abstract view of the model we need to first create an accurate model of the plant that incorporates the behavior of the human entity. The state transitions of the model are then analyzed, and we identify the state transitions caused by human interaction between any two states in the model. We check if these transitions can be replaced by a series of transitions that connect the two states. If such a series of transitions exist, we can remove the human-caused transition from our model. This gives us a reduced model that forms our abstract DES.

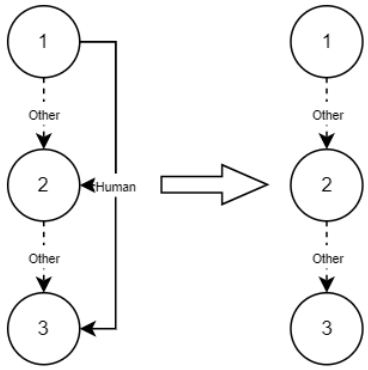

Figure 25. State transitions from a model where all the human-caused transitions can be removed.

Figure 25 shows the case where all human-caused transitions can be removed. The state transitions caused by humans are solid lines and are labeled ‘Human’, all other transitions are represented by a dashed line and labeled ‘Other’. It can be seen the transition ‘1’ → ‘2’ has another transition linking it, therefore the human-caused transition can be removed. Similarly, the human-caused transition from ‘1’ → ‘3’ can be removed as there is a series of transitions ‘1’ → ‘2’ → ‘3’ available. The resulting abstract model obtained is shown to the right of the model.

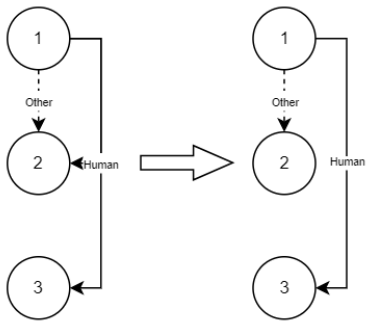

Figure 26. State transitions from a model where all the human-caused transitions cannot be removed.

Figure 26 shows the case when a human-caused transition cannot be removed. In this case, the human-caused transition between '1' → '2' can be removed as before. However, the human-caused transition '1' → '3' cannot be removed as there are no other series of transitions that can reach from state '1' to '3'.

4.6.3 A General Approach to Introduce Human Interaction Using Hybrid Layer

We can use the methodology outlined in this chapter to form a DEC based on the abstract DES without any changes to the methodology. The removed transitions have an influence on the model and therefore must be catered for, the hybrid layer then needs to incorporate these removed transitions while hiding the complexity from the DEC. Even though the implementation details for each plant model are different, we find that a general implementation is possible.

If the human user is altering the setpoints of the system such that the resultant desired state is on the current control path of the model, then the general approach can be applied. The analysis we do in obtaining the abstract view ensures that our system has such a control path. This is because we only removed those human-caused transitions from between the states that have another path between them. The general implementation is an

algorithm that ensures the DEC and the model eventually reach the desired state and is shown in Figure 27.

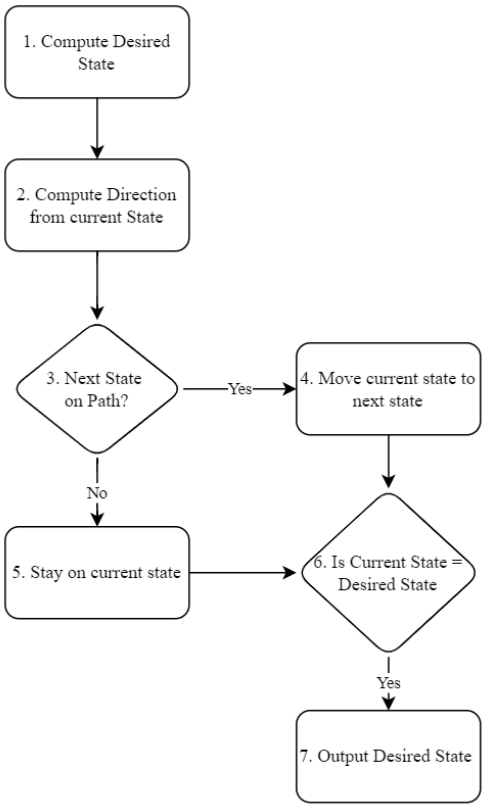

Figure 27. General algorithm to incorporate human interaction in the hybrid layer for the unidirectional case as a flowchart. The steps are numbered.

We will use numbers in curved brackets, for example "(1)", to refer to the step being described in Figure 27 in this section. (1) The algorithm first checks, that for the given human setpoint what the desired state of the model would be. We define the desired state as the state the model would currently be in given the new setpoint, the current state is the state of the abstract model being output to the DEC and is initially the state of the model immediately before the setpoint change. (2) The current state is then checked, and we observe if the desired state is ahead of the current state in the path or behind the current state in the path. This directionality is explained in Figure 28.

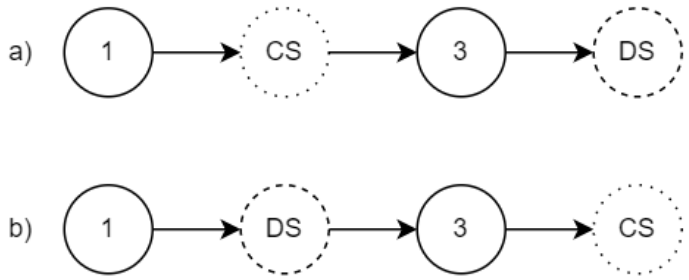

Figure 28. Position of the current state, marked as 'CS' (dotted circle), relative to the desired state marked as 'DS' (dashed circle): a) 'CS' is behind the 'DS'; b) 'CS' is ahead of 'DS'.

Figure 28a shows the case where the current state is behind the desired state, the transition from the current state can advance towards the desired state. The desired state is one of the next states of the path. In Figure 28b, the case where the current state is ahead of desired is shown. In this case, the current state cannot move towards the desired state as the desired state is one of the previous states on the path.

If our control loop is bidirectional, then we simply transition to the next or previous state and provide the DEC with information about this state change. (4) The states are sequentially advanced toward the direction of the desired state until (6) we reach the desired state. Bidirectional and unidirectional loops are shown in Figure 29.

Figure 29 shows two control loops displaying three states and transitions between them. In the bidirectional loop, each adjacent pair of states has transitions in both directions, allowing the current state to move to the next state or the previous state. In the unidirectional case, the states only have transitions in one direction. The current state can only move to the next state and cannot return to the previous state.

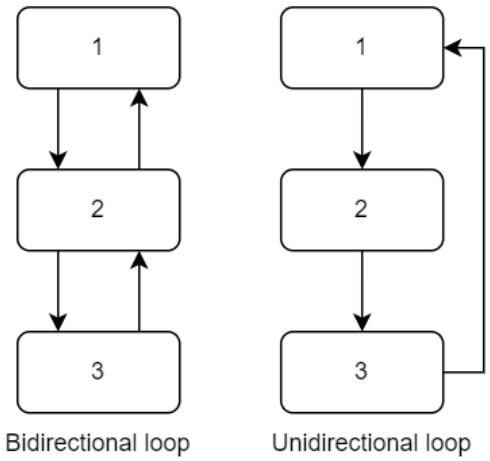

Figure 29. Bidirectional and unidirectional loops.

In case, the control path is unidirectional. We check if the desired state can be reached on the current path i.e., is the current state behind the desired state. (4) If this is possible then we carry out the procedure as described for the bidirectional case. (5) If this is not possible as the current state is ahead of the desired state, then we stay on this state until the desired state reaches this state.

It is important to note the hybrid layer keeps track of the condition of the model from the sensor information it receives. The desired state is not kept static, it is continuously updated according to the new information from the sensors. If the model is accurate the desired state can only transition and advance towards the current state and not move backward and away from the current state. (7)When the desired state reaches the current state, the hybrid layer outputs the current state to the DEC.

Consider an example for the unidirectional case where the current state is ahead of the desired state. If given states ‘A’, ‘B’, ‘C’, ‘D’ and ‘E’ such that the control path is unidirectional and the path is ‘A’ → ‘B’ → ‘C’ → ‘D’ → ‘E’. If the state currently is ‘D’

and human interaction with the hybrid layer causes the desired state to be ‘B’. We know the state of the model will advance in the sequence ‘B’ → ‘C’ → ‘D’ as this is how the plant was modeled. We, therefore, keep the current state as ‘D’ and let the desired state update as the model naturally moves to the current state. Once the desired state reaches the current state the hybrid layer and the DEC are working on the new set point.

The tradeoff of this approach is in state-space complexity vs execution time. If we do not use the hybrid layer the resulting state space and DEC will have more transitions, but any human interaction will reach the desired state in one transition, and therefore the time it takes is one time advance to reach the desired state. If instead, we use the hybrid layer the resulting state space and DEC will have fewer transitions, but the time taken for the current state and the desired state to reach the same state is dependent on whether the current state is being advanced to reach the desired state or vice versa.

The time it takes for the case, where the current state is being updated to reach the desired state is the $\sum (ta_i)$, where i goes from 1 to k and ta_i is the time advance of the current state and ta_k is the time advance of the state before the desired state. We can change the ta for each state change, to a uniform ta. The time it would take then becomes simply ta*n. Where n is the distance between the two states. Therefore, the time to switch to the desired state can be minimized. This is possible in cases where the DEC does not have transitions that need time to complete a step, or in the case, the transitions have the same time advance.

For the second case where the desired state is updated to reach the current state the time it takes depends on the individual model implementation. This is because the transitions execute due to the model crossing a threshold value. The time it takes for this threshold to be crossed is dependent upon environmental factors. If the time it takes to

reach the current state is unacceptable for a process, then we can model our abstract and plant model such that the control path is bidirectional. The time taken would then be ta*n as discussed in the previous paragraph.

This method works because in many situations the plant behaves predictably based on our models. For example, take the case of an air conditioner cooling a room (we use this as a case study in chapter 5). The room can only cool down if the air conditioner is on and actively cooling the room, on the other hand if the air conditioner is turned off the room will heat up over time. We can use the hybrid layer to exploit this behavior and simplify the plant model for the DEC. If for any reason the model deviates from this behavior, then there is something wrong with either the model or an error has occurred in the plant. We can have another supervisory controller that can detect this error and take corrective action. Error correction however is not in the scope of this thesis, and we only focus on creating a DEC based on abstract plant models using the hybrid layer.

Chapter 5: Supervisory Control of a Smart Building - A Case Study

In chapter 4, we described the methodology that is used to develop supervisory controllers using the DEVS formalism and our approximate method. In this chapter, we apply the methodology to demonstrate the practical applicability of this method and show it by implementing a case study that model of a smart building. The control system oversees room conditioning, including Air Conditioning (AC) and CO2 concentration detection. Additionally, we model smoke detectors for fire detection, occupancy detection for light control, and water temperature control for the room. The model consists of 5 control loops, each completing the desired control objective. We will go through the steps of defining the models and the control objectives, applying the objectives, obtaining a controller and then reducing the controller using a software simulation. Finally, we execute the models on a hardware platform to demonstrate the practical applicability of our method.

The AC model is used for room temperature control with human interaction using the hybrid layer. For comparison, we model it first without using the hybrid layer and then with the hybrid layer. We show how the hybrid layer is modeled in this case and how it reduces the complexity introduced by human interaction.

Before we implement our models, we test the modified methodology on a known model defined in the original methodology of Zeigler et al. [15] to demonstrate the correctness of our method. We implement and test the Discrete Event Controller (DEC) obtained using their method on hardware using RT-DEVS and the Cadmium tool. We then obtain a DEC using our method and test it with the same conditions used to test the original DEC. We then compare the result to show our method can obtain working controllers.

5.1 Implementing the Methodology on a Known Model

To test the correctness of our method, we apply our methodology to a coupled model of a water pump and tank system defined in the original methodology. By comparing the simulation results of this known model to the results that are obtained by our methodology, we show an example of how the results obtained are equivalent to the results in our method. In the first part of this section, including Section 5.1.1, we discuss the model as defined by Zeigler et al. [15] and their extracted DEC. We implement this model and DEC using RT-DEVS and Cadmium to execute on an embedded platform and observe its behavior. In section 5.1.2 we use the model definitions from the paper and implement it using the approximate method to obtain a DEC and then test it and compare the results from these two tests.

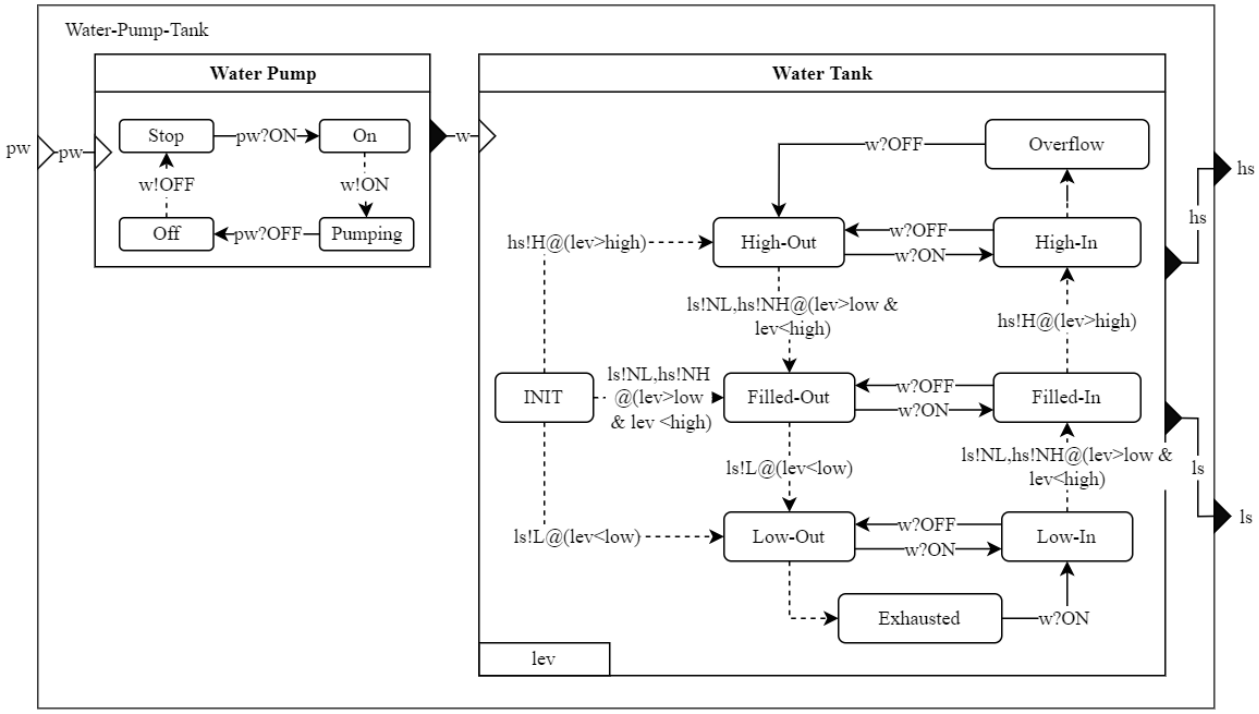

Figure 30. Water Tank model connected to a water pump from Zeigler et al. [15].

The model is shown in Figure 30. This model controls the flow of water into a water tank by switching a water pump on or off depending upon the level of water in the tank.. We use the DEVS graph notation as explained in Chapter 2 to graphically represent coupled and atomic models in this figure and this notation will be used in all figures in this chapter. As a quick recap, the states of the models in Figure 30 are shown by round boxes and the transition are arrows that connect them. The dashed lines represent internal transitions, and the label represents the output port used and the output generated by the output function. The condition for the transition is stated after the '@' symbol. The external transitions are represented by solid lines and the label represents the input port used and input required to trigger that state change in the external transition function. In a coupled model, solid lines represent couplings.

As we can see in the figure, at the top level coupled model we use two sensors, 'hs' and 'ls', which detect high low water levels respectively. Their outputs are transmitted using the 'hs' and 'ls' ports. The water pump can be turned on and off through commands in the external port 'pw'. There is an internal coupling between the pump model and the water tank model using the port 'w'. This allows incorporating the effect of the water pump. The water tank model defines an increase or decrease in water level due to the water pump's actions. The states ending in '-In' represent the water level rising due to the pump being on and those ending in '-Out' represent the water level falling due to the pump being off.

To obtain a DEC the control objectives for this model seek to achieve three things:

- No state rule: prevent the tank model from reaching overflow or underflow states.

- No transition rule: prevent the water tank model from switching needlessly between the Filled-In and Filled-Out states.
- The path exists rule: establish that there is a control loop from low-on to high-on and then from high-off to low-off.

With these control objectives in mind, Zeigler et al. [15], obtain a strongly controllable control path. The DEC that can be obtained from this control path will meet the control objectives. However, before the Inverse DEVS transform is applied to this control path the authors apply state reduction techniques to the GSS to reduce the number of states and transitions. From this, a minimal DEC is obtained and is shown in Figure 31.

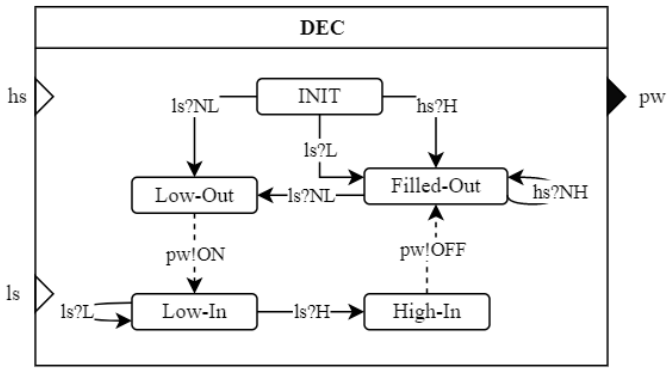

Figure 31. Minimal water pump controller from Zeigler et al. [15].

Figure 31 shows the minimal controller obtained by the authors. The minimal controller contains the minimum number of states and transitions needed to keep the model on the control path. This can be done by removing states for which the output function does not generate a controlling output to the model. For these states, multiple transitions can be combined to produce one transition. The controller has two input ports ‘hs’ and ‘ls’ and an output port ‘pw’. The controller monitors the water level of the water tank based on external

sensor inputs from the water tank through the input ports. If a control action is needed the controller can issue an appropriate command to the pump through the output port ‘pw’.

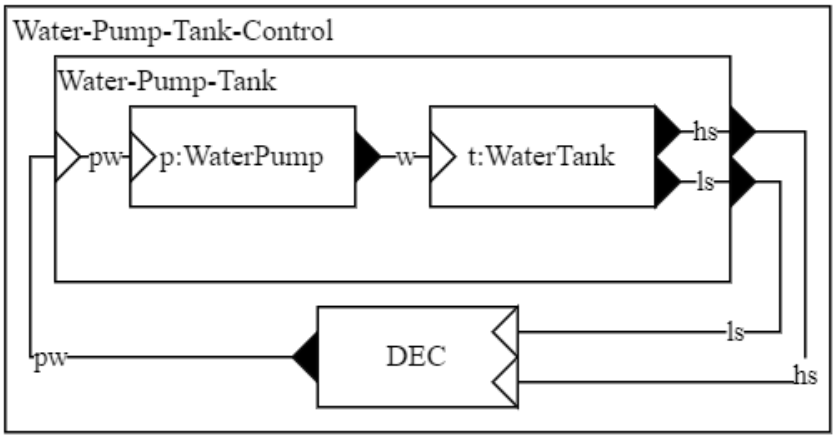

Figure 32. The coupling of the DEC with the water pump and tank coupled model labeled.

Figure 32 shows the coupled model definition, including the DEC with the water pump and tank coupled model. The DEC output port ‘pw’ is coupled with the input port ‘pw’ of the coupled model, the output ports of the coupled model ‘hs’ and ‘ls’ are similarly coupled with the input port of the DEC ‘hs’ and ‘ls’.

We implemented this coupled model with the minimal controller and the models in Figure 30 using Cadmium. Through simulation in software, we observe the minimal controller establishes a control loop such that the pump model does not fill the water tank model higher than the ‘High’ threshold. When this happens the DEC output function generates an output event ‘OFF’ through the ‘pw’ port that turns the pump ‘OFF’. Similarly, when the state is ‘Low’, the output function generates an ‘ON’ event that forces the pump to turn on. The model cycles between these states ensuring the water tank level does not over or underflow.

5.1.1 RT-DEVS Execution

To test the model on an embedded target platform, we implement it using the RT-DEVS formalism and the Cadmium tool. The DEC controls the level of a water tank based on the output of the sensor from a tank and issues controlling commands to a water pump. Due to limited resources and time, we simulate the water level in the water tank model and the action of the pump in the pump model. If given a water tank and pump were provided, the models and the DEC do not need to be modified except for replacing the simulated water level in the water tank model the water level sensors and internally sending control signals from the pump actuator model to the pump relay (including the corresponding calibration). The DEC will remain unchanged as it operates on discrete events and no changes are made to the events. This is true for the case study as well where the obtained DECs will work without modifications and only the models are altered, without changing the states and transitions, to interface with the sensors and actuators.

To test the model on an embedded target platform, we implement it using the RT-DEVS formalism and the Cadmium tool. We use the results of this test to compare with the DEC that we obtain in Section 5.1.2 using our approximate method and test in Section 5.1.3. By doing this we are comparing the results from our method with a known DEC in the literature and show our method works as expected. The DEC controls the level of a water tank based on the output of the sensor from a tank and issues controlling commands to a water pump. Due to limited resources and time, the water level in the tank is simulated in software. When the pump is on the water level is incremented in the water tank model. Once the pump is off the water level is decremented if the user is pressing a button to indicate the water is being drained from the water tank.

If given an actual water tank and pump, the models and the DEC do not need to be modified except for replacing the simulated water level in the water tank model with values from the water level sensors and internally sending control signals from the pump actuator model to the pump relay (including the corresponding calibration). The DEC will remain unchanged as it operates on discrete events and no changes are made to the events. This is true for the case study as well where the obtained DECs will work without modifications and only the models are altered, without changing the states and transitions, to interface with the sensors and actuators.

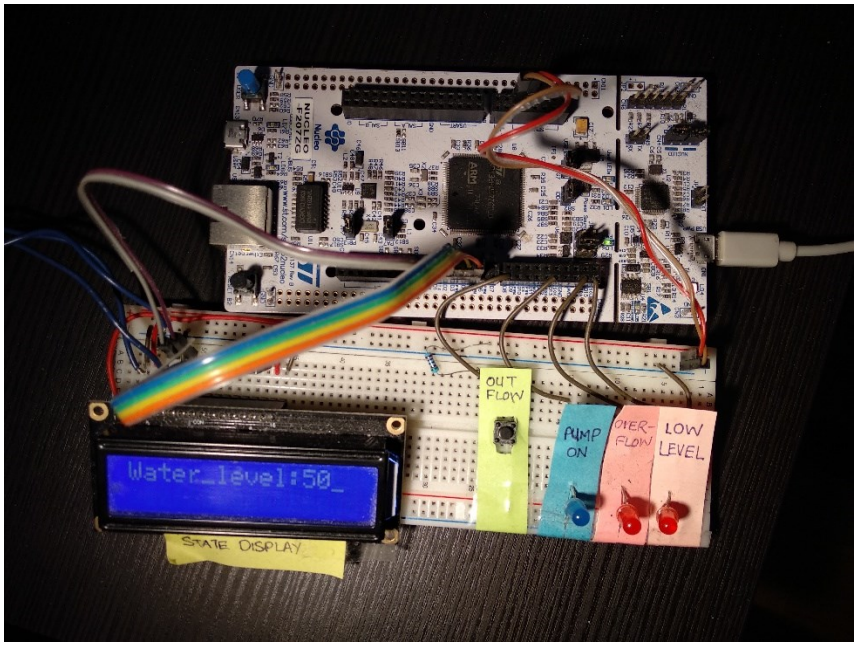

Figure 33. Water pump and tank model and controller hardware setup.

Figure 33 shows the hardware setup used for this experiment. We use a Nucleo F207ZG [75] board running Mbed-OS [73]. We use an LCD to display the current level of water in the tank. The push button drains water from the tank. The blue LCD indicates if the pump is on. The other two red LEDs are labeled and show if the water level reaches the overflow or underflow conditions.

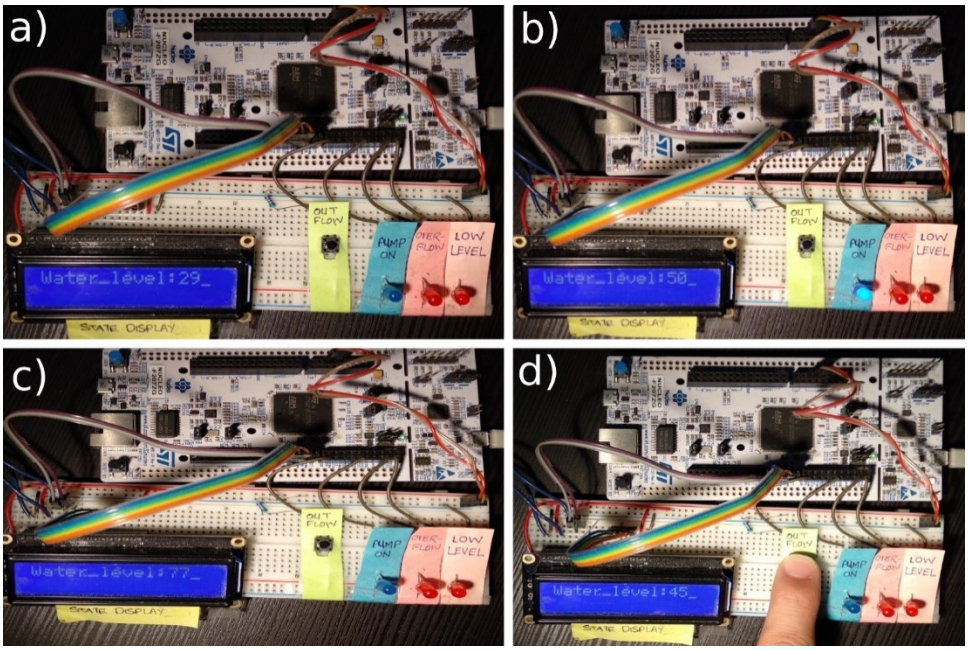

Figure 34. Starting from the top left shows different stages of the simulation. a) Water level reduced and left at 29 for a few moments; b) Pump 'ON' and the water level is actively increasing, currently at 50; c) Water tank full and at 77 with the pump turned off; d) Actively draining the tank after it has been filled once.

Figure 34 shows the different stages of the hardware simulation as the simulation proceeds. The water level is initially set at 50 and the pump is set to 'Off' as shown in Figure 33. We leave the model in the current state for a few seconds, by not pressing the drain button, to see if the state changes without any input from the user. We observe the state does not change, we then press the button and observe the water level decreases, we let the water drain until the water level reaches 29 as shown in Figure 34a. We keep the model in this state for a few seconds to observe if the state changes or the water level decreases without any input. No change was observed, we then press the button to decrease the water level. We set the 'LOW' threshold at 25, when the water level reaches 25, we observe the blue led lights up, indicating the pump has turned on, and the water level begins to rise as shown in Figure 34b. After some time, the water level reaches the 'High'

threshold at 75, when the water level reaches 75 the controller instructs the motor to turn off, however, there is a slight delay due to time advance in the DEC and the pump, and the water level reaches 77 before the pump turns off as shown in Figure 34c. We again leave the water tank in this state and observe if the state or water level changes without any input. We observe no change and therefore press the button again to reduce the water level until it reaches the low water level and the cycle repeats. This test shows how the original DEC fulfills the requirements.

5.1.2 Implementing the Model Using the Approximate Method

We implemented the model defined in the previous section using the approximate method. We use the model definitions shown in Figure 31 to define the model as a directed weighted graph, where the states are the vertices, and the transitions are weighted edges. The weights store the transition type, the input/output port, and the input/output event associated with the transition and time advance update after the transition. These directed weighted graphs are then passed through the algorithm to obtain a GSS. We use the components names defined in Figure 32, where 'p' is an atomic model of type Water Pump (simply called *pump* here) and 't' is an atomic model of type Water Tank (simply called *tank* here). Listing 1 shows a coupled model definition for the water-pump-tank model shown in Figure 30 as defined in our implementation.

```
X = pump.pw,tank.w,coupled.pw
Y = pump.w,tank.ls,tank.hs,coupled.ls,coupled.hs,coupled.ls (1)
D = p,t
M = pump,tank  (2)
EIC = coupled.pw,p.pw
IC = p.w,t.w
EOC = t.hs,coupled.hs
EOC = t.ls,coupled.ls
select = p,t
```

Listing 1. Definition of a coupled model in our implementation to generate a GSS.

Listing 1 shows the model definition to be used by the software implementation of the method. Each line starts with a keyword that defines what set this belongs to. The keywords are named as defined for a coupled model CM = < X, Y, D, {M_i}, EIC, IC, EOC, *select* >. If a keyword appears in more than one line, then the entries from the second line are appended to the entries from the first line. The string after the '=' sign is comma-separated-values, where each is in input for the set defined by the keyword. For 'X' and 'Y', each value is an input or output port in the coupled model (1) and the component models. The component model ports are mentioned here to let the program know these ports exist and will appear later in the definition, as the program may not have yet read the component models yet. The component ports and couplings are verified at the end by accessing the models and checking if the ports exist and they match the expected type. For example, an input port should be in set 'X'. The port name and the model are separated by a '.'. For example, 'tank.w' represents the port 'w' from atomic model 'tank', similarly 't.w' represents the port 'w' from the component 't'.

'D', 'M', and 'select' take values representing the component or model names. (2)The component names 'D' corresponds to the order of appearance of models in 'M', so 'p' and 't' specify the models 'pump' and 'tank' respectively'. The 'EIC', 'EOC', 'IC' are couplings of the coupled model and have two values. The first is the starting port and the second is the terminating port. For 'EIC' both ports must be in the set 'X', for 'IC' the input and output ports must be in the sets 'X' and 'Y' respectively, and finally for 'EOC' both ports must be in the set 'Y'.

We apply the algorithm defined in Section 4.4.2 on this coupled model. The algorithm first sorts the set D with the order of the *select* function list. In this case, 'p' has

a higher priority than ‘t’ in the *select* function. As ‘p’ is the first element, this component is selected from the set. From the coupled model definition, we know this is an instance of the pump atomic model. From this, the algorithm searches the list ‘EIC’ and finds the entry ‘`coupled.pw,p.pw`’. The port ‘pw’ of this component has an EIC with the port ‘pw’ of the coupled model. Then, the algorithm searches the list of the external transitions of the pump atomic model to find all the transitions that receive inputs from the port ‘pw’. In Figure 30, we can see there are two external transitions: ‘Stop’ → ‘On’ and ‘Pumping’ → ‘Off’ that meet the criteria. These are added to the GSS. Taking the first transition ‘Stop’ → ‘On’ the algorithm searches the list of combined states and finds all the pairs of combined states that fit the pattern: {Stop, S_i} and {On, S_i} where ‘S_i’ is the same and can be any state of ‘t’. The transition is added, with the weights, between {Stop, S_i} and {On, S_i}, such that the GSS now has a transition {Stop, S_i} → {On, S_i}. This is done for the second transition as well.

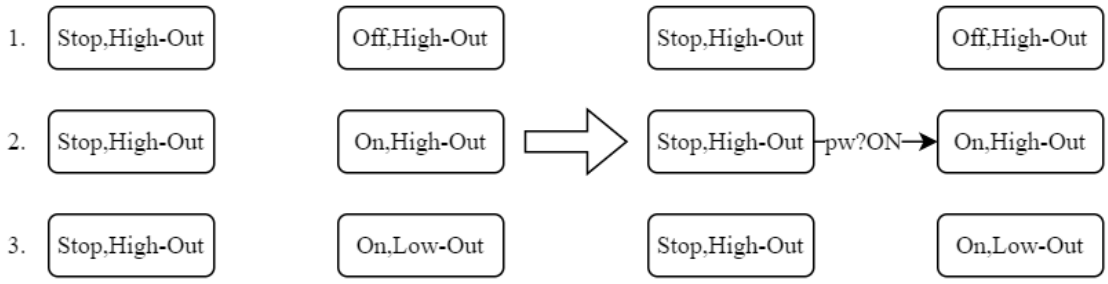

Figure 35. Adding external transition between combined states due to an EIC.

Figure 35 shows the algorithm choosing where to add the external transition ‘Stop’ → ‘On’ between the combined states of the GSS given three pairs of combined states. The first pair does not have ‘On’ in the second combined state and therefore a transition cannot be added between these pairs. The second pair fulfills the criteria, and the external

transition is added in the GSS as shown with the weights. The third pair does have the state change 'Stop' → 'On' between them, however, there is another state change as well 'High-Out' → 'Low-Out' violating the condition that there can only be one state change between a pair of combined states and therefore the external transition is not added between this pair.

The algorithm then moves to the list of 'IC' and finds 'p' has an IC from the entry '`p.w,t.w`'. This time the internal transition list of 'p' is searched looking for which associated output function uses the port 'w'. The transitions found are 'On' → 'Pumping' and 'Off' → 'Stop'. As before, the model finds the transitions that differ only by the state change and adds them to the GSS. In this case, however, the algorithm notes the output type and the second state of the model. Using the transition 'On' → 'Pumping' as an example, the algorithm notes the output 'ON' and the second state 'Pumping'. From the coupling information, the algorithm knows the second component is 't'.

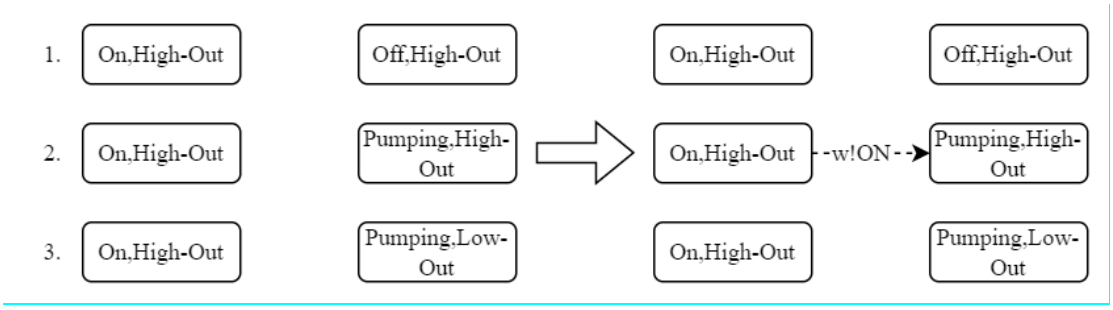

Figure 36. Adding internal transition between combined states due to the first component of an IC.

Figure 36 shows the algorithm choosing where to add the internal transition 'On' → 'Pumping' due to the first component of an IC, between the combined states of the GSS given three pairs of combined states. This case is like that in Figure 35, The first pair does not have 'Pumping' in the second combined state and therefore a transition cannot be added

between these pairs. The second pair fulfills the criteria, and the internal transition is added in the GSS as shown with the weights. The third pair violates the only one state change condition and therefore the internal transition is not added between this pair.

For ‘t’ the algorithm searches the list of external transitions whose input ports match ‘w’ and whose input type matches the output of ‘p’ in this example ‘ON’. It finds three transitions that fulfill the criteria, one of which is ‘Low-Out’ → ‘Low-In’. To add this transition to the GSS, the algorithm searches the list for combined states that match {S_i, Low-Out} → {S_i, High-In}. With the additional condition ‘S_i’ must be the second state of the internal transition of the first component. Since the coupled model only contains two components the transition is added only between {Pumping, Low-Out} → {Pumping, Low-in}, if there were more components then the transition would be added between all pairs {Pumping, Low-Out, S} → {Pumping, Low-In, S}, where ‘S’ is the state/s of the remaining component/s and S does not change.

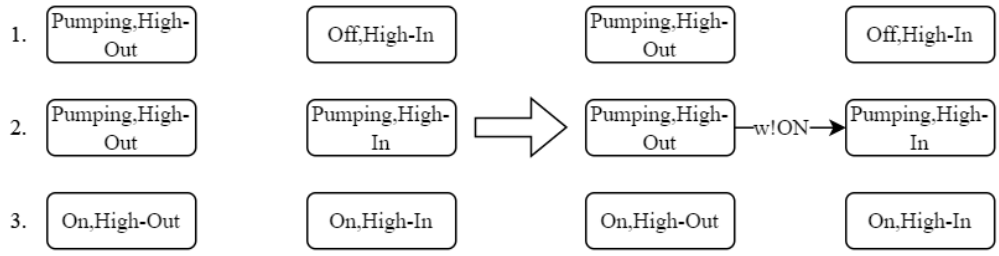

Figure 37. Adding internal transition between combined states due to the second component of an IC.

Figure 37 shows the algorithm choosing where to add the external transition ‘High-Out’ → ‘High-In’ between the combined states of the GSS due to the second component of the IC, given three pairs of combined states. The first pair violates the only one state change per transition condition and therefore a transition cannot be added between these pairs. The second pair fulfills the criteria, and the external transition is added in the GSS

as shown with the weights. The third pair violates the condition that the states of the first component must be the state after the internal transition. In this case, this is the state 'Pumping'. The combined state has the state 'On' and therefore the transition cannot be added, even though this pair satisfies the only one state change per transition condition and has the state change 'High-Out' → 'High-In'.

Finally, the list searches through the 'EOC' list and finds no entries for 'p' and moves to the next component from the set 'D', in this case, 't'. The algorithm finds no entries in the list of 'EIC' and 'IC' and therefore does not add anything. The algorithm finally finds 't' has two EOCs from the entries '`t.ls,coupled.ls`' and '`t.hs,coupled.hs`'. Using '`t.ls,coupled.ls`' the algorithm searches the list of internal transitions whose associated output function uses the port 'ls'. The algorithm finds seven transitions, one of which is 'Filled-Out' → 'Low-Out'. As was done for the EIC case, the algorithm finds all the pairs of combined states such that {S_j, Filled-Out} and {S_j, Low-Out}and in this case 'S_j' is the same and is any state of 'p'. For example, the transition is added to the GSS between {Stop, Filled-Out} and {Stop, Low-Out} such that the GSS has the transition {S_j, Filled-Out} → {S_j, Low-Out}.

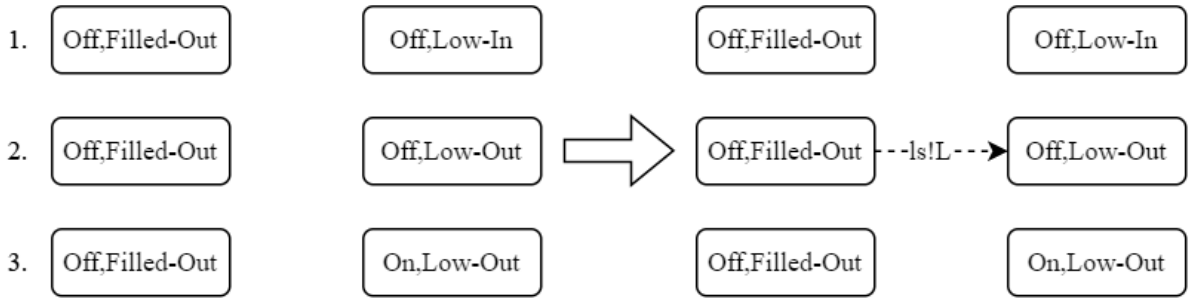

Figure 38. Adding internal transition between combined states due to an EOC.

Figure 38 shows three combined state pairs in the GSS. The addition of the transition, in this case, is due to the internal transition of the second component and needs to be added between the state change 'Filled-Out' → 'Low-Out'. The first pair does not have this state change and so the transition is not added between them. The second pair has this state change and fulfills the other criteria and so an internal transition is added between them. The third pair violates the only one state change for a transition and therefore no transition is added between them.

The algorithm repeats the steps until all the components of set D have been analyzed. The GSS obtained with this method contains 36 states and 96 transitions, in comparison the cartesian product produces 100 transitions. The GSS is then passed through the same control objectives as the paper. In Listing 2, we show how the control objectives are defined in our software implementation.

```
path,?,t.low-out,p.stop,t.high-out
no_state,?,t.exhausted
no_state,?,t.overflow
no_transition,?,t.filled-in,?,t.filled-out
```

Listing 2. The control objectives for the water-pump model.

Listing 2 shows the format in which the control objectives are defined. Each new line defines a separate control objective. The first field is the keyword "path", "no_state" or "no_transition". Following this, the component and states are mentioned that are affected by the objective. The component name 'd' and state are separated by a '.' with the component name written before the state and are ordered according to the *select* function of the coupled model. For example, '`t.exhausted`' represents the state 'exhausted' from component 't'. As discussed in Section 4.3.1.2, the '?' indicates that we do not care

about the states of this component. During implementation the states were defined in all lower-case letters and hence why in Listing 2 they are in lower-case, however, we will use the names from Figure 30 to explain the rules.

Regardless of the ordering in Listing 2, the rules are applied in the order no-state, no-transition, and no-path. The no-state rule '`no_state,?,t.exhausted`' finds all the states in the GSS that match {S_i, Exhausted} where 'S_i' is any state of 'p' and removes these state and transitions originating from these states. A similar procedure is carried out for the second no-state rule '`no_state,?,t.overflow`'.

The no-transition rule '`no_transition,?,t.filled-in,?,t.filled-out`' is applied next that removes any transitions from the GSS between the combined states {S_i, Filled-In}and {S_i, Filled-Out} and 'S_i' is any state of 'p'.

Finally, the path-rule '`path,?,t.low-out,p.stop,t.high-out`' is applied. This states the starting state is {?, Low-Out} regardless of the state of 'p' and the ending state is {Stop, High-Out} where 'p' is in state 'Stop' and 't' is in state 'High-Out'. The program goes through the states {S_i, Filled-Out} until one of the starting states has a path in both forward and reverse directions to {Stop, High-Out}.

After passing the GSS through these rules we obtain a reduced GSS of 28 states and 84 transitions that has a control path that meets our objectives. From this, we obtain a DEC using the Inverse DEVS transformation. We then analyze the DEC and trace out the path of the control loop by starting at the starting state and then moving towards the ending state, in the process we identify and remove any unused transitions and states. The final

DEC that we obtain by tracing the path is shown in Figure 35. The final DEC has 12 states and 14 transitions.

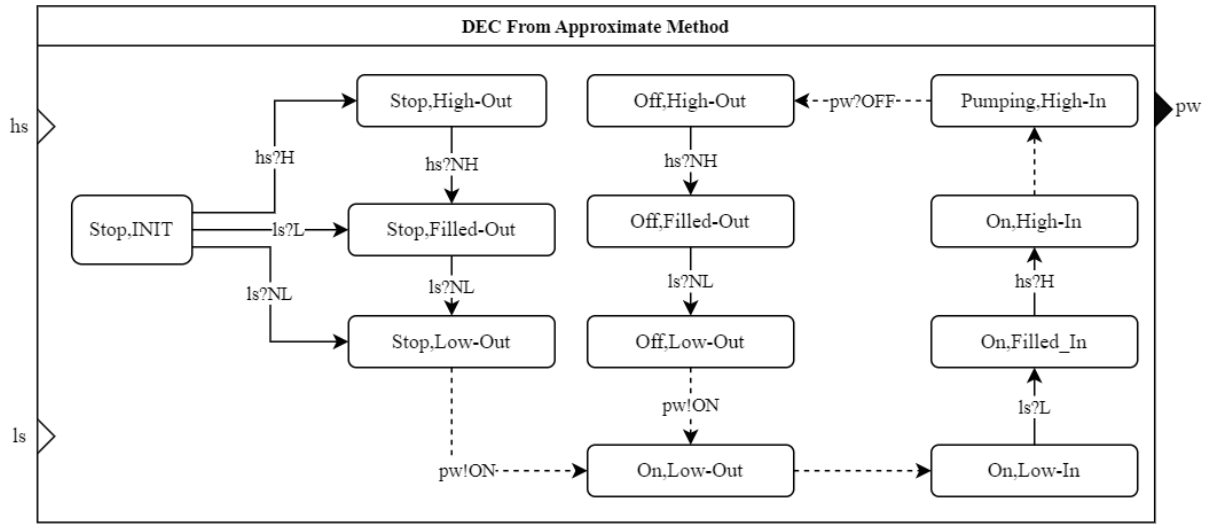

Figure 39. DEC from the modified method after removing unused states and transitions.

Figure 39 shows the obtained DEC. The combined states are arranged with the states of ‘p’ first and states of ‘t’ second. The DEC achieves the objectives by outputting ‘OFF’ to ‘p’, causing ‘p’ to transition to ‘OFF’ when ‘t’ reaches the ‘High-In’ state. Similarly, ‘p’ is set to ‘ON’ after it receives ‘ON’ from the DEC when the water tank reaches the ‘Low-Out’ state. The DEC also prevents the transition between ‘Filled-In’ and ‘Filled-Out’ states of ‘t’ by not having the output function generate ‘OFF’ or ‘ON’ when the states are ‘Filled-In’ and ‘Filled-Out’.

The controller can be further reduced as was done in the paper to obtain a minimal controller. We know that the path from ‘Low-In’ reaches ‘High-In’ and from ‘High-Out’ to ‘Low-Out’, also the output function of the DEC will not produce an output between these states. We can therefore remove the states in-between these states. Using such manual analysis, we can remove other states in the DEC. By doing so we can obtain a minimal

controller as shown in Figure 31. However, we did not make these changes to our controller for transparency, demonstrating that the method itself is producing these controllers and we are not making manual changes to make it perform as required.

5.1.3 RT-DEVS Execution of the DEC from the Approximate Method

We couple the DEC and the known model as shown in Figure 32. Using the Cadmium simulation environment, we simulate the models in software. We observe the water tank model ‘t’ behaves such that when the pump model ‘p’ outputs the event ‘ON’, the water level rises. Similarly, when ‘p’ outputs the event ‘OFF’ the water level in ‘t’ falls.

The time advance of ‘t’ remains finite throughout the simulation, thus the internal transition function of ‘t’ continuously monitors the simulated water level and assigns the current state of the tank based on its numerical value. When a state change happens in the internal transition function the output function outputs the relevant output event.

We start ‘t’ in the state ‘Filled-Out’ and ‘p’ in the state ‘Off’ and the DEC in the state ‘Stop_INIT’. The DEC quickly transitions to ‘Stop_Filled-Out’ due to the input event ‘NH’ from the coupled model. We observe that the water level declines, and ‘t’ changes its state to ‘Low-Out’. The model produces the output “NH” and “NL” through the “hs” and “ls” ports respectively. The external transition causes the DEC to change state to ‘Off_Low-Out’ and then has an internal transition to ‘On_Low-Out’ along with the output function generation event ‘ON’. This event reaches ‘p’ through the port ‘pw’. ‘p’ transitions to the ‘On’ state and generates an output event ‘ON’. Due to the internal coupling, this event triggers ‘t’ to move from a state with the ‘-Out’ suffix to a state with the ‘-In’ suffix, causing the water level to start rising in the tank.

When the water level crosses the threshold for the ‘High-In’ mark. The output function of component ‘t’ produces events ‘H’ and ‘L’ from the ‘hs’ and ‘ls’ ports respectively. The controller has an internal transition to ‘Off_High-Out’ and associated with it is an output function that outputs an “OFF” event to ‘p’. ‘p’ then transitions to the ‘Off’ state and generates an output event ‘Off’ to ‘t’. This input event to ‘t’ move ‘t’ from a state with the ‘-In’ suffix to a state with the ‘-Out’ suffix, causing the water level to start falling in the tank. This causes the water level to fall in the tank until it crosses the threshold for ‘Low-Out’ and the loop described repeats again.

We then port the model to an embedded platform using the Cadmium tool and the same hardware setup as shown in Figure 33. We execute the experiment exactly as described for the original methodology and shown in Figure 34) a-d. The results that we observe match the embedded execution results described in Section 5.1.2 and the software simulation that we described in this section. From this, we can conclude the DEC works as expected and fulfills the control objectives. If an actual water tank with sensors and a water pump with a relay switch is given. Then as before we only need to change the simulated water level with actual values of the sensors in ‘t’ and the output function of ‘p’ needs to output control signals to the relay. The DEC and the states of the models remain unchanged.

With this implementation and experiment with a model described in the literature, we show that our method works as expected and can produce a working DEC that fulfills the control objectives. In the subsequent sections, we implement our models to demonstrate that the method works for other models as well.

5.2 Supervisory Control of a Smart Building

In this section, we go through the process of defining the models in our study, comparing the extracted GSS from the cartesian product and the approximate method, defining the control objectives, applying them, and then performing controllability analysis on the result. We also show how the 'missing' transitions are identified and how we resolve them. 'missing' transitions described in Section 4.2.2.6, occur when a DEC is passivated and therefore cannot execute its internal transition function. Finally, we simulate the models coupled with the DEC using Cadmium simulation environment in software, to test if our DEC works as intended. For model execution on a Mbed-OS based embedded platform, we describe the hardware setup and the results in section 5.4.

5.2.1 Defining the Smart Building Models

We model a smart room as our plant. The plant consists of three sensors and 4 actuators forming our plant. We define four sub-models that define the states of the physical property that we want to control and model the effects of the actuator action on the plant. These four models are:

1. Room CO2 concentration model.
2. Room occupancy model.
3. Room fire alarm model.
4. Room water temperature model.

5.2.1.1 Room CO2 Concentration Model

The Room CO2 concentration model monitors the CO2 level of the room and maintains it at an acceptable level. The excess level of CO2 concentration in a room leads to poor air quality and can lead to adverse effects on human health like headaches and fatigue [92].

The room can be ventilated using vents that open and close allowing air exchange with the outside. The vents are controlled using an actuator that is activated based on the concentration of the CO2 in the room. Once the CO2 concentration rises above a certain level the vents should be opened to allow the concentration to fall. When the CO2 level falls, below a threshold, the actuators should be closed.

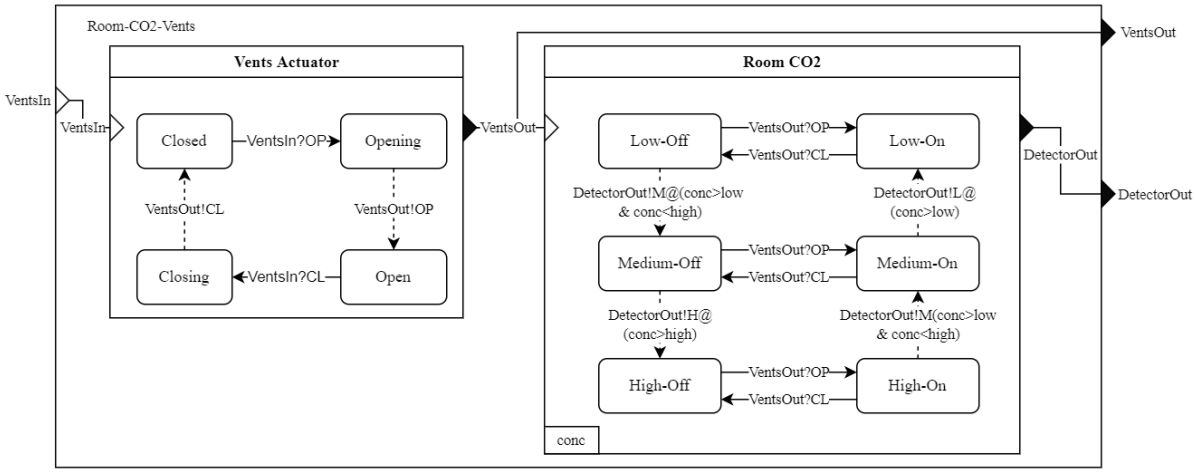

Figure 40. A coupled model consisting of vent actuator and Room CO2 atomic models.

Figure 40 shows the coupled model Room-CO2-Vents and the definitions of vents actuator and Room CO2 atomic models. The vents actuator model has four states and four transitions. The states 'Open' and 'Closed' represent the actuator being fully open or fully closed. The states 'Opening' and 'Closing' are transition states, representing the actuator moving from close to open and vice versa. The external input port 'VentsIn' accepts input

events ‘OP’ and ‘CL’. The event ‘OP’ triggers a transition from state ‘Closed’ to ‘Opening’ while the event ‘CL’ triggers a transition from the state ‘Open’ to ‘Closing’. The output port ‘VentsOut’ outputs the events ‘OP’ and ‘CL’ signifying vents are open or closed respectively.

The Room CO2 model incorporates a simulated CO2 sensor and has three different states ‘Low’, ‘Medium’, ‘High’ each signifying a different level of CO2 concentration in the room stored in the variable ‘conc’. The suffix ‘-Off’ and ‘-On’ signify if the actuator is closed or open respectively. The internal transition function checks the value of the simulated CO2 sensor and based on its numerical value selects the current state. The output function outputs ‘L’, ’M’ or ‘H’ signifying the current concentration of CO2 level through the output port ‘DetectorOut’. The input port ‘VentOut’ accepts two input events ‘OP’ and ‘CL’. ‘OP’ is a triggering external event that causes a transition moving any of the ‘Off’ states to the corresponding ‘On’ state. For example, ‘High-Off’ has an external transition to ‘High-On’ when the input is ‘OP’. Similarly, the input event ‘CL’ triggers a transition from the ‘On’ states to the corresponding ‘Off’ states.

The room is modeled with the assumption that when the vents are closed the CO2 concentration rises, and when the vents are open the CO2 concentration falls. This is because when the vents are closed the air exchange is not possible and hence due to occupants the CO2 concentration rises. When vents are open, air exchange is possible and therefore excess room CO2 can move outside and the CO2 concentration falls. Hence the model only moves from ‘High-On’ to ‘Low-On’ and from ‘Low-Off’ to ‘High-Off’.

The Top model defines the IC between the output port ‘VentsOut’ from the vents actuator component and the input port ‘VentsOut’ from the room CO2 component. The

EIC is coupled with the ‘VentsIn’ port and the EOC has two couplings, one with the ‘VentsOut’ port and the other with the ‘DetectorOut’ port.

5.2.1.2 Room Occupancy Model

The Occupancy model controls the lights of the room based on the occupancy state of the room. If a room is occupied the lights should turn on and vice versa. This requires the use of an actuator to turn the lights on and off and an occupancy sensor that detects the presence of people.

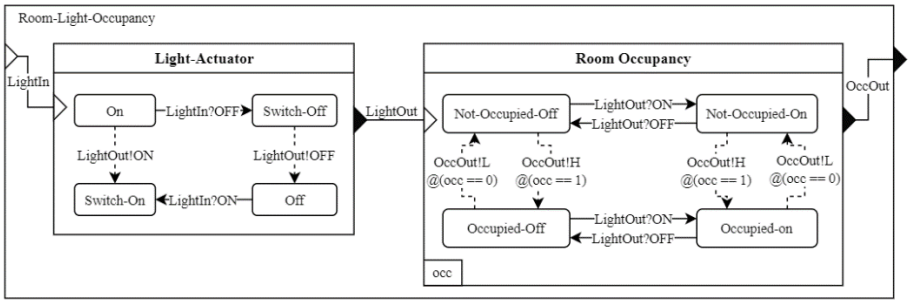

Figure 41. Room-Light-Occupancy coupled model composed of the light actuator and room occupancy atomic models.

Figure 41 shows the coupled model definition of the Room-Light-Occupancy model and the definitions of the atomic components. The Light Actuator model receives external triggering events ‘ON’ and ‘OFF’ from the input port ‘LightIn’. The state ‘On’ signifies the light has been turned on and the state ‘Off’ describes the light has been turned off. The ‘Switch-Off’ and ‘Switch-On’ states describe the model when the actuator is switching from ‘On’ → ‘Off’ and ‘Off’ → ‘On’ respectively.

The Occupancy Model receives triggering external events ‘ON’ and ‘OFF’ from the input port ‘LightOut’ and produces outputs ‘L’ and ‘H’ from output port ‘OccupancyOut’. The external transition function is executed when input is ‘ON’ changing

the ‘-Off’ states to ‘-On’ states and vice versa when the input even is ‘OFF’. The internal transition function checks the value of the sensor stored in ‘occ’, when the value is ‘1’ the state transitions to ‘Occupied’, when it is ‘0’ the state transitions to ‘Not-Occupied’. The transition from ‘Not-Occupied’ to ‘Occupied’ is accompanied by the output function producing ‘H’ and the transition from ‘Occupied’ to ‘Not-Occupied’ produces the output ‘L’.

The coupled model defines an IC between the output port ‘LightOut’ port and the input port ‘LightOut’. The EIC is defined with the ‘LightsIn’ port and the EOC has definitions with the ‘LightOut’ port and the ‘OccupancyOut’ port.

5.2.1.3 Room Fire Alarm Model

The room fire alarm model consists of a smoke detector and a fire alarm actuator. The presence of smoke in a room indicates the possibility of a fire. If the smoke concentration passes a certain threshold, then the possibility of a fire is likely, and a fire alarm should be triggered.

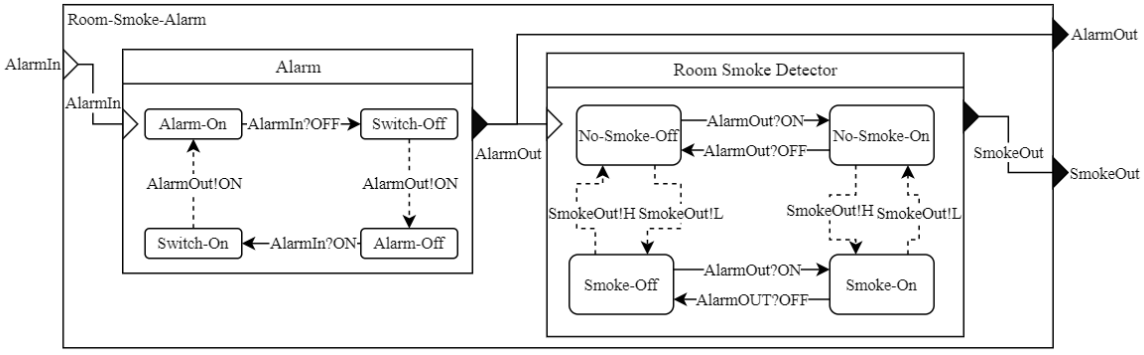

Figure 42. Fire alarm actuator and room smoke detector models.

Figure 42 shows the coupled Room-Smoke-Alarm model with the component alarm model and room smoke detector model. Like the occupancy model, the smoke

detector model incorporates the effect of a smoke detector sensor and based on its value triggers the state change ‘Smoke’ → ‘No-Smoke’ or ‘No-Smoke’ → ‘Smoke’. The output function associated with these transitions produces the output ‘L’ for low smoke concentration and ‘H’ for high smoke concentration. The suffix ‘-On’ and ‘-Off’, as before, specify if the alarm is ‘On’ or ‘Off’.

The fire alarm model receives input event ‘ON’ and ‘OFF’ from the input port ‘AlarmIn’ and these input trigger transitions ‘Alarm-Off’ → ‘Switch-On’ and ‘Alarm-On’ → ‘Switch-Off’ respectively. After a time advance duration has passed, the internal transition function moves the state from ‘Switch-On’ to ‘Off’, with the output function generating ‘OFF’ through the output port ‘AlarmOut’. If the state is ‘Switch-On’ the internal transition function moves the state to ‘Switch-Off’ with the output function generating ‘OFF’ through the output port ‘AlarmOut’.

The IC is defined between the fire alarm ‘AlarmOut’ port and the room smoke detector ‘AlarmOut’ port. The EIC is defined for the ports ‘AlarmIn’ and there are two EOCs defined, one with ‘SmokeOut’ port and the other with fire alarm ‘AlarmOut’ port.

5.2.1.4 Room Water Temperature Model

The water temperature in a room is to be 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.

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, in addition to the water temperature sensor and the boiler actuator, we also include an occupancy sensor.

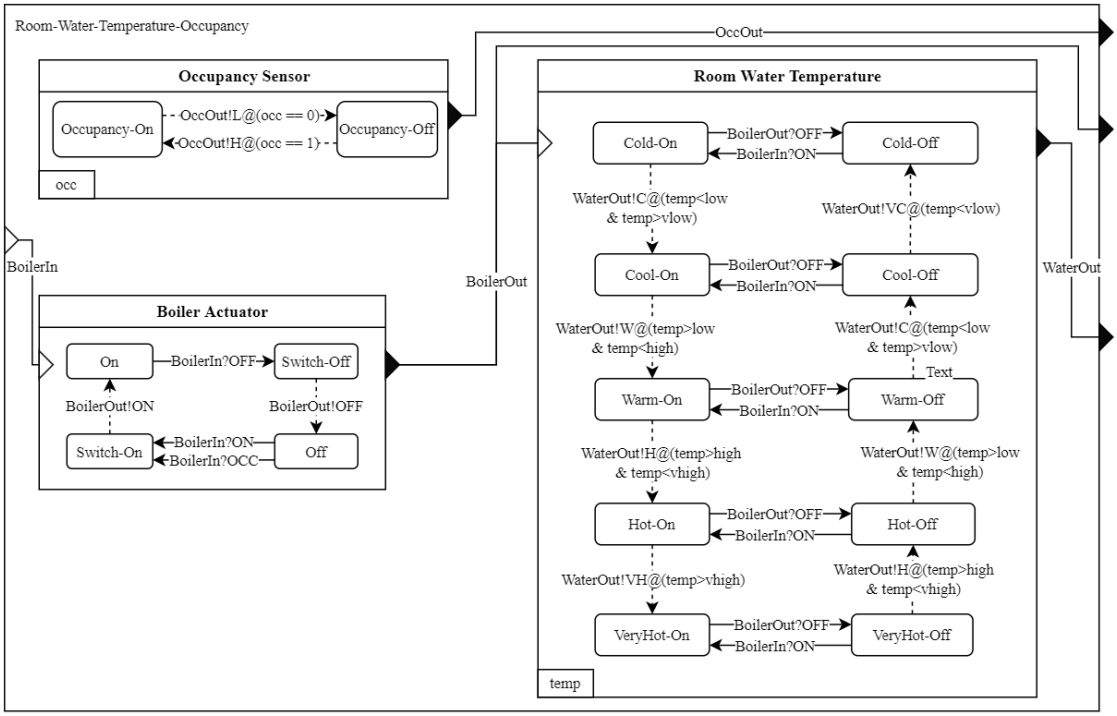

Figure 43. Boiler actuator, occupancy sensor, and the room water temperature model.

Figure 43 shows Room-Water-Temperature-Occupancy coupled model and its components. The internal transition function of the occupancy model reads the value of an occupancy sensor. Based on the sensor value the internal transition function changes the state back and forth between 'Occupancy-On' and 'Occupancy-Off'. When the state is 'Occupancy-On' the output function generates the output 'H' and when the state is 'Occupancy-Off' the output generated is 'L'.

Like the actuator models discussed in the previous section, the actuator model operates similarly. The addition, in this case, is that the transition 'Off' → 'Switch-On' can be caused by two input events both of which use the input port 'BoilerIn'. The first input

event ‘ON’ triggers the transition due to water temperature rising above the threshold value. The second input event ‘OCC’ is caused by the room being occupied, changing the state ‘Off’ → ‘Switch-On’. The internal transition function changes the state ‘Switch-On’ → ‘On’ and the output function produces the output ‘ON’ through the output port ‘BoilerOut’. Similarly, for the ‘Switch-Off’ state the internal transition function changes the state to ‘Off’ and the output function generates the output ‘OFF’ through the output port ‘BoilerOut’.

The room water temperature model has 10 states, 5 represent the water temperature ranges when the boiler is on, with the suffix ‘-On’, and the other 5 with the suffix ‘-Off’ represent the temperature range when the boiler is off. When the boiler is on the water temperature should rise and therefore the internal transitions take the water temperature from the state ‘Cold-On’ to ‘VeryHot-On’. Similarly, when the boiler is off the water temperature should fall, and hence the internal transitions move from the state ‘VeryHot-Off’ to ‘Cold-Off’. The output function outputs values for each of the water temperature ranges ‘Cold’, ‘Cool’, ‘Warm’, ‘Hot’ and ‘VeryHot’ and these are the output ‘VL’, ‘L’, ‘M’, ‘H’ and ‘VH’ respectively. The internal transition function checks the value of the sensor stored in ‘temp’ and changes the states based on this value. The external transition switches between the ‘-Off’ and ‘-On’ versions of the temperature states based on the input.

The IC coupling is defined between the boiler actuator output port ‘BoilerOut’ and the room water temperature input port ‘BoilerOut’. The EIC is defined for the ‘BoilerIn’ port. The EOC is defined for the room water temperature model ‘WaterOut’ port and the occupancy sensor model ‘OccOut’ port.

5.2.2 Comparison between Cartesian Product and the Modified Method

Once we have defined the models of the plant, we need to extract the GSS from the coupled models. We extract the GSS using first the cartesian product and then we use our method. To compute the GSS using the cartesian product we use the NetworkX [90] Python [93] package to compute the cartesian product. The package provides tools to create graph structures and perform operations on the graphs. The package is well tested and therefore we can be reasonably certain that the results obtained from this package are accurate. We define each atomic as a 'directional multigraph' in the code as the transitions are directional and there are multiple transitions between two states. We order 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 as described in Section 5.1.2, by passing it through a custom python script. The script returns a text file that lists the vertices, and underneath each vertex is a list of associated edges that are transitions from this combined state. We note the number of states and transitions for each of the models.

Table 1 lists the results of our experiments. The first column, starting from the left, lists the model names. The second and third columns list the resulting combined states and transitions from the cartesian product. The fourth and fifth columns list the resulting combined states and transitions from the approximate method. Finally, the seventh column is the percentage difference between the total number of transitions from the cartesian product and the approximate method.

Table 1. States and transitions of the resultant GSS from 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	6.58
Room Occupancy	16	48	16	40	16.67
Room Smoke Concentration	16	48	16	40	16,67
Room CO2 concentration	24	80	24	68	17.65

The results show the number of states for the cartesian method, and the approximate method is the same in all cases and thus we can be assured our method is not missing any possible combined states. For the room occupancy and room smoke concentration models the percentage difference is approximately 16.67 percent. This is because they generally have the same structure and the same number of states and hence the percentage difference is the same. The room CO2 concentration model has a slightly larger difference at 17.67%, while the overall structure of this model in terms of couplings of the TOP model is similar to the previous two models, the room CO2 atomic model is larger, hence we see a slight increase. Finally, for the room water temperature model, we observe that our method produces approximately a 6.58% reduction in the number of transitions.

The reduction in the number of transitions is because our method can reject some transitions that are not possible to occur based on the coupled structure of the model. The cartesian product on the other hand still computes all the possible transitions and therefore performs worse. A reduction of 6.58% is significant and in larger cases can cause a large

reduction in the total number of transitions, making the next steps easier and more manageable to complete.

Our method cannot generate more transitions than the cartesian product. This is because the method generates the same number of combined states in the GSS as the cartesian product and restricts the addition of transition only between combined states with only one state change between them. This is similar to how the cartesian product operates where it can only add a transition between two vertices if they only have one state change between them. Hence the approximate method cannot add more transitions than the cartesian product.

5.2.3 Obtaining the DEC from the GSS

We can extract the controller for the room model in two ways. We can either obtain a monolithic supervisor that would control all the sub-models, or we can use a modular implementation where we break the system up into smaller parts where each part has one control loop to deal with. Figure 44 shows the monolithic supervisor case.

In Figure 44, we see that all the coupled models are connected to one supervisor that then generates control outputs for the three actuators. We can separate the models into smaller modular supervisors because in this case, the models have multiple control loops that do not overlap as shown in Figure 45.

In this case, each separate control loop has its own DEC that monitors the state of the model and issues commands to the actuator. The room CO2 DEC supervises the CO2 concentration in the room and adjusts as necessary. The occupancy DEC monitors the occupancy status and adjusts the lights of the room. Finally, the fire alarm DEC monitors

the smoke concentration of the room and triggers the fire alarm. Each of the DECs works in parallel and fulfills the control requirements of the modeler.

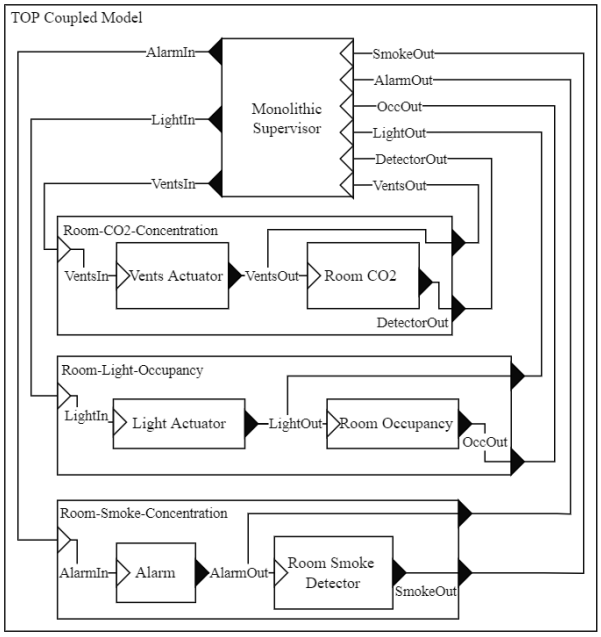

Figure 44. Monolithic supervisory control model.

The reduction in the overall complexity in terms of the states and transitions by using the modular supervisor approach, in this case, leads us to adopt this approach. We calculate the GSS of the monolithic case and find it has 1536 states and 9088 transitions using the approximate method. In comparison, the three modular GSS have a combined 112 states and 364 transitions. We have therefore chosen to use the modular approach to obtain supervisory controllers.

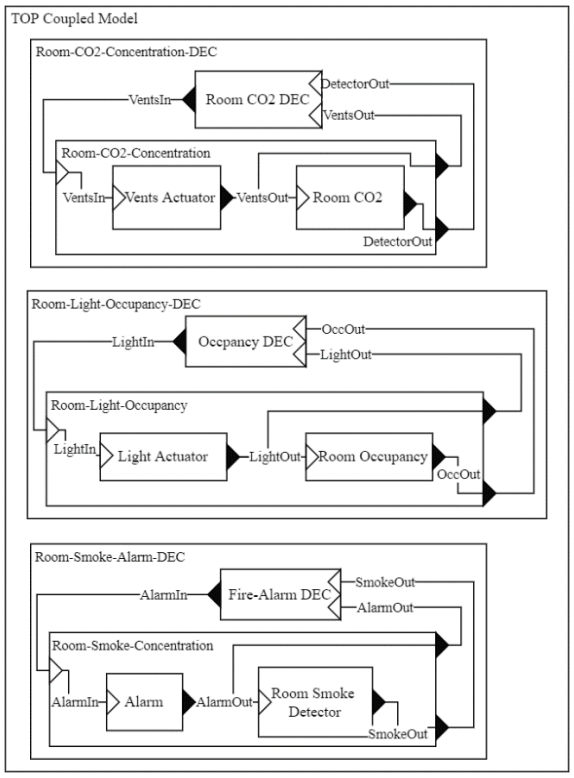

Figure 45. Modular supervisors for the room conditioning model.

5.2.3.1 Applying the Control Objectives and Performing Controllability Analysis

In this section, we define and apply the control objectives to the obtained GSS and perform controllability analysis on the results. For each of the four models, we describe the control specifications that create a controllable loop in the GSS. We define the control specifications in a text file that our program can read and interpret. In Listing 3, the specifications for the Room CO2 model are given. The format used here is the same as in Section 5.1.2 and described for Listing 2.

```
path,CO2.low-closed,vents.closed,CO2.high-open,vents.open
no_state,CO2.low-closed,vents.open
no_state,CO2.low-closed,vents.opening
no_state,CO2.high-open,vents.closed
no_state,CO2.high-open,vents.closing
```

```
no_transition,CO2.medium-closed,?,CO2.medium-open,?
no_transition,CO2.medium-open,?,CO2.medium-closed,?
```

Listing 3. Control specifications of room CO2 concentration model.

We named the room CO2 model ‘CO2’, and the actuator is named as ‘vents’. The path defined, i.e. our desired control loop, for the supervisor should be able to move from a low concentration of CO2, and vents being ‘Closed’ to high concentrations of CO2 with the vents being ‘Open’.

‘no_state’ defines problematic combined states. The combined states ‘CO2.low-closed, vents.opening’ and ‘CO2.high-open, vents.closing’ disrupt the control loop by allowing the actuator to open, when CO2 concentration is ‘Low’ and allowing the vents to close when the CO2 concentration is ‘High’ respectively. The remaining two combined states cause the vents to open and be in state ‘Open’ when the room CO2 concentration is ‘Low’, and its state is ‘Low-Closed’. The two states are contradictory to each other and therefore should not be in the GSS. Similarly, when CO2 concentration is ‘High’, and its state is ‘High-Open’ the state of vents cannot be ‘Closed’. Finally, we disallow transitions between the states ‘CO2.medium-closed’ and ‘CO2.medium-open’ regardless of the state of the vents actuator to prevent the controller from switching the actuator back and forth between on and off when CO2 concentration is ‘Medium’.

```
path,smoke.no_smoke-off,alarm.alarm_off,smoke.smoke-
on,alarm.alarm_on
no_state,smoke.no_smoke-off,alarm.alarm_on
no_state,smoke.smoke-on,alarm.alarm_off
```

Listing 4. Control specifications of room smoke concentration model.

Listing 4 states the control specification for the room smoke concentration model. The desired control path is from the fire alarm being ‘Off’ when there is ‘No-Smoke’ to

the fire alarm being ‘On’ when there is ‘Smoke’. The no_state rule removes the illegal states where the alarm is ‘On’ when there is ‘No-Smoke’ or when the alarm is ‘Off’ when there is ‘Smoke’.

```
path,occ.noc_off,light.off,occ.oc_on,light.on
no_state,occ.noc_off,light.on
no_state,occ.oc_on,light.off
```

Listing 5. Control specifications of room occupancy model.

The desired control loop for the room occupancy model requires the plant to keep the lights ‘Off’ when the sensor is in the state ‘Not-Occupied’ and turn the lights ‘On’ when there are occupants as shown in Listing 5. We also remove the combined states where the lights would be ‘On’ when the occupancy sensor is ‘Not-Occupied’ and the state where lights would be ‘Off’ when it is in state ‘Occupied’.

```
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,?
```

Listing 6. Control specifications of room water temperature model.

The water temperature model needs to keep the water temperature between ‘Cool’ and ‘Hot’ temperature ranges and turn the boiler ‘On’ when there is an occupant in the room. Listing 6 shows the control loop that achieves those goals, 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’ the control loop can go back to the ‘water.cool-off, boi.off’ state by turning the boiler ‘Off’ and letting the water temperature return to ‘Cool’. The states ‘Cold-On’ and ‘Veryhot-Off’ are completely

removed as they are illegal states and shouldn't occur. The states 'water.cold-off, boi.on?' and the states 'water.cold-off, boi.switch_on' are removed to force the DEC 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'.

From this point on, we will focus on the room CO2 concentration model, room water temperature model, and describe the process in detail for these models for the sake of brevity. These two models are larger and have more complexity than the remaining two models and should be looked at in detail. What we describe for the CO2 concentration model and room water temperature model apply to the remaining two models.

We apply the rules of controllability to the reduced GSS that we obtain after applying the control specifications. For the room CO2 model, the analysis showed that GSS only contains strong or very-strongly controllable transitions. This means that we can synthesize a DEC using this reduced GSS by applying inverse DEVS transformation, without making any changes. A portion of the controllability analysis for the room water temperature is listed in Listing 7.

```
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,
```

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

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

Listing 7. Weak transitions of the room water temperature model.

The transitions in Listing 7 are weak transitions, arranged here 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. The third and fourth triplets are similar with the difference being the occupancy sensor is ‘Off’ and signifies the room is unoccupied. From this analysis, the control loop is unlikely to transition to those states and therefore we can apply the inverse DEVS to the reduced model.

5.2.3.2 Application of the Inverse DEVS Transform and Finding Missing Transitions

The Inverse DEVS transformation can now be applied to the reduced GSS after we have carried out controllability analysis and confirmed our control path would not stray due to weak transitions.

We start with the reduced GSS of the room CO2 concentration model. The application of the Inverse DEVS transformation is straightforward. The input ports of the coupled model become output ports and vice versa and the internal transitions switch to external transitions with their weights preserved.

We analyze the coupled model and observe that the transitions 'Opening' → 'Open' and 'Closing' → 'Closed' from the vents actuator model occur as part of the internal coupling with the room CO2 concentration model. In this case, the external transitions 'High-Off' → 'High-On' and 'Low-On' → 'Low-Off' from the room CO2 model can be missed because the DEC would be passivated. We, therefore, change the external transition of the DEC such that the transition becomes 'High-Off, Vents-Opening' → 'High-On, Vents-Open' and 'Low-On, Vents-Closing' → 'Low-Off, Vents-Closed'.

In Figure 46 the resultant DEC is displayed for the Room CO2 concentration model. The order of the combined states is 'Room CO2 Concentration state, Vents actuator state'. From the arbitrary starting point of 'Low-Off, Vents-Closed', the DEC progresses through the states due to a series of triggering external events from the port 'DetectorOut' until we reach the state 'High-Off, Vents-Closed'. The external events are coming from the room CO2 concentration model.

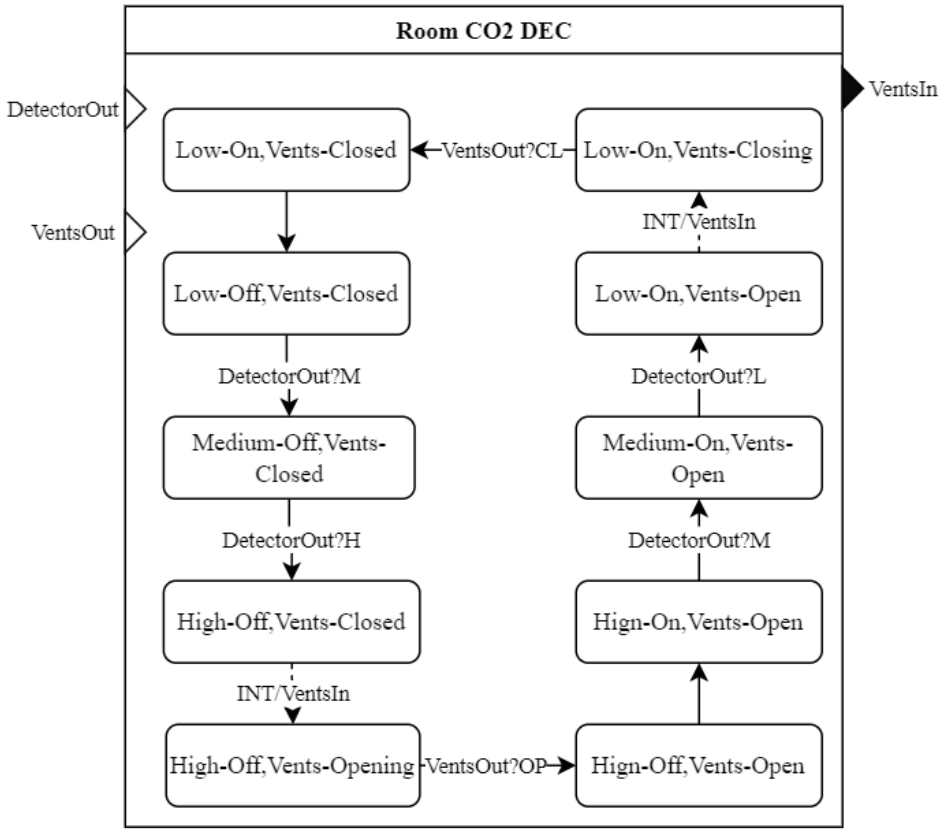

Figure 46. DEC for the Room CO2 concentration model.

At this state, an internal transition moves the DEC to 'High-Off, Vents-Opening' and an output is generated and sent to the vents actuator model through the 'VentsIn' port. opening the vents. The controller then waits for the actuator to confirm it has opened through a triggering external event from the 'VentsOut' port. The controller moves to the 'High-Off, Vents-Open' state, at this instant we know that the room CO2 concentration model is in the 'High-On' state, therefore we move the state to 'High-On, Vents-Open'. This transition is shown as an external transition with no input port and event for clarity and is done in the same external transition, This can be replaced by one external transition from 'High-Off, Vents-Opening' to 'High-On, Vents-Open'. From here the controller progresses to the 'Low-On, Vents-Open' state because of decreasing CO2 concentration in

the room. The DEC then signals the vents to shut-off, causing the vents actuator model to move to ‘Low-On, Vents-Closing’ as a result of an internal transition. As before, the room CO2 concentration model has transitioned to ‘Low-Off’ when the actuator model signals it is ‘Closed’, we, therefore, move the state of the DEC to ‘Low-Off, Vents-Closed’. This is the starting state and from here the DEC and room model can go through the loop again.

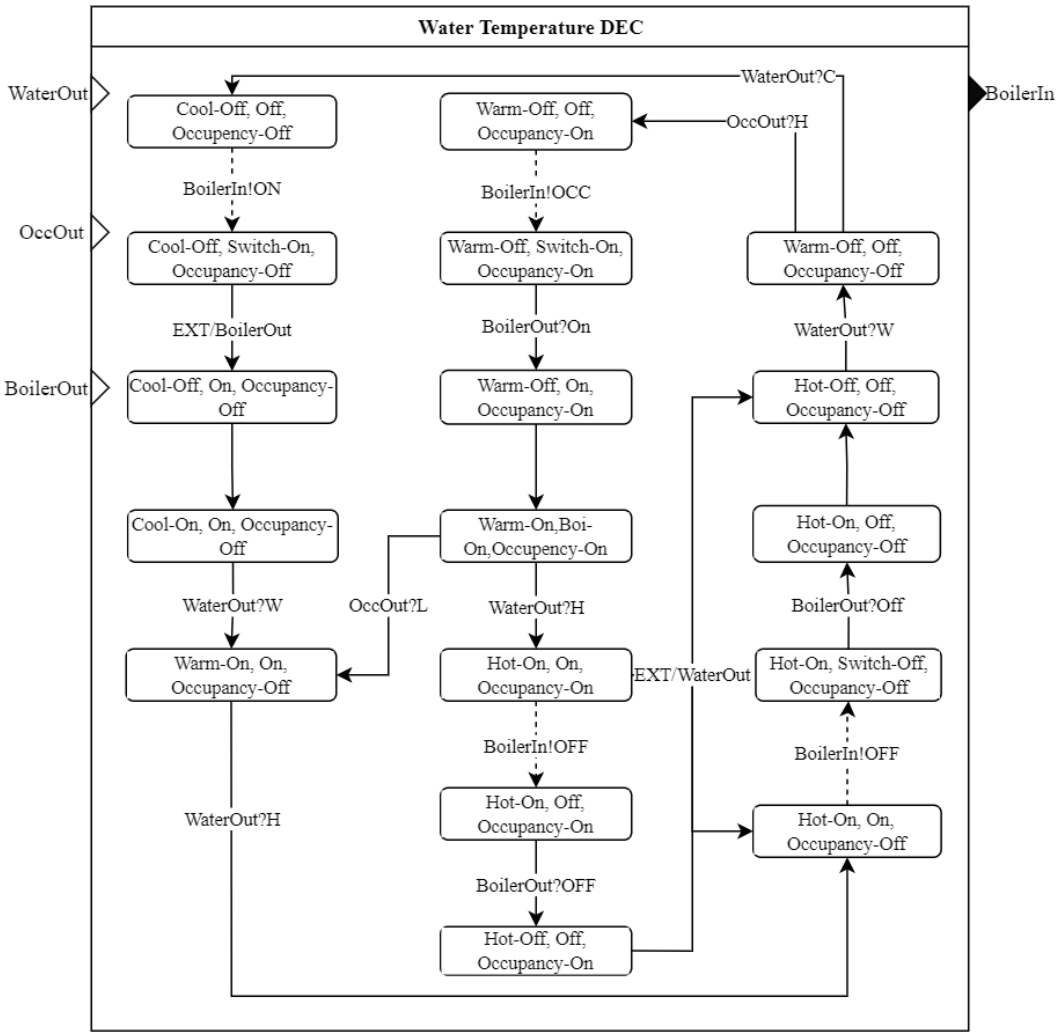

Figure 47. DEC for the Room Water Temperature model.

Figure 47 shows the DEC obtained from the reduced GSS after applying the Inverse DEVS transformation and resolving the missing transitions. The order of combined states

is ‘Room Water Temperature state, Boiler Actuator state, Occupancy Sensor state’. 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 DEC is in the state ‘Cool-Off, Off, Occupancy-Off’ and outputs an ‘ON’ event to the actuator model and moves to the state ‘Cool-Off, Switch-On, Occupancy-Off’. The actuator model executes its external transition function and changes to the state ‘Switch-On’, this is followed by the execution of the internal transition function, moving the actuator to state ‘On’ and the output function generating ‘On’. The time advance function is set to passivate the actuator. The room water temperature model moves to the state ‘Cool-On’.

However, the DEC is in the ‘Cool-Off, On, Occupancy-Off’ state after it receives the input event ‘On’ from the ‘BoilerOut’ port and is passivated. This transition is therefore changed, and we add the state change ‘Cool-Off, On, Occupancy-Off’ → ‘Cool-On, On, Occupancy-Off’ and set the time advance to the time advance of the room water temperature model. This transition is shown in Figure 47 as a separate external transition with no port or input event so that this modification is marked. In the implementation, there is only one external transition that changes the state ‘Cool-Off, Switch-On, Occupancy-Off’ → ‘Cool-On, On, Occupancy-Off’.

Similarly, the other two missing transitions are ‘Hot-On, Switch-Off, Occupancy-Off’ → ‘Hot-Off, Off, Occupancy-Off’, and ‘Warm-Off, Switch-On, Occupancy-On’ → ‘Warm-On, On, Occupancy-On’. The first scenario is the actuator turning the boiler ‘Off’ when the room water temperature is ‘High’, and the second scenario is the actuator turning the boiler ‘On’ because the room has been occupied.

The DEC starts from the ‘Cool-Off, Off, Occupancy-Off’ state and instructs the boiler to turn ‘On’ by generating the output ‘On’ through the output port ‘BoilerIn’. The boiler turns on and the water temperature in the room rises. The DEC keeps up with the state of the room water temperature model from the input events it receives from the water temperature model through the input port ‘WaterOut’. When the DEC reaches the state ‘Hot-On, On, Occupancy-Off’, the DEC generates the output ‘Off’ that causes the boiler actuator model to move to the ‘Off’ state. The DEC moves to the state ‘Hot-Off, Off, Occupancy-Off’. The water temperature should now start to fall, and the DEC once again uses the inputs from the water temperature model to keep up with the states of the water temperature model. The DEC moves from ‘Hot-Off, Off, Occupancy-Off’ to eventually ‘Cool-Off, Off, Occupancy-Off’, and the cycle repeats.

If the DEC is in the state ‘Warm-Off, Off, Occupancy-Off’ and it receives an input from the occupancy sensor model ‘H’ through the ‘OccOut’ input port. The DEC moves to state ‘Warm-Off, Off, Occupancy-On’ and then outputs an ‘On’ event to the boiler actuator model. Causing it to eventually move to the ‘On’ state and causing the water temperature to rise again. The DEC will not instruct the boiler actuator to turn ‘Off’ until the water temperature reaches ‘Hot’ again. If the room is occupied when the water temperature model reaches the ‘Warm-Off’ state then the DEC causes the boiler actuator to turn on again otherwise the water temperature is allowed to fall to ‘Cool’ and the DEC reaches the starting state ‘Cool-Off, Off, Occupancy-Off’.

5.2.3.3 Performing Simulations in Software of the Case Study

The models and their DECs discussed in this chapter were individually tested for proper working. Following this, a simulation was carried out with all the models together in a

larger model as shown in Figure 45. This larger simulation is discussed in Section 5.4 where the models are ported and executed onto an embedded platform and the tests repeated. Here we focus on the individual simulation and testing of the room CO2 concentration model and the room water temperature model using the Cadmium simulation tool.

5.2.3.3.1 Simulating the Room CO2 Concentration Model

The room CO2 concentration model, vents actuator model, and the DEC are passed through a scenario where the CO2 concentration rises when the vents are closed. When the vents are opened the CO2 concentration in the room begins to fall. This is done by simulating the value of the CO2 concentration 'conc' in the room CO2 concentration model. When the room CO2 concentration model is in the '-On' states the value of 'conc' is decremented after each time advance, and the CO2 concentration model is in the '-Off' states the value is incremented. The role of the DEC is then to instruct the vents actuator to turn on and open the vents when concentration is 'High' and close the vents when the concentration is 'Low'.

The vents are started in the states 'Closed', the room CO2 concentration model is started in the state 'Low-Off' and the DEC is started in the state 'Low-Off, Vents-Closed'. The room CO2 concentration model increments the CO2 concentration value as it is in the '-Off' state. The CO2 concentration rises and crosses the threshold for 'Medium', set at 30 in this case. The room concentration model outputs 'M' through the 'DetectorOut' port. We observe the DEC receives this as a triggering input event through port 'DetectorOut' and transitions to 'Medium-Off, Vents-Closed' state. The DEC has no internal transitions that will move from this state and so the DEC remains in this state. The value of 'conc'

continues to rise until it reaches ‘70’. The threshold for ‘High’ has been crossed and the DEC receives an input that causes it to move to ‘High-Off, Vents-Closed’. The internal transition function of the DEC moves the state to ‘High-Off, Vents-Opening’ and the output function simultaneously outputs ‘OP’ through the ‘VentsIn’ port.

The actuator model receives ‘OP’ as an input and we observe it transitions to the ‘Open’ state and outputs the ‘OP’ through the ‘VentsOut’ state. Both the room CO2 concentration model and the DEC receives this output. We observe the DEC successfully transitions to the ‘High-On, Vents-Open’ state. The ‘conc’ value now decrements as the vents are open. The DEC keeps up with the concentration of CO2 in the room from the inputs through the ‘DetectorOut’ port until the ‘Low-On, Vents-Open’ state is reached. The DEC outputs ‘CL’ to the vents actuator causing it to close the vents. After receiving the input ‘CL’ from the vents actuator model the DEC returns to the starting state ‘Low-Off, Vents-Closed’. We observe all of the models have returned to their starting states and the cycle repeats indefinitely.

The DEC was observed to command the vents actuator to open the vents when the concentration was ‘High’ and close the vents when the concentration was ‘Low’ as required by the control objectives. We also observe the DEC does not cause the vents actuator model to open or close when the room CO2 concentration was ‘Medium’, as this was explicitly prohibited in the control objectives. We, therefore, conclude the DEC meets the requirements of the control objectives and is working properly.

5.2.3.3.2 Simulating the Room Water Temperature Model

The room water temperature model, boiler actuator model, occupancy model, and the DEC are passed through a testing scenario where the 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 'Occupancy-Off' states at fixed intervals. The room water temperature model simulates the water temperature value of the sensor by using a 'temp' variable. This is incremented when the model is in one of the '-On' states and decrements the value if it is in one of the '-Off' states. The interval for the occupancy sensor model in this scenario was set to 10 time advance of the occupancy sensor model. The DEC must keep the water temperature between 'Warm' and 'Hot' and must not go to 'Cold' and 'VeryHot' states. It also needs to adequately react to inputs from the occupancy model.

The models are started in the states 'Cool-Off', 'Off' and 'Occupancy-Off' for the room water temperature, boiler actuator, and occupancy sensor models respectively. The DEC is started in the state 'Cool-Off, Off, Occupancy-Off'. The initial value for 'temp' is set at 30. When the simulation is started the water temperature starts to fall as the boiler is 'Off'. The DEC is observed to output 'ON' through the 'BoilerIn' port and moves to the state 'Cool-Off, Switch-On, Occupancy-Off'. This is received by the input port 'BoilerIn' of the boiler actuator model causing it to turn 'On' after an external and internal transition. The boiler actuator model outputs 'ON' through the port 'BoilerOut'. The room water temperature model receives this as an input and moves to the state 'Cool-On'. The DEC upon receiving this input moves to the state 'Cool-On, On, Occupancy-Off'. We observe

the water temperature begins to rise as the ‘temp’ variable is incremented now due to the model being in one of the ‘-On’ states.

After some time, the occupancy sensor model changes its state to ‘Occupancy-On’ and outputs ‘H’ through the ‘OccOut’ port. The DEC receives this as an input 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 DEC changes its state 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 DEC 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 DEC moves to the state ‘Hot-Off, Off, Occupancy-Off’.

During this time the occupancy model keeps switching between ‘Occupied-On’ and ‘Occupied-Off’ states, however, the DEC does not signal the boiler to turn on and only switches between ‘Occupied-On’ and ‘Occupied-Off’ state to keep in sync with the occupancy sensor model.

The DEC is observed to move from the state ‘Hot-Off, Off, Occupancy-On’ → ‘Warm-Off, Off, Occupancy-On’. This time the DEC generates an output ‘OCC’ through the ‘BoilerIn’ output port. 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 DEC keeps the water temperature between ‘Cool’ and ‘Hot’ temperature ranges and does not let the water temperature reach ‘VeryHot’ or ‘Cold’ thresholds. The DEC can keep in sync with the states of the models. When the occupancy sensor indicates the room is occupied and the boiler is off. The DEC signals the boiler to turn ‘on’. The boiler is kept on until the water temperature reaches ‘Hot’ regardless of whether the room becomes unoccupied. This behavior is per the control objectives set and we can conclude the DEC is working as intended.

5.3 Using the Hybrid Layer to Model

In this section, we implement a room temperature model whose setpoints are controlled by a human operator. The room temperature is controlled by an AC unit that turns on and off based on the room temperature. The human operator can change the temperature at which the AC unit can turn on and off. Two models are implemented, one with the hybrid layer and one without the hybrid layer. We will focus on the steps to implement the hybrid layer to reduce the complexity introduced by the human operator. The steps to obtain the DEC have been explained exhaustively in the previous sections and therefore we will omit some steps from that process. For both models, the control objective is to keep the room temperature between the ‘Cool and ‘Hot’ temperature ranges.

5.3.1 Non-Hybrid Room Temperature Model

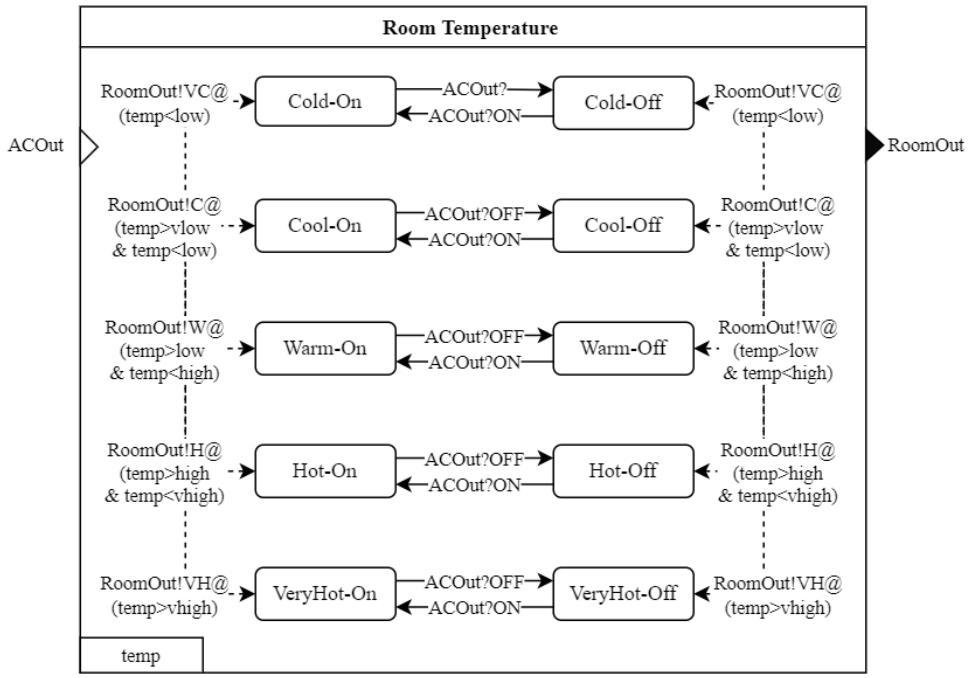

Figure 48. Room Temperature model without hybrid layer.

Figure 48 shows the room temperature atomic model without the hybrid layer. As we discussed in Section 4.6.1, we model the actions of the human model as a sensor triggering internal transitions. We do this by defining four different sets of ranges for the temperature threshold values and switching between the sets to change the setpoints of the model. We see due to the effect of changing the setpoints of the system, all temperature ranges are linked together by internal transitions. When the room temperature model is in the '-On' states the 'temp' variable is decremented and when the AC is in the '-Off' states the 'temp' variable is incremented. This is because the room temperature must rise when the AC is 'Off' and fall when the AC is 'ON'.

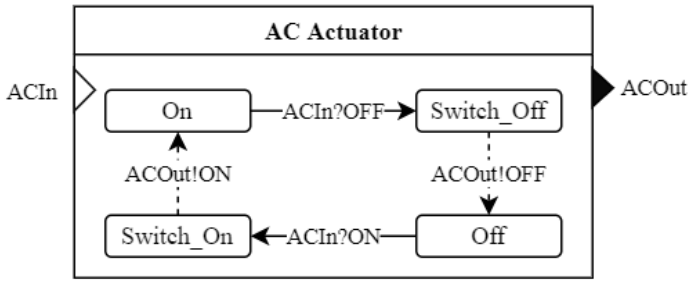

Figure 49. AC actuator model

The AC actuator model shown in Figure 49 models the action of an actuator and outputs ‘ON’ or ‘OFF’ through the ‘ACOut’ port when it is in the state ‘On’ or ‘Off respectively. The AC actuator is turned ‘Off’ when it receives an input ‘OFF’ from the ‘ACIn’ port and ‘On’ when it receives ‘ON’ from the ‘ACIn’ port.

We defined the control objectives such that the AC should be turned ‘On’ when the room temperature is ‘Hot’ and ‘Off’ when the temperature is ‘Cold’. There are no forbidden states as the change of the setpoint can cause the temperature to end up in the ‘VeryHot’ or ‘Cold’ temperature range. Passing the models through the steps described in this and the previous section we obtain a DEC that has 24 states and 77 transitions. It is not possible to draw a diagram for this DEC, however, we show some of the transitions of the DEC in Figure 50.

Figure 50 shows the DEC model transitions from the combined ‘Warm-Off, AC-Off’ state. From this state, the DEC can move to any of the four states shown in the figure. This is because a change of setpoint can cause the temperature value to land in any one of the temperature ranges. If the room temperature model outputs ‘H’ or ‘VH’ due to temperature rising or a change of setpoint, the DEC moves to the corresponding state.

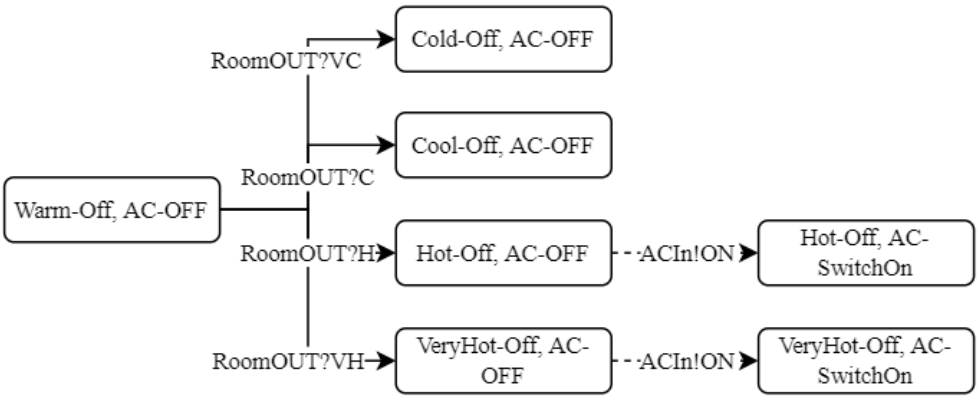

Figure 50. Selected transitions from the DEC for the room temperature model.

From both 'Hot-Off, AC-Off' and 'VeryHot-Off, AC-Off' states the DEC signals the AC to turn on so that the room temperature can start to decrease. It is important to mention the four states in the middle of the diagram are all interconnected with external transitions linked to the 'RoomOut' port, however, we have chosen not to show these transitions to keep the figure uncluttered and clear.

5.3.2 Hybrid Layer Room Temperature Model

The hybrid layer is now utilized to create an abstract room temperature model that will be passed through the methodology to obtain a DEC. This abstract room temperature model will have fewer transitions and will therefore produce a less complex DEC. The first step to creating an abstract temperature model is to make an accurate model with the human interaction included. This model is the model that we defined in the previous section and is shown in Figure 48.

From this model, we identify the human-caused transitions. There are 40 human-caused transitions between the different temperature ranges. We analyze the model to see if there are alternative paths in the model between these states. We observe that the states with the '-On' suffix have internal transitions between them caused by the room

temperature sensor changing the ‘temp’ value. Similarly for the states with the ‘-Off’ suffix the room temperature sensor can cause the internal transitions to trigger.

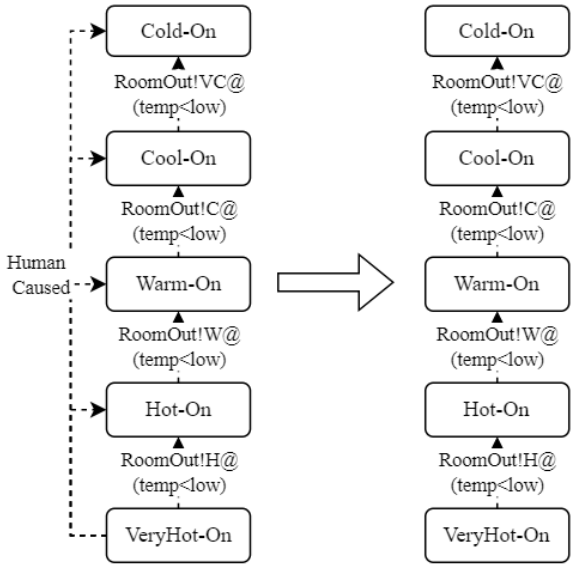

Figure 51. Removing Human-caused transitions from the ‘VeryHot-On’ state.

Figure 51 shows the process by using the ‘VeryHot-On’ state as an example. We can see it has human-caused transition to all other ‘-On’ states. The human-caused transitions from the other states are not shown. The transition ‘VeryHot-On’ → ‘Hot-On’ has an alternative path due to the change in ‘temp’ value from the sensor. Therefore, we can remove the human-caused transition for this case. Similarly, ‘VeryHot-On’ → ‘Warm-On’ has an alternative path through the transition ‘VeryHot-On’ → ‘Hot-On’ → ‘Warm-On’ and we can remove this human-caused transition as well. Analyzing the remaining transitions and removing the human-caused transitions we get the path on the right, where the transitions move from ‘VeryHot-On’ → ‘Hot-On’ → ‘Warm-On’ → ‘Cool-On’ → ‘Cold-On’. We repeat this analysis for the remaining states to obtain our abstract room temperature model.

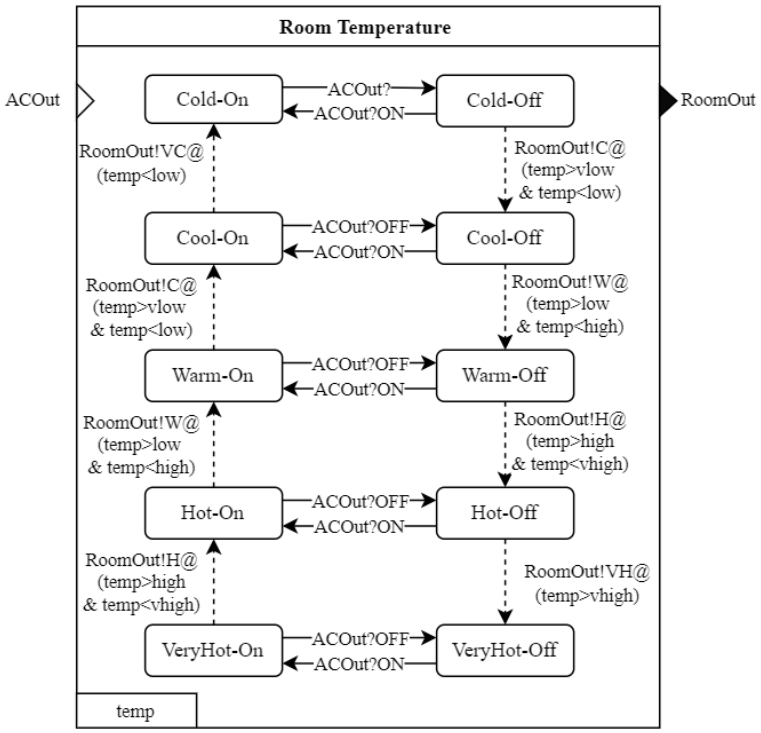

Figure 52. Abstract room temperature model using the Hybrid Layer.

Figure 52 shows the abstract room temperature model that is obtained after removing the human-caused transitions. This model has a unidirectional loop as the states only have a forward path between them. The abstract model can be made bidirectional by adding the sensor transition in the backward direction. However, since this is a room that is cooled by an AC, the expected behavior of the model should decrease the room temperature from the 'VeryHot-On' state as the AC is actively cooling the room. Similarly, from the 'Cold-Off' state the room temperature should increase as the AC is off. If the model deviates from this behavior, then either our model is incorrect or there is a fault in our implementation that needs to be fixed. The external transitions associated with the port 'ACOut' are left as is and these inform the room model if the AC is on or off.

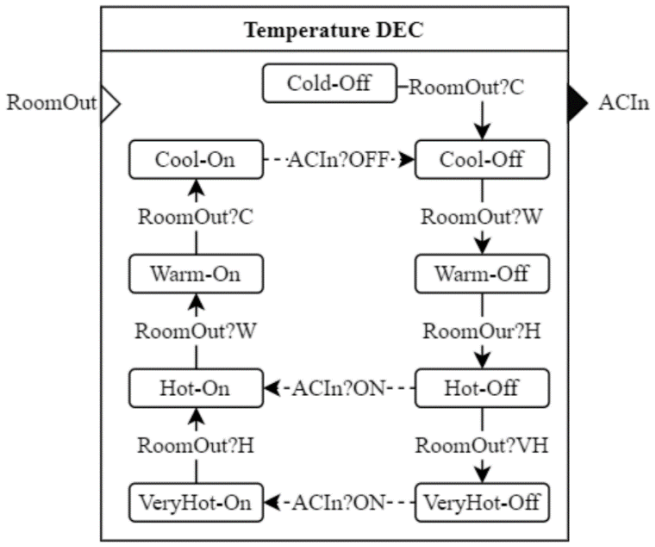

Figure 53. A DEC obtained, for the abstract room temperature model using the hybrid layer.

Figure 53 shows the DEC obtained after using the abstract model of the room temperature model. This DEC is much simpler and only has 9 states and 11 transitions. Since we are using the hybrid layer the AC actuator is also hidden from the DEC and the hybrid layer relays the commands of the DEC to the AC actuator.

5.3.3 Implementing and Simulating the Models in Software

In this section, we implement the models in Cadmium and simulate them as well as discuss the hybrid layer implementation. The simulation is carried out in software and tests the scenario where the model is allowed to complete one control loop without changing the set-points. Once this is done the set-points are changed to observe the behavior of the DEC to the changing set-point. An implementation of the models on embedded hardware is done in Section 5.4.

5.3.3.1 Implementation of Non-Hybrid Model

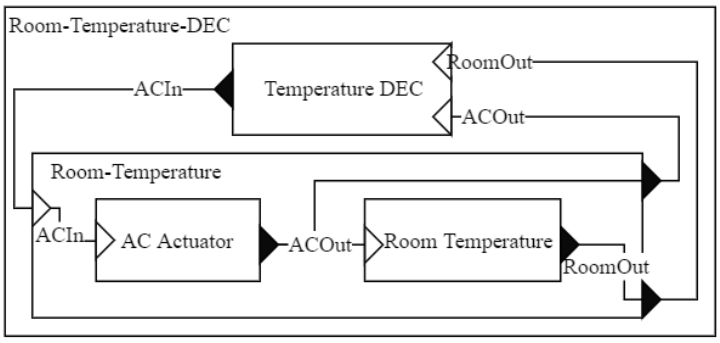

Figure 54. A coupled model of the DEC with the room temperature and AC models.

Figure 54 shows the way the DEC is coupled with the room temperature and the AC model for simulation and execution on an embedded platform. The DEC input ports 'RoomOut' and 'ACOut' are coupled with the output port 'RoomOut' from the room temperature model and the 'ACOut' port of the AC actuator model. The output port 'ACIn' of the DEC is coupled with the input port 'ACIn' of the AC Actuator. The DEC monitors the room temperature through the inputs from the room temperature model and outputs commands to the AC actuator model through the output port if needed.

The models are implemented in the Cadmium tool such that the room temperature model increments the 'temp' variable when it is in the '-Off' states and decrements 'temp' when it is in the '-On' states. The human interaction with the system is implemented by having four different sets of temperature ranges and having a variable whose value switches between them. The first set sets the thresholds at '20' and '60' for 'Cold' and 'Warm' respectively, the remaining thresholds are defined at +20 from the previous temperature threshold, so '40', '80', '100' are the thresholds for 'Cool', 'Hot' and 'VeryHot' respectively. The ranges are altered after 200 time advances of the room temperature model

have passed. This is to allow the testing of the room temperature model and the DEC without interference from the human operator.

The DEC is initially set to the state ‘Cool-Off, AC-Off’ and the room temperature model and AC actuator model are set to the state ‘Cold-Off’ and ‘AC-Off’ respectively. The ‘temp’ variable is set to ‘0’. When the simulation is started the DEC transitions to the ‘Cold-Off, AC-Off’ state as the room temperature model outputs ‘VL’ to the DEC. The room temperature model is observed to increment the ‘temp’ after every time advance and at ‘20’ the model changes its state to ‘Cool’ and outputs ‘C’. The DEC receives this output and changes its state to ‘Cool-Off, AC-Off’. The room temperature and DEC states remain in sync as the ‘temp’ variable is incremented. When the room temperature state is ‘Hot-Off’ and the DEC is in the state ‘Hot-Off, AC-Off’ the DEC executes an internal transition that takes the state from ‘Hot-Off’, ‘AC-Off’ → ‘Hot-Off’, ‘AC-Switch-On’ and the output function generates the output ‘ON’. The actuator model receives this as an input and changes its state to ‘On’ and outputs ‘ON’ to the room temperature and DEC. The room temperature model executes an external transition function and changes its state to ‘Hot-On’ while the DEC transitions to ‘Hot-On, AC-On’.

We observe the room temperature begins to fall as the ‘temp’ variable is decremented. The DEC states stay in sync with the room temperature model and when the DEC reaches the state ‘Cool-On, AC-On’ it outputs an ‘Off’ to actuator model causing it to move to the ‘Off’ state, and the cycle repeats. The DEC states remain in sync with the room temperature and AC actuator models for the duration of this period.

The setpoint now changes such that the new thresholds are, starting in sequence from ‘Cold’, ‘20’, ‘40’, ‘60’ and ‘80’ respectively. The room temperature model was in the

state ‘Warm-On’ when this change happened and now its state becomes ‘Cool-On’. The DEC changes its state to ‘Cool-On, AC-On’ and outputs ‘Off’ to the actuator. The room temperature is observed to rise again. The setpoints are changed again and set to ‘40’, ‘60’, ‘80’ and ‘100’. The room temperature model was in the state ‘Warm-Off’ and this change caused it to change its state to ‘Cool-Off’. Since the DEC does not take any action in the state ‘Cool-Off, AC-Off’, the AC actuator remains in the ‘Off’ state.

The setpoint is now changed back to ‘20’, ‘40’, ‘60’ and ‘80’. The room temperature model is again in the state ‘Warm-Off’, and this change causes it to move to ‘Hot-Off’. The DEC executes an internal transition that causes the DEC state to move from ‘Hot-Off, AC-Off’ → ‘Hot-Off, AC-Switch-On’. The AC is turned on and the room temperature falls. The setpoints are changed two more times and are set to ‘0’, ‘20’, ‘40’ and ’60, and ‘-20’, ‘0’, ‘20’ and ‘40’. Both of these changes delay the AC from turning ‘Off’ as the room temperature state does not reach the ‘Cool-On’ state. After the final setpoint change, the temperature continues to drop until it reaches ‘0’, which is the threshold between the ‘Warm’ and ‘Cool’ ranges. The DEC is now in the state ‘Cool-on, AC-On’ and therefore instructs the AC actuator to move to the ‘Off’ state.

The DEC is observed to work as required by the control specifications. The DEC attempts to keep the temperature between the ‘Cool’ and ‘Hot’ ranges and instructs the AC actuator to turn ‘On’ or ‘Off’. When the setpoints are changed the room temperature model updates its states and outputs the new range to the DEC. The DEC changes its states and if needed generates outputs to the AC actuator model. We can therefore conclude that the model and the DEC work as expected.

5.3.3.2 The Hybrid Layer Model Implementation

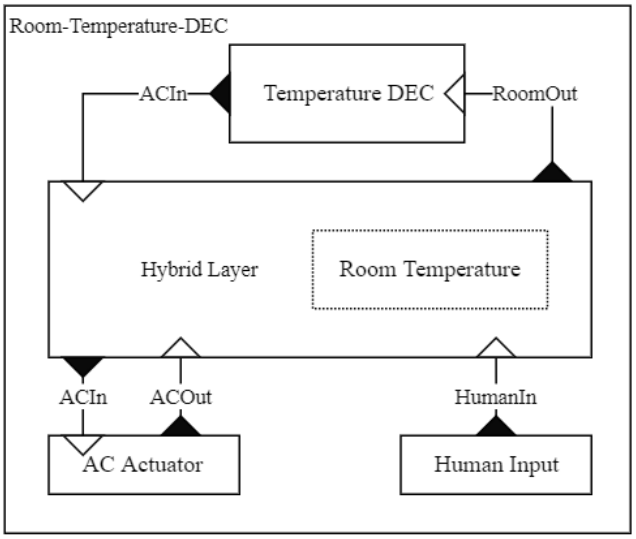

Figure 55. Coupling of the DEC, Hybrid layer, and the AC actuator and human input models.

Figure 55 shows the model coupling with the DEC. The hybrid layer has a dotted room temperature model. This is the abstract model the DEC sees and is shown in Figure 52. The AC actuator model remains unchanged, the only difference is that the model now interacts with the hybrid layer as opposed to directly with the DEC. The human input model simply outputs the integer values '0' to '3' based on the value of an internal variable.

The hybrid layer, in this case, is defined as an atomic model and it implements the algorithm for the unidirectional case as discussed in Section 4.6.3. In addition, the hybrid layer accepts as input, integer values from the human input model and based on this value chooses the setpoints of the model. Finally, the model relays the commands from the DEC to the actuator model. The formal definition of the hybrid layer is defined in Listing 8.

$HL = <X, Y, S, \delta_{INT}, \delta_{EXT}, \lambda, ta>$

```
X = {'ACIn', 'ACOut', 'HumanIn'}
Y = {'RoomOut', 'ACOut'}
```

```
S = {state ∈ {ColdOff, ColdOn, CoolOff, CoolOn, HotOn, HotOff, VeryHotOn,
VeryHotOff}, increment ∈ {true, false}, range ∈ N, stable ∈ {true, false}}

δINT (s)     {

    // (1) Changing the 'temp' variable depending on the state of increment.
    if (increment) temp++
    else temp--
    clamp(temp, -20, 140) // prevents temp from going below -20 or above 140
    // (2) Determining the setpoints based on the range value.
    if (prev != range)
        {
        switch (range)
            {
            case 1: set_range(0, 40)
            case 2; set_range(20, 60)
            case 3; set_range(40, 80)
            case 4; set_range(80, 100)
            }
        stable = false;
        }
    // (3) set the state to the current temperature.
    if (stable)
        {
        state = current_temp()
        }
    else
        {
        // (4) catering for change of setpoint
        cur_state = state
        dest_state = current_temp(state)
        // (5) if desired state is ahead of current state
        if ( direction(dest_state, cur_state))
            {
            cur_state = next_state()
            }
        // (6)is desired state and current state equal?
        if (dest_state == cur_state)
            {
            stable = true
            state = dest_state
            }
        }
    }
δEXT (s, e, x)    {

        // (7) input from the DEC for the AC actuator
        if (x.port == ACIn)
        {
            setState(x.value)
        }
        // (8) Input from the AC actuator.
        if (x.port == ACOut)
        {
            if (x.value == ON)
                increment = true
            else
                increment = false
        }
        // (9) Input from the Human Input
        if (x.port == HumanIn)
            {
                range = x.value
```

```
            }
    }
```

Listing 8. Formal Definition of the hybrid layer.

'state' stores the state of the room temperature as defined in the abstract model. 'increment' state decides whether the temperature is to be incremented or decremented and in hardware simulation used to decide the directionality of the current and desired state. 'range' decides what setpoints are used by the model and 'stable' defines whether the setpoint has changed.

The internal transition function first, (1) increments or decrements the 'temp' variable, depending upon the value of the state 'increment'. Following this (2) it checks if the 'range' state has changed its value. If yes, then the function sets the new setpoints according to the value of 'range' and 'stable' changes its state to 'false'. For example, for 'range == 1' the 'set_range(0,40)' sets the thresholds at '0' and '40' for 'Cold' and 'Warm, as before the remaining threshold values are set at +20 from the previous threshold.

(3) If the state of 'stable' is true then the transition function simply updates the 'state' based on the value of 'temp'. Otherwise, (4) the function stores the current state in 'cur_state' and calculates the desired state and stores it in 'dest_state'. (5) if the desired state is ahead of the current state, then 'cur_state' is updated to the next state. (6) if the desired state and current state are equal then 'stable' is set to 'true' again. If the desired state is behind the current state then (4) the desired state is updated with the new value of 'temp' in the next execution of the internal transition, until the desired state matches the current state.

The external transition function (7) accepts the input from the 'ACIn' port and if it is 'ON' moves 'state' from '-Off' states to the '-On' states. The opposite is true for an

‘OFF’ input. (8) The input from the ‘ACOut’ port updates the ‘increment’ state. This is done so that the hybrid layer knows the state of the AC actuator and can then decide the directionality of the desired and current state. (9) The input from ‘HumanIn’ simply updates the value of ‘range’ according to the input.

Now that we have defined the hybrid layer and the coupled model, we can simulate them. The models are started in the states ‘Cool-Off’, ‘AC_Off’ and ‘Cool-Off’ for the hybrid layer model, the AC actuator model, and the DEC. The ‘increment’ is set to ‘true’, and the room temperature rises. As was the case in the previous Section 5.3.3.1 we observe the DEC state remains in sync with the hybrid layer. When the DEC reaches the state ‘Hot-Off’ it outputs ‘ON’ to the hybrid layer and moves to the ‘Hot-On’ state and stays in this state until the hybrid layer indicates the temperature has decreased. The hybrid layer also moves to the ‘Hot-On’ state and outputs the ‘On’ command to the AC actuator model. After the AC actuator model moves to the ‘On’ state, it outputs ‘On’ through the ‘ACOut’ port. The hybrid layer receives this as an input and changes the ‘increment’ state to ‘false’. The hybrid layer now begins to decrement the value of ‘temp’. The hybrid layer along with the DEC moves toward the ‘Cool-On’ state at which point the DEC outputs an ‘Off’ command and the AC actuator moves to the ‘Off’ state.

The human input model waits for 250-time advances of its time advance and then outputs ‘2’ through the ‘HumanIn’ port. The hybrid layer changes its setpoint and sets ‘stable’ to false. At this point the hybrid layer is in the state ‘Hot-On’ state, the change in setpoint sets the desired state as ‘Warm-On’. Since the desired state is ahead of the current state, we observe the hybrid layer changes its state to ‘Warm-On’. ‘stable’ is set to ‘true’

indicating the current state is equal to the desired state. ‘W’ is output from the hybrid layer to the DEC and in response, the DEC changes its state to ‘Warm-On’.

The human input model after 350 time advance outputs ‘3’ to the hybrid layer. The hybrid layer once again changes the setpoints and sets ‘stable’ to ‘false’. At this point the hybrid layer and DEC have reached the state ‘Cool-Off’ and the AC actuator is in the ‘Off’ state. The new setpoint moves the desired state to ‘Cold-Off’. We observe the hybrid layer stays at the current state and outputs ‘C’ to the DEC. Therefore, the DEC stays in the state ‘Cool-Off’. This is because the desired state is behind the current state. The ‘temp’ variable continues to increment as ‘increment’ is set to true and after each time advance the desired state is calculated again. After about 15-time advances, we observe ‘stable’ state changes to ‘true’. The desired state has reached the current state ‘Cool-Off’. After some time, the hybrid layer outputs ‘W’ to the DEC, and the DEC and hybrid layer finally move to the state ‘Warm-Off’.

We observe similar behavior when the human input model generates the outputs ‘3’ and ‘4’. In both of those cases, the desired state was ahead of the current state. From this simulation, we can see the hybrid layer can recognize when the desired state is ahead or behind the current state. It can then either advance the current state or stay at the current state. It successfully relayed outputs of the DEC to the actuator and presented an abstract model to the DEC. The obtained DEC was able to keep the temperature between ‘Cool’ and ‘Hot’ states and issued correcting outputs when the states moved to ‘Cold’ or ‘VeryHot’. In Section 5.4 we implement the hybrid layer model on an embedded platform to test for proper working.

5.3.3.3 State-Space Complexity of the Model Implementations

Table 2. The state-space complexity of the room temperature and abstract models.

Model	GSS States	GSS Transitions	Reduced States	Reduced Transitions	DEC states	DEC Transitions
Room Temperature Model	40	220	25	136	25	77
Room Temperature Abstract Model	10	19	9	18	9	11

Table 2 shows the state-space complexity of the room temperature model and the abstract model as it was passed through the steps of the approximate method. The initial GSS obtained for the room temperature model had 40 states and 220 transitions. In comparison, the abstract model's state space was only 10 states and 19 transitions. The reduced states and transitions represent the complexity after the application of the control objectives. Despite an almost 50% reduction in complexity for the room temperature model, the abstract model is still comparatively smaller. The final DEC obtained after removing unused states and transitions still yields a larger DEC for the room temperature model as compared to the abstract model.

From this and the implementation result in the previous section we can conclude the hybrid layer can help in greatly reducing the state-space complexity of a system with human interaction and that it can be used in conjunction with a DEC to achieve the control objectives successfully.

5.4 RT-DEVS Execution on an Embedded Platform

We implemented all our models discussed in this chapter on two Nucleo F207ZG boards using the RT-DEVS formalism and RT-Cadmium tool. The first board implements the

room temperature model that consists of the room CO2 concentration model, room occupancy model, and the room fire alarm model. We substitute sensors, with potentiometers in the room conditioning model to simulate the action of the sensors. This allows us the freedom to imitate the action of the sensors in a more controlled manner and is much quicker and more cost-effective than using real sensors. The potentiometers, however, can be replaced by real sensors and actuators and the DEC and models will work as expected, provided the sensors are calibrated properly. This is because in this implementation the input from the potentiometer is used directly as values from a sensor and no software simulated 'temp' or 'conc' values are used. The potentiometers can therefore be replaced by the sensors and the models will work as expected. Similarly, the LEDs are turned on and off by the actuator model themselves and therefore can be replaced by real actuators, and the signal from the models can turn these actuators on or off.

The second board implements the room and water temperature model and consists of the room water temperature model and the hybrid layer room temperature model. For the room temperature model, the temperature of the room and water is simulated in software. The models and DEC respond to the simulated values of the temperature sensors. As before, real sensors can replace the simulated sensor and the models, and the DEC will work as expected.

5.4.1 Executing the Room Conditioning Model on Embedded Hardware

Figure 56 shows the experimental setup for the room conditioning model. The CO2 and Smoke detectors use the potentiometer to generate continuous values between 0 and 1 that are then converted to discrete values between 0-100. The three blue LEDs are turned on by

the actuators when they are in the ‘On’ state. The white LED indicates the occupancy sensor is in the ‘On’ state due to pressing the button.

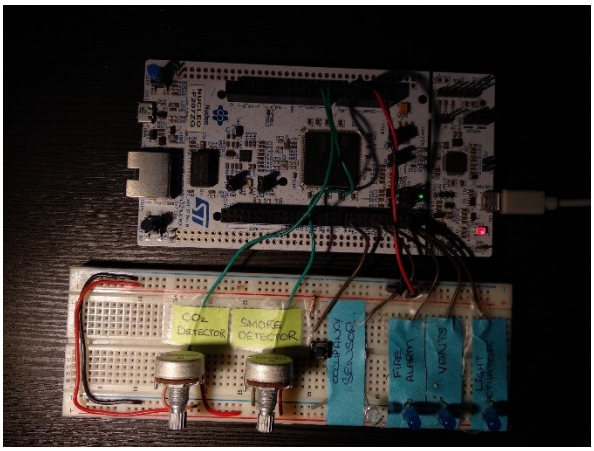

Figure 56.Room Conditioning Model Setup.

The models are coupled as shown in Figure 45 and each of the models is started with the actuators in the ‘Off’ states and the models in the ‘Low-Off’, ‘No-Smoke-Off’, ‘Not-Occupied-Off’ for the room CO2 concentration, fire alarm, occupancy model respectively. The DECs are started in the corresponding states that are in sync with the states of the models.

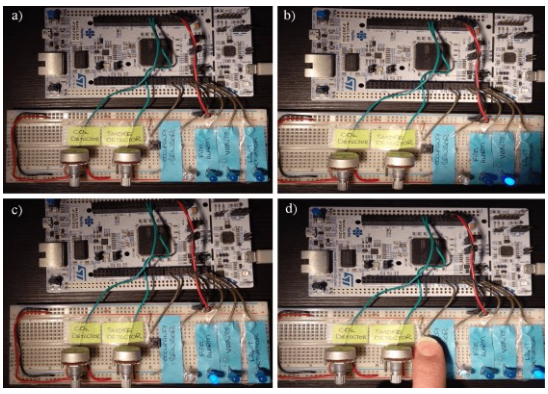

Figure 57. Individual model tests. a) Initial states; b) Increasing CO2 concentration until vents open; c) Increasing smoke concentration until fire alarm is triggered; d) Triggering the occupancy sensor.

Figure 57 shows the stages of executing the model on embedded hardware. In Figure 57a) all the models are in the initial states. We leave the models in this state for some time to see if the states change on their own. No change was observed. We then in Figure 57b) twist the CO2 detector potentiometer until the vents LED lights up. We leave the hardware in this state for a while to observe if any change will occur. No change occurs.

In Figure 57c) we move the CO2 potentiometer back to the starting point and the vents actuator turns off. We then twist the smoke detector potentiometer until the fire alarm actuator is triggered and the LED lights up. We twist the potentiometer back and forth and observe the LED turns off when concentration is lowered and lights up again when the concentration threshold is crossed. Finally, in Figure 57d) we bring the smoke detector potentiometer back to the starting point and press the occupancy sensor switch. We observe the light actuator LED almost instantaneously lights up. When the button is unpressed the LED turns off.

From these observations, we note the state does not change unless the inputs from the sensors change, potentiometers in this case. For example, the room CO2 concentration model DEC remained in the 'High-On' state until the concentration was lowered by twisting the potentiometer back to the starting position. The DECs react as expected to the inputs from the models and output appropriate control outputs to the actuators. The DEC does not transition away from the state of the models if it is left in a state for some time and we can conclude the obtained DECs fulfill the control objectives and do not veer from the control path.

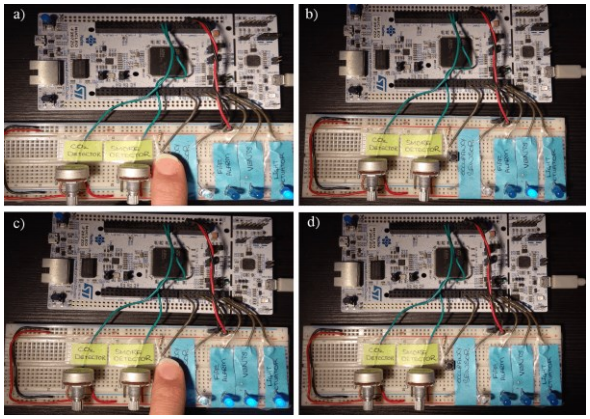

Figure 58. Testing models simultaneously. a) Triggering the occupancy sensor again, b) Increasing CO2 concentration until vents open; c) Increasing smoke concentration and triggering the occupancy sensor; d) returning to the starting states.

In Figure 58 we further test the models by having all of them react to inputs at once. In Figure 58a) we trigger the occupancy sensor repeatedly and observe the light actuator model continues to turn the LED on and off. In Figure 58b) we turn the CO2 concentration potentiometer until the vents actuator model is moved to the 'On' state by the DEC and turn the LED on. In Figure 58c) we move the smoke detector potentiometer until the actuator is moved to the 'On' state and trigger the occupancy sensor button. We leave the hardware in this state for a while and observe no change in the models. Finally, in Figure 58d) we move everything back to the starting state and repeat the scenario in Figure 57.

From this test, we conclude that modular DECs can work in parallel without affecting the operation of the other DECs. The DECs can work in parallel and then work as expected on their own. We can conclude the modular DECs obtained from the approximate method can work in parallel on embedded hardware using the RT-DEVS formalism.

5.4.2 Executing the Room & Water Temperature Model on Embedded Hardware

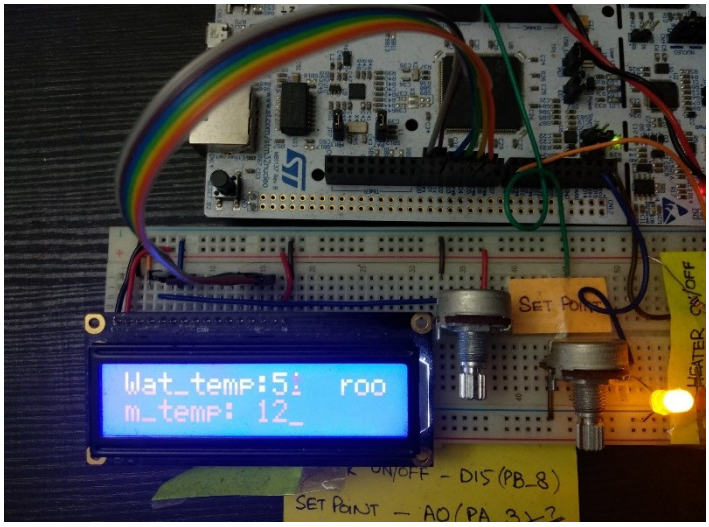

Figure 59. Room and water temperature model hardware setup.

Figure 59 shows the hardware setup for the room and water temperature model. The potentiometer labeled 'Setpoint' controls the setpoint of the room temperature model. The amber LED is turned on by the boiler actuator model when it is in the 'On' state. The red LED to the right of the amber LED is turned on by the AC actuator model when it is in the 'On' state. The LCD shows the current temperature value with the first line showing the water temperature and the second line showing the room temperature. The models are started in the states defined in the software simulation in Section 5.2.3.4. The models are coupled similarly to the room conditioning model. With each model operating independently of the other. The hybrid room temperature model uses the hybrid layer and is coupled as shown in Figure 55.

Figure 60 shows the different stages in the execution of the room and water temperature model. Figure 60a) shows the initial states, the water temperature is set at 30 and is falling as the boiler is off and the room temperature is at 0 and rising.

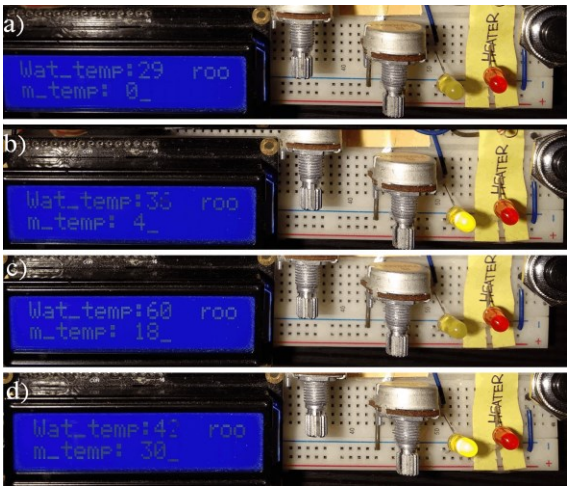

Figure 60. Testing the room water temperature model. a) Initial states; b) The Boiler is on, and the water temperature rises; c) The boiler has turned off and water temperature is falling; d) The boiler is turned on again and the water temperature rises.

Figure 60c) The upper water threshold 60 is crossed and the DEC responds by turning the boiler actuator 'Off', this causes the LED to turn off and the water temperature begins to fall. Figure 60 d) The water temperature once again crosses the lower threshold value of 20 and the boiler actuator turns on again. Throughout this time it can be seen the room temperature continues to rise (from 0 to 30) however it has not crossed any threshold yet and so the AC actuator is not turned on by the DEC.

Figure 61 shows the continuation of the model execution. This time we focus on the room temperature model, in Figure 61a) we observe the model crossing the threshold for the 'Hot' temperature at 40, causing the DEC to output an 'ON' event to the AC actuator model. The AC actuator model moves to the 'On' state and turns the red LED on. The room temperature begins to fall and is currently at 36. In Figure 61b) the room temperature model has crossed the 'Cool' threshold when the temperature reaches 20, and so the DEC signals

the AC actuator to move to the ‘Off’ state. The AC actuator moves to the ‘Off’ state and turns the LED off. We observe the temperature begins to rise.

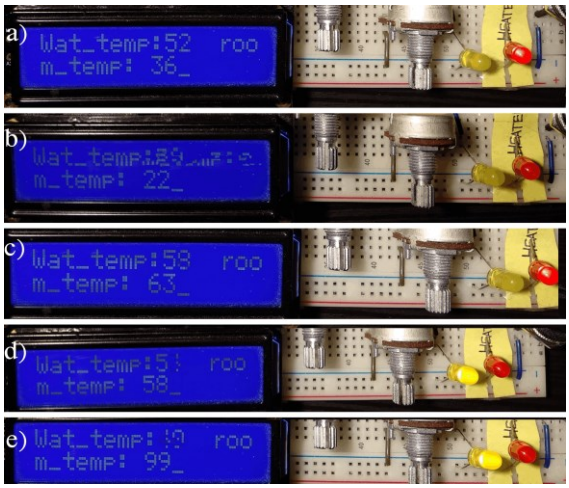

Figure 61. Executing the room and water temperature model, testing the room temperature model. a) AC actuator turns on and the temperature falls; b) AC actuator turns off and the temperature rises again; c) Setpoint is changed, and the AC turns on at new setpoint; d) Setpoint is changed again, and room temperature rises; e) The room temperature keeps rising to the new setpoint.

We change the setpoint of the system by twisting the setpoint potentiometer so that the new setpoints are at 40 and 80. In Figure 61c) we observed the room temperature rises to 80 and then the AC turns on, the temperature then begins to decrease. While the temperature was decreasing, we change the setpoints again so that the thresholds are at 60 and 100. The AC turns off in Figure 61d) as the temperature is below the threshold. The temperature begins to rise again and reaches 99 in Figure 61e). The AC does not turn on yet as the threshold value is at 100.

Figure 62 continues the execution of the model. In Figure 62a) the threshold was crossed, and the room temperature model is in the ‘Hot’ state. Therefore, the DEC instructs the AC actuator to move to the ‘On’ state. The temperature begins to fall again.

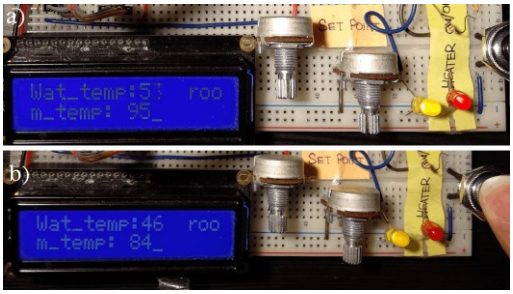

Figure 62. Executing the room and water temperature model. a) AC actuator turns on at the new setpoint and the temperature falls; b) occupancy sensor is triggered, and the water boiler turns on.

Finally, in Figure 62b) we press the button for the occupancy sensor when the water temperature was falling, and the water temperature model was in the 'Warm' state. The DEC moves to 'Warm-Off, Off, Occupancy-On', and from this state, it generates an output 'ON' to the boiler actuator. The water boiler actuator model in response moves to the 'ON' state and turns the LED on and we observe the water temperature rise until it reaches the threshold for 'Hot' state and turns off again.

From this scenario, we tested the ability of the room water temperature model and DEC to keep the water temperature between the 'Cool' and 'Hot' states. The DEC was able to keep the room water temperature model in between these states. In addition, we tested the effect of an occupancy sensor on the DEC 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, in this case, the water temperature, and those sensors can still affect actuators provided the actuator is modeled to include the effect of the sensor. Finally, the states of the DEC and the models stayed in sync and the DEC did not move away from the control path. Therefore,

we can reasonably conclude the DEC obtained fulfills the control objectives in both software simulation and execution on an embedded platform.

For the room temperature model, we have successfully demonstrated that the DEC obtained from the abstract model can meet the objectives and that the hybrid layer is a useful tool to incorporate human interaction in a model. This result held for the software simulation as well as model execution on the embedded platform. The hybrid layer can adequately cater to the changing setpoints of the model, reducing the complexity of the DEC.

After testing in software and executing on a hardware platform, we can be reasonably certain the models and the DECs obtained in this thesis can be used with real sensors and actuators and be expected to perform as designed. The sensors and actuators only need to be interfaced with the models and the DECs and calibrated to the use case. The approximate method then is a useful method to obtain practical working DEC for a given DEVS model.

Chapter 6: Conclusion & Future Work

The use of supervisory control for robotics has shown the potential of using formal techniques to model a system and then formally obtain a working Discrete Event Controller (DEC) for the model. This allows the potential use of M&S techniques to simulate and test the models and the DECs in the development cycle of an application. However, supervisory control theory has issues to scale for larger applications due to the problem of state-space explosion. Reducing the state-space complexity of the models would therefore allow for greater adoption of supervisory control theory. We used the DEVS formalism and devised an approximate method, based on previous works, which replaced the use of 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. Specifically, we utilized the coupling information of the model to decide with certainty if a transition can exist during model execution. In this method, time is approximated, a transition only specifies if it updates the time advance to a finite value or an infinite value (passivation of the model), without specifying the actual value of time. We did not consider zero-time advance in this thesis. The approximate method will always produce a Global State Space (GSS) that has either an equal number of transitions as a GSS from the cartesian product or less. It can never perform worse than the cartesian product in terms of GSS complexity.

In addition, the use of the hybrid layer was investigated to incorporate human interaction with a model such that the human operator changes the setpoints of the model. We demonstrated how human interaction with a model increases the transitions of the

model and therefore leads to increased GSS complexity. We then proposed a general approach to extract a smaller abstract model from a model with human interaction, provided some conditions are met. The DEC can then be obtained from this abstract model. Furthermore, the hybrid layer can implement an algorithm that can over time present the change of setpoint from a human operator to the DEC, while hiding the complexity of human interaction.

We initially used the RT-DEVS extension to DEVS to implement a water tank, pump, and DEC model to test for the correctness of our method. The models were executed on an embedded platform, and we found our method generated a DEC that met the control objectives. The GSS generated by the approximate method had lesser transitions than the cartesian product. We then presented a case study that modeled a smart building with multiple sensors and actuators to further demonstrate the applicability of the method and to test the hybrid layer approach. The models were passed through the approximate method and the cartesian product to compare the obtained GSS. The approximate method produced a reduction of 6% to 17% in terms of transitions in our case study. The obtained DECs and the models were then simulated in software using CADMIUM and then ported to and executed on an embedded platform. The results from these tests showed the obtained DECs fulfilled the control objectives. Similarly, for the hybrid layer, the tests demonstrated the DEC obtained for a room temperature model with human interaction using the hybrid layer had lesser complexity than without using the hybrid layer. From this case study, we showed that the approximate method produces working DECs with reduced complexity and using RT-DEVS the obtained DECs can be implemented and executed on embedded hardware.

We showed the methodology for obtaining a DEC from a coupled DEVS model represented as a directed weighted graph and in the process, we found limitations of the methodology and workarounds to the limitations. 'missing' transitions occur when a DEC is passivated while the coupled model is still active. Manually analyzing the coupled model and the DEC to identify the offending transitions and correcting them in the DEC resolves this issue. This issue occurs because the method approximates time and does not perform a timing analysis of the coupled model. Therefore, it does not recognize the coupled model is active. The approximate method does not perform reachability analysis on the states and transitions of the GSS. This leads to states and transitions being added to the GSS that cannot occur during the execution of the model, unnecessarily increasing GSS complexity. These states and their transitions can be removed with no effect on the DEC.

Our work has shown the use of the approximate method to obtain DECs with a reduced GSS complexity in terms of states. This reduction can extend the applicability of the method to larger models. In addition, we showed the hybrid layer can reduce the complexity of some applications that have human interaction with the models.

This work can be extended in future studies to incorporate the use of timing analysis to resolve the issue of missing transitions and with reachability analysis identifying impossible states and transitions, greatly 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.

An interesting avenue for research would be to explore the possibility of modifying the methodology such that it produces a GSS based on only the input and outputs of the

top coupled model. The ICs are analyzed, to determine only their effect on the outputs of the coupled model. The transitions and states involved in the ICs themselves are not included in the GSS. This would greatly reduce the number of transitions and states in the GSS. The justification for this is that DEC only operates on the inputs and outputs of the coupled model and therefore the effect of the IC can be studied only in terms of its effect on the output of the model.

References

[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of discrete event processes," in *Feedback Control of Linear and Nonlinear Systems*, D. Hinrichsen and A. Isidori, Eds., Springer-Verlag, 1982, p. 202–214.

[2] M. R. Driels, Linear control systems engineering, New York: McGraw-Hill, 1996.

[3] C. G. Cassandras and S. Lafortune, Introduction to Discrete Event Systems, II ed., New, York: Springer US, 2008.

[4] F. L. Lewis, Applied optimal control & estimation : digital design & implementation, Englewood Cliffs, N.J: Prentice Hall, 1992.

[5] M. S. Fadali and A. Visioli, "Chapter 1 - Introduction to Digital Control," in *Digital Control Engineering (Second Edition)*, Second Edition ed., M. S. Fadali and A. Visioli, Eds., Boston, Academic Press, 2013, p. 1–8.

[6] W. M. Wonham, K. Cai and K. Rudie, "Supervisory control of discrete-event systems: A brief history," *Annual Reviews in Control,* vol. 45, p. 250–256, 2018.

[7] K. Tahera, D. C. Wynn, C. Earl and C. M. Eckert, "Testing in the incremental design and development of complex products," *Research in Engineering Design,* vol. 30, p. 291–316, 2019.

[8] F. F. H. Reijnen, T. R. Erens, J. M. van de Mortel-Fronczak and J. E. Rooda, "Supervisory control synthesis for safety PLCs," *IFAC,* vol. 53, p. 151–158, 2020.

[9] A. Gill, Introduction to the theory of finite-state machines., New York: McGraw-Hill, 1962.

[10] K. (. Voss, H. Genrich, G. Rozenberg and C. A. Petri, Concurrency and nets : Advances in Petri nets, Berlin: Springer-Verlag, 1987.

[11] B. P. Zeigler, "Hierarchical, Modular Discrete-Event Modelling in an Object-Oriented Environment," *SIMULATION,* vol. 49, p. 219–230, 1987.

[12] P. Gohari and W. M. Wonham, "On the complexity of supervisory control design in the RW framework," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics),* vol. 30, p. 643–652, 2000.

[13] W. M. Wonham and P. J. Ramadge, "Modular supervisory control of discrete-event systems," *Mathematics of control, signals and systems,* vol. 1, p. 13–30, 1988.

[14] G. Wainer and R. Castro, "DEMES: a Discrete-Event Methodology for Modeling and Simulation of Embedded Systems," *SCS Modeling & Simulation Magazine,* vol. 2, p. 65–73, January 2011.

[15] B. P. Zeigler, H. S. Song, T. G. Kim and H. Praehofer, "DEVS framework for modelling, simulation, analysis, and design of hybrid systems," in *Hybrid Systems II*, Springer Berlin Heidelberg, 1995, p. 529–551.

[16] A. Gollu and P. Varaiya, "Hybrid dynamical systems," in *Proceedings of the 28th IEEE Conference on Decision and Control,*, 1989.

[17] J. S. Hong, H.-S. Song, T. G. Kim and K. H. Park, "A Real-Time Discrete Event System Specification Formalism for Seamless Real-Time Software Development," *Discrete Event Dynamic Systems,* vol. 7, p. 355–375, 1997.

[18] M. Teixeira, J. E. R. Cury and M. H. de Queiroz, "Local modular Supervisory Control of DES with distinguishers," in *ETFA2011*, 2011.

[19] R. Malik and M. Teixeira, "Modular supervisor synthesis for extended finite-state machines subject to controllability," in *2016 13th International Workshop on Discrete Event Systems (WODES)*, 2016.

[20] H. Zhong and W. M. Wonham, "On the consistency of hierarchical supervision in discrete-event systems," *IEEE Transactions on Automatic Control,* vol. 35, p. 1125–1134, 1990.

[21] B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control,* vol. 39, p. 329–342, February 1994.

[22] F. Lin and W. M. Wonham, "Supervisory control of timed discrete-event systems under partial observation," *IEEE Transactions on Automatic Control,* vol. 40, p. 558–562, 1995.

[23] S. Takai and T. Ushio, "A New Class of Supervisors for Timed Discrete Event Systems Under Partial Observation," *Discrete Event Dynamic Systems,* vol. 16, p. 257–278, April 2006.

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

[25] M. V. S. Alves, L. K. Carvalho and J. C. Basilio, "Supervisory control of timed networked discrete event systems," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017.

[26] X. D. Koutsoukos, K. X. He, M. D. Lemmon and P. J. Antsaklis, *Discrete Event Dynamic Systems,* vol. 8, p. 137–173, 1998.

[27] L. E. Holloway and B. H. Krogh, "Synthesis of feedback control logic for a class of controlled Petri nets," *IEEE Transactions on Automatic Control,* vol. 35, p. 514–523, 1990.

[28] K. Yamalidou, J. Moody, M. Lemmon and P. Antsaklis, "Feedback control of petri nets based on place invariants," *Automatica,* vol. 32, p. 15–28, January 1996.

[29] A. Giua and M. Silva, "Petri nets and Automatic Control: A historical perspective," *Annual Reviews in Control,* vol. 45, p. 223–239, 2018.

[30] M. Silva, "Individuals, populations and fluid approximations: A Petri net based perspective," *Nonlinear Analysis: Hybrid Systems,* vol. 22, p. 72–97, November 2016.

[31] C. Vázquez, A. Ramírez-Treviño and M. Silva, "Controllability of timed continuous Petri nets with uncontrollable transitions," *International Journal of Control,* vol. 87, p. 537–552, December 2013.

[32] M. Silva, J. Júlvez, C. Mahulea and C. R. Vázquez, "On fluidization of discrete event models: observation and control of continuous Petri nets," *Discrete Event Dynamic Systems,* vol. 21, p. 427–497, September 2011.

[33] C. Mahulea, C. Seatzu, M. P. Cabasino and M. Silva, "Fault Diagnosis of Discrete-Event Systems Using Continuous Petri Nets," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans,* vol. 42, p. 970–984, 2012.

[34] M. A. Drighiciu, "Hybrid Petri nets: A framework for hybrid systems modeling," in *2017 International Conference on Electromechanical and Power Systems (SIELMEN)*, 2017.

[35] C. R. Vázquez and M. Silva, "Stochastic Hybrid Approximations of Markovian Petri Nets," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 45, p. 1231–1244, 2015.

[36] A. Nazeem and S. Reveliotis, "Designing Compact and Maximally Permissive Deadlock Avoidance Policies for Complex Resource Allocation Systems Through Classification Theory: The Nonlinear Case," *IEEE Transactions on Automatic Control,* vol. 57, p. 1670–1684, 2012.

[37] M. Fabian and A. Hellgren, "PLC-based implementation of supervisory control for discrete event systems," in *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)*, 1998.

[38] F. Basile, P. Chiacchio and D. Gerbasio, "On the Implementation of Industrial Automation Systems Based on PLC," *IEEE Transactions on Automation Science and Engineering,* vol. 10, p. 990–1003, 2013.

[39] M. H. de Queiroz and J. E. R. Cury, "Synthesis and implementation of local modular supervisory control for a manufacturing cell," in *Sixth International Workshop on Discrete Event Systems, 2002. Proceedings.*, 2002.

[40] A. F. Vaz and W. M. Wonham, "On supervisor reduction in discrete-event systems," *International Journal of Control,* vol. 44, p. 475–491, 1986.

[41] J. Zaytoon and B. Riera, "Synthesis and implementation of logic controllers - A review," *Annual Reviews in Control,* vol. 43, p. 152–168, 2017.

[42] M. Cantarelli and J.-M. Roussel, "Reactive control system design using the Supervisory Control Theory: Evaluation of possibilities and limits," in *2008 9th International Workshop on Discrete Event Systems*, 2008.

[43] B. Bonafilia, P. Carlsson, S. Nilsson and M. Fabian, "Robust Manual Control of a Manufacturing System using Supervisory Control Theory," *IFAC Proceedings,* vol. 47, p. 748–753, 2014.

[44] P. C. Y. Chen, J. I. Guzman, T. C. Ng, A. N. Poo and C. W. Chan, "Supervisory control of an unmanned land vehicle," in *Proceedings of the IEEE Internatinal Symposium on Intelligent Control*, 2002.

[45] C. R. C. Torrico, A. B. Leal and A. T. Y. Watanabe, "Modeling and Supervisory Control of Mobile Robots: A Case of a Sumo Robot," *IFAC,* vol. 49, p. 240–245, 2016.

[46] M. S. Khan, "Fuzzy time control modeling of discrete event systems," *ICIAR-51, WCECS,* p. 683–688, 2008.

[47] M. Brambilla, E. Ferrante, M. Birattari and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence,* vol. 7, p. 1–41, 2013.

[48] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd and R. Groß, "Supervisory control theory applied to swarm robotics," *Swarm Intelligence,* vol. 10, p. 65–97, 2016.

[49] Y. Tatsumoto, M. Shiraishi, K. Cai and Z. Lin, "Application of online supervisory control of discrete-event systems to multi-robot warehouse automation," *Control Engineering Practice,* vol. 81, p. 97–104, 2018.

[50] L. Feng and W. M. Wonham, "TCT: A Computation Tool for Supervisory Control Synthesis," in *2006 8th International Workshop on Discrete Event Systems*, 2006.

[51] S. T. J. Forschelen, J. M. van de Mortel-Fronczak, R. Su and J. E. Rooda, "Application of supervisory control theory to theme park vehicles," *Discrete Event Dynamic Systems,* vol. 22, p. 511–540, 2012.

[52] R. Su, J. H. van Schuppen and J. E. Rooda, "Synthesize nonblocking distributed supervisors with coordinators," in *2009 17th Mediterranean Conference on Control and Automation*, 2009.

[53] R. Su, J. H. van Schuppen and J. E. Rooda, "Aggregative Synthesis of Distributed Supervisors Based on Automaton Abstraction," *IEEE Transactions on Automatic Control,* vol. 55, p. 1627–1640, 2010.

[54] P. Aigner and B. McCarragher, "Human integration into control systems: Discrete event theory and experiments," in *Proceedings of the 2nd World Automation Congress*, 1996.

[55] D. Q. Mayne, "Switching control of constrained linear systems," 1996.

[56] E. Mosca and T. Agnoloni, "Closed-loop performance inference for controller selection in switching supervisory control," in *1999 European Control Conference (ECC)*, 1999.

[57] V. Phoha, A. Nadgar, A. Ray and S. Phoha, "Supervisory Control of Software Systems," A. Ray, V. Phoha and S. Phoha, Eds., Springer, 2005, p. 207–238.

[58] R. Ehlers, S. Lafortune, S. Tripakis and M. Y. Vardi, "Supervisory control and reactive synthesis: a comparative introduction," *Discrete Event Dynamic Systems,* vol. 27, p. 209–260, March 2016.

[59] A.-K. Schmuck, T. Moor and K. W. Schmidt, "A Reactive Synthesis Approach to Supervisory Control of Terminating Processes," *IFAC,* vol. 53, p. 2149–2156, 2020.

[60] R. Majumdar and A.-K. Schmuck, "Supervisory Controller Synthesis for Non-terminating Processes is an Obliging Game," *IEEE Transactions on Automatic Control,* p. 1, 2022.

[61] S. Lafortune, "Discrete Event Systems: Modeling, Observation, and Control," *Annual Review of Control, Robotics, and Autonomous Systems,* vol. 2, p. 141–159, May 2019.

[62] L. Prenzel and J. Provost, "PLC Implementation of Symbolic, Modular Supervisory Controllers," *IFAC-PapersOnLine,* vol. 51, p. 304–309, 2018.

[63] Z. Ramezani, J. Krook, Z. Fei, M. Fabian and K. Akesson, "Comparative Case Studies of Reactive Synthesis and Supervisory Control," in *2019 18th European Control Conference (ECC)*, 2019.

[64] Y. Selvaraj, Z. Fei and M. Fabian, "Supervisory Control Theory in System Safety Analysis," in *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops*, vol. 12235, Springer International Publishing, 2020, p. 9–22.

[65] R. Malik and P. N. Pena, "Optimal Task Scheduling in a Flexible Manufacturing System using Model Checking," *IFAC-PapersOnLine,* vol. 51, p. 230–235, 2018.

[66] B. P. Zeigler, "Embedding DEV & DESS in DEVS : Characteristic Behaviors of Hybrid Models," in *DEVS Integrative Modeling & Simulation Symposium*, 2006.

[67] G. Wainer, Discrete-Event Modeling and Simulation, CRC Press, 2017.

[68] H. S. Song and T.-G. Kim, "DEVS Diagram revised: A Structured Approach for DEVS Modeling," in *2010 European Simulation and Modelling Conference*, 2010.

[69] D. Niyonkuru and G. A. Wainer, "Discrete-Event Modeling and Simulation for Embedded Systems," *Computing in Science & Engineering,* vol. 17, p. 52–63, September 2015.

[70] L. Belloli, D. Vicino, C. Ruiz-Martin and G. Wainer, "Building Devs Models with the Cadmium Tool," in *2019 Winter Simulation Conference (WSC)*, 2019.

[71] C. R. Martin, R. Cárdenas, B. St-Aubin and B. Earle, *Cadmium Simulation Environment,* 2019.

[72] B. Earle, K. Bjornson, C. Ruiz-Martin and G. Wainer, "Development of A Real-Time Devs Kernel: RT-Cadmium," in *2020 Spring Simulation Conference (SpringSim)*, 2020.

[73] Arm, "An Introduction to Arm Mbed OS 5," Arm, Cambridge, United Kingdom, 2022.

[74] L. Torvalds, "The Linux Kernel documentation," Linux Foundation, San Francisco, USA, 2022.

[75] STMicroelectronics, *NUCLEO-F207ZG,* Geneva,Switzerland: STMicroelectronnics International N.V., 2022.

[76] R. Ł. Grossman, A. Nerode, A. Ravn and H. Rischel, "Introduction," in *Hybrid Systems*, Springer Berlin Heidelberg, 1993, p. 1–3.

[77] M. Jamshidi, S. Sheikh-Bahaei, J. Kitzinger, P. Sridhar, S. Beatty, S. Xia, Y. Wang, T. Song, U. Dole, J. Liu, E. Tunstel, M. Akbarzadeh, P. Lino, A. El-Osery, M. Fathi, X. Hu and B. P. Zeigler, "V-Lab®– A Distributed Intelligent Discrete-Event Environment for Autonomous Agents Simulation," *Intelligent Automation & Soft Computing,* vol. 9, p. 181–214, January 2003.

[78] S. Sheikh-Bahaei, J. Liu, M. Jamshidi and P. Lino, "An Intelligent Discrete Event Approach to Modeling, Simulation and Control of Autonomous Agents," *Intelligent Automation & Soft Computing,* vol. 10, p. 337–348, January 2004.

[79] G. Maione and D. Naso, "Modeling evolutionary supervisors for multi-agent manufacturing control with discrete event formalism," in *SMCia/01. Proceedings of the 2001 IEEE Mountain Workshop on Soft Computing in Industrial Applications (Cat. No.01EX504)*, 2001.

[80] G. Maione and D. Naso, "Modelling adaptive multi-agent manufacturing control with discrete event system formalism," *International Journal of Systems Science,* vol. 35, p. 591–614, August 2004.

[81] X. D. Koutsoukos, P. J. Antsaklis, J. A. Stiver and M. D. Lemmon, "Supervisory control of hybrid systems," *Proceedings of the IEEE,* vol. 88, p. 1026–1049, July 2000.

[82] E. P. Marcosig, J. I. Giribet and R. Castro, "Hybrid adaptive control for UAV data collection: A simulation-based design to trade-off resources between stability and communication," in *2017 Winter Simulation Conference (WSC)*, 2017.

[83] S. A. Zudaire, M. Garrett and S. Uchite, "Iterator-Based Temporal Logic Task Planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[84] P. Tabuada, G. J. Pappas and P. Lima, "Compositional Abstractions of Hybrid Control Systems," *Discrete Event Dynamic Systems,* vol. 14, p. 203–238, April 2004.

[85] B. C. Rawlings and B. E. Ydstie, "Falsification of combined invariance and reachability specifications in hybrid control systems," *Discrete Event Dynamic Systems,* vol. 27, p. 463–479, April 2017.

[86] M. H. Hwang and B. P. Zeigler, "Reachability Graph of Finite and Deterministic DEVS Networks," vol. 6, p. 468–478, 2009.

[87] D. L. Dill, "Timing assumptions and verification of finite-state concurrent systems," in *Automatic Verification Methods for Finite State Systems*, Springer Berlin Heidelberg, 1990, p. 197–212.

[88] H. S. Song and T. G. Kim, "Application of Real-Time DEVS to Analysis of Safety-Critical Embedded Control Systems: Railroad Crossing Control Example," *SIMULATION,* vol. 81, p. 119–136, February 2005.

[89] H. Praehofer and D. Pree, "Visual modeling of DEVS-based multiformalism systems based on higraphs," in *Proceedings of the 25th conference on Winter simulation - WSC \textquotesingle93*, 1993.

[90] A. Hagberg, P. Swart and D. S Chult, "Exploring network structure, dynamics, and function using networkx," January 2008.

[91] T. H. Cormen, Introduction to algorithms, III ed., Cambridge, Mass: MIT Press, 2009.

[92] T. A. Myatt, J. Staudenmayer, K. Adams, M. Walters, S. N. Rudnick and D. K. Milton, "A study of indoor carbon dioxide levels and sick leave among office workers," *Environmental Health,* vol. 1, October 2002.

[93] G. Van Rossum and F. L. Drake, Python 3 Reference Manual, Scotts, Valley: CreateSpace, 2009.