DEFINITION OF A TRANSPARENT CONSTRAINT-BASED MODELING AND SIMULATION LAYER FOR THE MANAGEMENT OF COMPLEX SYSTEMS

Kevin Henares
José L. Risco-Martín

Dept. of Computer Architecture and Automation
Complutense University of Madrid
Madrid, Spain
khenares@ucm.es, jlrisco@dacya.ucm.es

Marina Zapater

Embedded Systems Laboratory (ESL)
Swiss Federal Institute of
Technology Lausanne (EPFL)
Lausanne, Switzerland
marina.zapater@epfl.ch

ABSTRACT

Modeling and Simulation (M&S) is one of the most multifaceted topics present today in both industry and academia. However, we are involved in a new M&S paradigm. Systems are becoming more complex and new simulation needs arise and have to be studied. As a consequence, the way in which we perform M&S must be adapted, providing new ideas and tools. In this paper, we propose a rule-based constraints evaluator, which facilitate the validation and verification of complex models in a transparent manner. For this, constraints are defined. The constraints definition process is completely independent of the model development process because (a) the set of constraints is defined once the model has been developed, and (b) constraints are validated at simulation time. The proposed Constraint M&S architecture has been built using the Discrete Event System Specification (DEVS) formalism and has been tested on a validated data center simulation model.

Keywords: model checking, constraint modeling and simulation, discrete events, verification, data centers.

1 INTRODUCTION AND RELATED WORK

The design, development and implementation of current systems continues to be a a challenging effort at the systems engineering level. The problem is much more accentuated today, as the new era of the Internet of Things (IoT) dawns. Modeling and Simulation (M&S) of these complex systems are continuously demanding new formal methods to manage the design, development and implementation of such ultra-large systems with a high level of quality and accuracy while fulfilling a wide range of real-time constraints(Mittal 2014). The way in which we perform M&S must be adapted, providing new ideas and tools to separate the model from the simulation, and the implementation from the analysis. These M&S techniques remain difficult to verify and validate. Performing Verification and Validation (V&V) is an exhaustive exercise for any simulation model. Due to the inherent complexity in current simulation models that comprise multi-faceted data-driven methodologies or co-simulation methodologies, V&V is a challenge of its own (Mittal and Risco-Martín 2017).

Validation and verification (V&V) are used to reach the assurance that models are accurate representations of their corresponding systems. *Validation* is the process of testing a model for validity. To validate, input and output trajectories between the source system (whether real or conceptual) and the model under test must be generated. Validity, whether replicative, predictive, or structural, requires these trajectories to be

equal (Zeigler, Praehofer, and Kim 2000). *Verification* is the attempt to establish that the simulation relation holds between a simulator and a model (i.e. the simulator faithfully implements the model's dynamic behavior). There are two general approaches to verification: formal proof of correctness and extensive testing (Zeigler, Praehofer, and Kim 2000), (Sargent 2011). In this paper we focus on extensive testing. According to Zeigler (Zeigler, Praehofer, and Kim 2000) and Sargent (Sargent 2011), the relationship between the conceptual model and the computerized model is identified as computerized model verification. For Zeigler, the simulator ensures that a strict relation (i.e. simulation relation) exists between the conceptual model and the computerized model (Mittal and Risco-Martín 2017). This is the reason why our V&V approach is finally implemented at the simulation layer.

In this paper, we capture the idea of performing V&V at the simulation level and have implemented a constraint specification architecture inside the simulation layer, transparent to the system engineer who defines the model. Quantifying the interaction of all relevant entities or components in a complex simulation model may be a complicated task. Moreover, the modelling process is a priority task in the system design workflow. As a result of this, it is desirable to analyze its interactions in an orthogonal step to the definition of the model. The implementation of such architecture has been tested on a complex data center simulation model (Penas, Zapater, Risco-Martín, and Ayala 2017). Once the model is defined, the set of constraints are introduced using a text file. Next, the model can be simulated and each constraint is checked at every iteration, showing whether the set of constraints is fulfilled or not. The proposed constraints M&S architecture allows for the first time the definition of the validation layer in an orthogonal process to the modeling phase, and use it on a real data center simulation tool for research purposes.

There exist several modeling specifications that provide support for model checking. Among them, Timed Automata with UPPAAL is a popular toolset for model checking (Bengtsson, Larsen, Larsson, Pettersson, and Yi 1996). Another popular modeling language is Petri Nets (Jensen 2013), capable of describing distributed systems. Both Petri Nets and Timed Automata are widely used to model, verify and validate concurrent real-time systems. The main drawback of these two formalisms is that both of them share their inability to handle complex data types. Timed Automata supports the exchange of basic signals and Petri Nets can only operate using tokens. Our approach, on the contrary, uses the Discrete Event System Specification (DEVS) (Zeigler, Praehofer, and Kim 2000) to define the Constraint M&S architecture. DEVS perfectly separates model structure and model behavior, as well as the model itself from the simulator. This simplifies the definition of our architecture. Furthermore, in DEVS, a model specification is not hardly coupled to the application domain, which validates our architecture under a platform-independent specification context.

A variant of DEVS named Finite-Deterministic DEVS (FD-DEVS) was already introduced to support model checking by previous work (Hwang and Zeigler 2009). However, FD-DEVS has been formulated for deterministic systems. Therefore, non-determinism in transitions and advance functions are not accounted for, whereas many engineered and natural systems are inherently non-deterministic. Another variant of DEVS, called Rational Time-Advance DEVS (RTA-DEVS) was defined to allow only rational values in the time advance function (Saadawi and Wainer 2013). RTA-DEVS has a method to transform RTA-DEVS models to Timed Automata, allowing model checking. There are other approaches based on transformations (Pasqua, Foures, Albert, and Nketsa 2012), where authors introduce a method to transform UML sequence diagrams to FD-DEVS models. However, the limitations of both FD-DEVS and Timed Automata are still present.

There are other approaches tied to the application domain. For example, Di Filippo et al. present a mechanism to model and simulate the metabolism of cell populations based on the specification of a model and a set of constraints(Di Filippo, Damiani, Colombo, Pescini, and Mauri 2016). Gholami and Sarjoughian show a model checking verification method for Network-on-Chip models(Gholami and Sarjoughian 2017). In this case, a constrained version of the atomic DEVS modeling formalism is formulated and applied to the verification of a Network-on-Chip router. Our approach is different since the implementation is performed

in the simulation layer. Furthermore, to verify the model we allow the possibility of adding a set of inequations in the same way that constraints are defined in mathematical programming. They are not only related to numeric variables, but also to complex data types.

The main contribution of this research is the definition and implementation of a method to perform model checking through the specification of a rule-based constraints evaluator. It helps the system engineer to validate and verify the functionality of complex models in a transparent manner. After evaluating the pros and cons, we decided to evaluate the constraints in the simulation layer instead of the experimental frame(Zeigler, Praehofer, and Kim 2000). In this way, the declaration of the constraints become transparent of the user.

The paper is organized as follows. We closely examinate our constraint M&S architecture in Section 2. Section 3 shows the implementation of our approach. In Section 4, a complex model of a data center is presented as the use case to validate our proposal. Finally, in Section 5, we summarize this research and discuss some future work.

2 CONSTRAINT MODELING AND SIMULATION: ARCHITECTURE

In this Section we first provide a brief introduction of the DEVS formalism, upon which our proposal is based. Next, we show the architecture of our approach and introduce a motivational example.

2.1 Introduction to the DEVS formalism

DEVS is a general formalism for discrete event system modeling based on set theory (Zeigler, Praehofer, and Kim 2000). The DEVS formalism provides the framework for information modeling which gives several advantages to analyze and design complex systems: completeness, verifiability, extensibility, and maintainability. Once a system is described in terms of the DEVS theory, it can be easily implemented using an existing computational library. The parallel DEVS (PDEVS) approach was introduced after 15 years as a revision of Classic DEVS. Currently, PDEVS is the prevalent DEVS, implemented in many libraries. In the following, unless it is explicitly noted, the use of DEVS implies PDEVS.

DEVS enables the representation of a system by three sets and five functions: input set (X), output set (Y), state set (S), external transition function (δ_{ext}), internal transition function (δ_{int}), confluent function (δ_{con}), output function (λ), and time advance function (ta).

DEVS models are of two types: atomic and coupled. Atomic models are directly expressed in the DEVS formalism specified above. Atomic DEVS processes input events based on their model's current state and condition, generates output events and transition to the next state. The coupled model is the aggregation/-composition of two or more atomic and coupled models connected by explicit couplings. Given the recursive definition of coupled models, they can be a part of a component in a larger coupled model system giving rise to a hierarchical DEVS model construction.

DEVS conceptually separates models from the simulator, making it possible to simulate the same model using different simulators working in centralized, parallel or distributed execution modes. DEVS models can be encoded in different programming environments and simulated with a simple ad-hoc program written in any language. However, there exist many DEVS M&S engines around the world, like DEVSJAVA, CD++, xDEVS, aDEVS, etc. (Risco-Martín, Mittal, Fabero, Zapater, and Hermida 2017)

2.2 Constraint modeling architecture

Once a DEVS model has been defined, it is good practice to enable mechanisms to validate the implementation and check the correct behavior of all its components. Model verification can be performed defining a set of constraints that must be fulfilled. These constraints are dynamic in nature. They can depend on both the initialization parameters of the simulation and on the current state of the system. These constraints may not only be related to the outputs of a single component of the system, but involve combinations of outputs of different components. These components can even coexist in the same constraint, but coming from different levels of the hierarchical DEVS structure.

In the following, we present the architecture of the proposed constraint M&S system for model checking. The set of constraints that the DEVS model must satisfy is defined through simple JSON text files and is completely decoupled from the model and the simulation engine syntax. These constraints are specified following a mathematical approach, in terms of arithmetic and logical equations. It is worthwhile to mention that complex data structures can be used in these equations, as long as the operators are defined for such data structures.

As stated above, DEVS models imply a hierarchical structure of components. There are two types of components: coupled and atomic. Coupled components group other components inside them and are used to reach this hierarchical design. Atomic components have an event based operation and define the actual behaviour of the system. Both of them manage their input/output operation with ports. These ports are linked through couplings that define the relation between components.

```
{
   "vars": {
      <variable_name1>: <arithmetic_expr1>,
      <variable_name2>: <arithmetic_expr2>,
      ...
      <variable_nameN>: <arithmetic_exprN>
   },
   "constraints": {
      "constraint_name1": {"expr": <logic_expr1>, "level": <"info"/"error">},
      "constraint_name2": {"expr": <logic_expr2>, "level": <"info"/"error">},
      ...
      "constraint_nameN": {"expr": <logic_exprN>, "level": <"info"/"error">}
   }
}
```

Figure 1: Formal definition of the set of constraints.

As mentioned before, constraints are specified using JSON files. Following the proposed architecture, two main sections can be defined, namely *vars* and *constraints*. Each section is represented with two main elements in the JSON document. The *vars* section is optional and includes a collection of variables. Each variable can represent a DEVS port or an arithmetic combination of DEVS ports. The *constraints* section is mandatory and specifies the set of constraints that will be checked in simulation time. These constraints are expressed as logical or arithmetic expressions. These expressions can include DEVS ports or previously defined variables. As stated above, these variables (or port values) can involve complex data types, as long as their corresponding operators have been overloaded. Arrays of variables are also allowed. Operations over arrays require the same-length for both arrays, since the inequality is computed element-by-element.

Figure 1 shows the expected structure of a JSON constraints file. Each variable in the *vars* section is expressed as a pair of <*variable_name*>: <*arithmetic_expression*>. The variable name must consist of uppercase and lowercase letters, numbers and underscores. Output ports must be used as operators of the arithmetic expressions. These ports are identified following the full DEVS path in the following format: `coupled`$_1$`.coupled`$_2$`.....coupled`$_N$`.atomic`$_a$`.port`$_p$, where `coupled`$_i$, `atomic`$_a$ and `port`$_p$, are coupled, atomic and port identifiers specified in the definition of the DEVS model structure. If the port used in the expression generate arrays instead of single values, it is necessary to indicate the slice of the array that will be used. This is done specifying the start and the end indexes, as follows: `coupled1.coupled2.atomic1.port1[<start_index>:<end_index>]`.

In the *contraints* section, each constraint has the following specification format: *'<constraint_name>': 'expr': '<logic_expr>', 'level': '<info/error>'*, being:

- *constraint_name*: identifier of the constraint. It has the same restrictions as variable names. This identifier will be shown in the output messages when the expression is not satisfied.
- *logic_expr*: this expression indicates the activation condition of the constraint. It can contain arithmetic ('+', '-', '*' and '/') and logical ('==', '!=', '<', '<=', '>', '>=', '&&' and 'll') operators. As operands, the expression can contain the full path of a port, a number or boolean literal, or a defined variable in the *vars* section. Some auxiliary functions can be used to deal with arrays, such as *sum*, *len*, *min*, and *max*.
- level: this item specifies the severity level of the constraint. If is set to *info*, only warning messages will be produced when the constraint is checked. If is set to *error*, the constraint will be treated as critical and the simulation will be stopped when the related constraint is not satisfied.

Constraints are evaluated when all the involved ports have produced an output. Hence, if at least one of the ports used as operator is empty, the constraint is not evaluated.

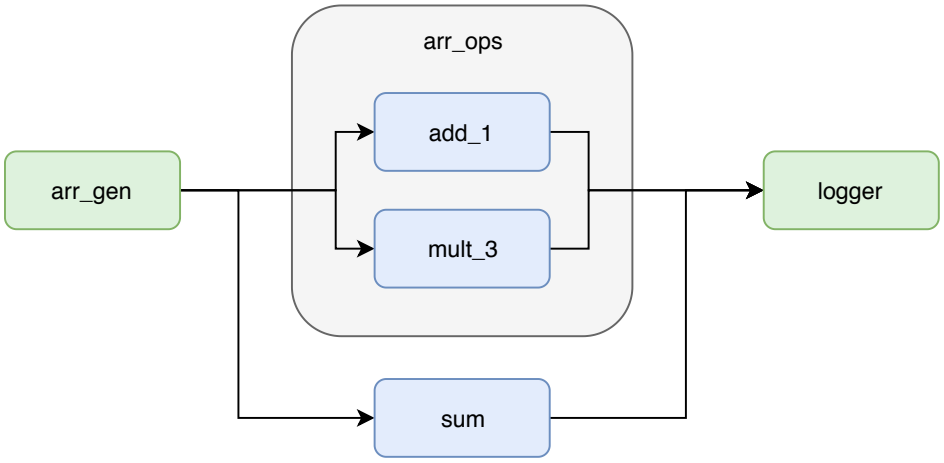

Figure 2: Example of a DEVS system.

2.3 Motivational example

Figure 2 depicts a complete example of a JSON constraints file. This set of constraints is applied to the DEVS example given in Figure 4. The example comprises a generator (`arr_gen`), that generates arrays of size 5 with random integer numbers. Some basic operations are applied on those arrays. Inside the `arr_-`

```
{
    "vars": {
        "arr_adder": "arr_ops.add_1.out[0:5]",
        "arr_mult": "arr_ops.mult_3.out[0:5]",
          "gen_sum": "sum.out",
        "adder_mult": "arr_adder + arr_mult",
        "mult_sum_0_3": "sum(arr_ops.mult_3.out[2:5])"
    },
    "constraints": {
        "adder_eq_mult": {"expr": "arr_adder == arr_mult", "level": "info"},
        "ms_lt_gs": {"expr": "mult_sum_0_3 < gen_sum", "level": "info"},
        "check_mult": {"expr": "arr_mult > {59,61,3,4,5}", "level": "info"},
        "check_mult_sum": {"expr": "mult_sum_0_3 >= 100", "level": "error"}
    }
}
```

Figure 3: Example of JSON constraints file.

`ops` coupled module, `add_1` and `mult_3` atomic modules add and multiply all the elements of the input arrays by constants, respectively. The `sum` module adds up all the elements of the input arrays, returning an integer as a result. All the outputs of these last three modules are sent to a `logger` module, that shows the results.

The first two variables of the constraints file (Figure 3) simply get the values of the two atomic modules inside the `arr_ops` module. The third one (`gen_sum`) computes the sum of the original arrays generated by the `arr_gen` module. The fourth one (`adder_mult`) sums the arrays contained in the first two variables. The last one, (`mult_sum_0_3`), returns a scalar with the sum of the three last elements of the `mult` module output. In the constraints section, four constraints are defined. They specify some basic constraints that must be applied over the system through logical expressions, using the previous defined variables and some literals. The first three constraints are only informative, so when they are not fulfilled only warning messages will be displayed. The last one is a critical constraint, and the simulation will be terminated when it is checked.

3 CONSTRAINT MODELING AND SIMULATION: IMPLEMENTATION

The proposed architecture has been implemented in the C++ branch of the xDEVS simulation engine [1]. Figure 4 depicts a scheme of the final implementation. The set of constraints is initially given as a text file. All the results of the model checking are stored in an output text file as well. When the simulation starts, the DEVS engine performs the following steps (see Figure 4): (i) execution of all the output (lambda) functions, (ii) propagation of the events produced and execution of the corresponding external, internal (or both) transition functions, (iii) evaluation of constraints (new step), and (iv) all the events stored at the input/output ports are removed. Finally, the simulation engine goes to step (i) and starts again, until the maximum number of iterations or the maximum simulation time limit are reached.

As can be seen, the constraints are repeatedly evaluated at step (iii), when all the events have been propagated from the output ports to the corresponding input ports. It is worthwhile to mention that, depending on the simulation engine, events are copied and propagated, existing in both output and associated input ports, or just propagated, existing in the destination ports, only. In the case of xDEVS events are copied and

[1] https://github.com/jlrisco/xdevs

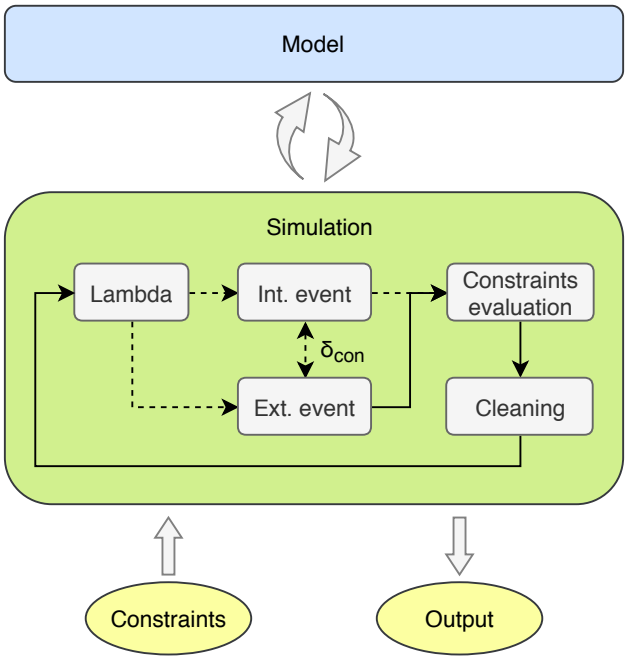

Figure 4: General view of the proposed constraint M&S system.

propagated. This must be taken into account when defining the set of constraints. As a general rule, we assume the most restrictive policy, i.e., events are propagated but not copied.

Summarizing, right after the lambda and transition functions have been evaluated, and before the DEVS simulation engine cleans the ports information, the set of constraints is evaluated. At this point, the outputs of the ports implied in the specified constraints are examined. Based on these values and the constraints definition, some mathematical expressions are evaluated. When a constraint is not satisfied, warning messages are displayed or the simulation is finished (depending on the severity level of the constraint). As can be seen, since the constraints are checked at the simulation layer, the execution is transparent to the system engineer and independent from the model definition. This aspect is very useful when validating models, since the validation process can be tackled once the model has been completely defined. Under a model checking approach implemented at the modeling layer, the validation process must evolve with the definition of the model, which from our point of view, is not practical.

Algorithm 1 shows the complete implementation of the model/constraints checking implementation in form of pseudocode. A maximum time of simulation is given as argument (`max_time`). The simulation time is initialized at the beginning of the procedure and updated at the end of each iteration. In this way, the simulation will continue the execution until the simulation time exceeds the specified maximum time. The actual xDEVS implementation allows us to specify a maximum number of DEVS iterations as well.

As stated in Section 2, there are three types of couplings in a DEVS model: (i) Internal Couplings (IC) connecting components that share the same first parent, (ii) External Input Couplings (EIC) connecting the input ports of coupled modules to one or more of their child components, and (iii) External Output Couplings (EOC) connecting ouput ports of components to one or more output ports of their first parents. Following Algorithm 1, the propagation of values in these couplings is separated into two functions. Firstly, IC and EOC couplings are propagated using the `propagateOutput` function following a bottom-up procedure. After this, values in EIC are propagated using the `propagateInput` function in a top-down way. This separation ensures a correct propagation of values, grouping the values generated by the different

Algorithm 1 Constraint DEVS M&S implementation.

```
1:  procedure SIMULATE(max_time)
2:      sim_time ← 0
3:      while sim_time <= max_time do
4:          for comp ∈ components do
5:              if sim_time = comp.next_event() then
6:                  comp.lambda()
7:          propagateOutput()
8:          propagateInput()
9:          for comp ∈ components do
10:             if sim_time = comp.next_event() then
11:                 comp.int_event()
12:             if comp.has_input() then
13:                 comp.ext_event()
14:         evaluate_constraints()
15:         sim_time ← next_event()
```

components in the input ports of the coupled modules to then transmit them to the corresponding child components and allowing an smooth check of constraints.

4 APPLICATION AND DISCUSSION

In order to validate the constraint M&S architecture we use as a case study the SFIDE data center simulator developed as part of our previous work (Penas, Zapater, Risco-Martín, and Ayala 2017). SFIDE is a DEVS-based data center simulation framework that enables researchers to incorporate server and data center models, and assess the impact of both workload allocation and cooling control strategies. Given that the main goal of SFIDE is to allow researchers to abstract their developments from the simulation internals, having ways of checking the correctness of the models and policies implemented is of utmost importance.

In this section we start by briefly introducing the SFIDE simulator, highlighting the needs for verification. Then, we describe the experimental scenario used as a case study, together with the different types of constraints being checked. Finally, we describe how the constraints M&S architecture is able to efficiently spot both modeling errors and issues associated with the policies being tested in the simulator.

4.1 Overview of the SFIDE data center simulator

The skyrocketing energy consumption of data center facilities, which accounted for over 1.3% of the world energy consumption in 2011, and grows at a yearly rate of 20% (Koomey 2011) has resulted in a growing interest in the research community for the development of both workload allocation and cooling control strategies to minimize the power consumption of data centers, while guaranteeing the performance (understood as execution time or throughput) of applications. Because of the dynamism of server workloads, as well as of the large amount of servers in data center facilities, simulating the behavior of these infrastructures is not a trivial task.

The DEVS-based SFIDE (*Simulation Framework and Infrastructure for Data cEnters*) simulator allows researchers in computer architecture and engineering interested to assess the performance, power consumption and thermal behavior of the servers and data centers when running a workload while setting specific cooling parameters. SFIDE allows defining the configuration of a data center, understood as both the server arrange-

ment in the room, the server type and model, its power and performance model, as well as the workload it executes. Moreover, it allows to simulate the cooling both inside the data center room and the overall facility level, allowing to obtain both computing and cooling figures for the overall facility. Furthermore, the real benefit of SFIDE resides on its capability to implement, test and assess arbitrary workload allocation strategies (i.e., algorithms to decide the specific allocation of incoming jobs to servers) and cooling control policies. These two factors dramatically impact the overall energy consumption of the facility and have been widely analyzed in literature.

4.2 Experimental setup and scenario

The scenario tackled in this case study consists on a data center equipped with 20 racks, each hosting 10 servers. The cooling equipment simulated consists of in-row coolers attached to the server racks, and cooled down using a chiller and tower outside the server room. For further details about the cooling infrastructure, the reader is referred to our previous work (Zapater, Turk, Moya, Ayala, and Coskun 2015). To showcase the importance of the constraints M&S architecture, we assume an heterogeneous setup in which half of the servers in each rack are Intel S2600GZ servers, while the other half are SPARC T3-2. These two server models are chosen because they exhibit important differences both in their power consumption and thermal behaviour, achieving different maximum power consumption (P^{max}, due to variations in the power consumption of both the CPU and memory subsystems), and maximum reliable peak temperature thresholds (T^{max}). The servers execute workloads of the SPEC CPU 2006 benchmark suite. As described in (Penas, Zapater, Risco-Martín, and Ayala 2017), the server models of the SFIDE simulator provide realistic power and temperature values, which were obtained and validated using real traces.

Table 1: Summary of the most relevant parameters of the servers being modeled and simulated.

Server type	P^{max}	T^{max}	Core count
Intel S2600GZ	225	85	12
SPARC T3-2	620	90	32

The most relevant characteristics of the servers used in our experiments are highlighted in Table 1. P^{max}_{CPU}, P^{max}_{mem} and T^{max} are the most relevant parameters that need to be checked to ensure the correctness of the simulation. They are highly sensitive to both to programming errors (i.e, bugs) that could arise during the implementation of both workload allocation and cooling control algorithms; and to errors on the models themselves. Because of this, we use them to test the constraints M&S architecture. In particular, we add constraints to the system to check the following characteristics:

1. Per-server maximum temperature threshold violation (T^{max}_{intel}, T^{max}_{sparc}). Surpassing this threshold would lead to server turn-off in a real environment.
2. Per-server maximum power consumption (P^{max}_{intel}, P^{max}_{sparc}). Surpassing this threshold is physically impossible, and highlights a software implementation error.

In our scenario, both constraints are assumed to be critical and, therefore, abort the simulation when met. Figure 5 shows a diagram of the data center scenario being simulated, whereas Figure 6 shows the DEVS model of the data center room (cooling is not shown for the sake of clarity), where we have highlighted in green the variables upon which constraints are being checked, together with the parameters that have the highest impact on these variables (server air inlet temperature and job allocation, in orange and red, respectively).

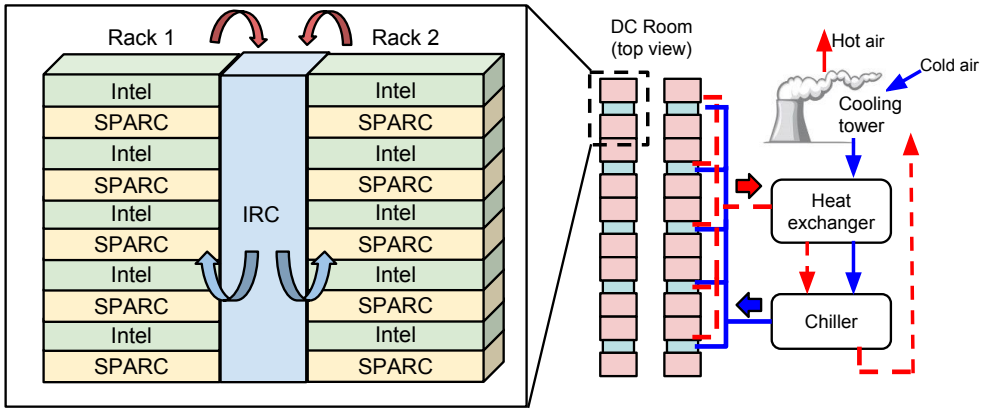

Figure 5: Diagram of the data center scenario.

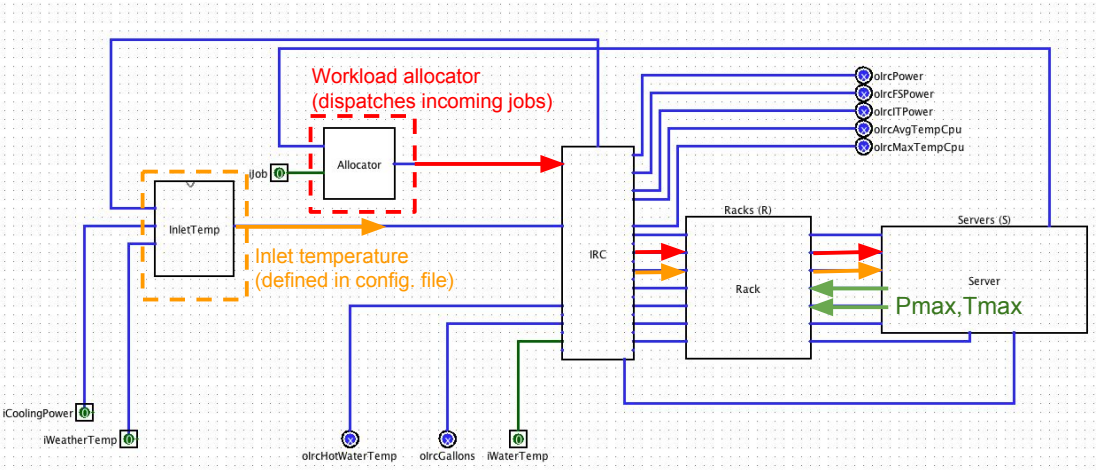

Figure 6: Diagram of DEVS room model of the simulated data center scenario.

4.3 Results and discussion

We run SFIDE with the constraints M&S architecture in the experimental setup described above, and we manually introduce implementation errors to mock the bugs usually introduced during any development process. We introduce two errors: (i) an error in the C++ implementation of the workload allocator module, and (ii) an error in the input configuration file, which sets the cooling temperature to a value that is too high.

Workload allocator policy: We create a new *allocator* policy, in charge of assigning tasks to servers, that disregards the core count of the server and wrongly assigns more than one 12-core job to the Intel servers, doubling the maximum server capacity. Given that the *Server* power model is workload-agnostic, power computation will be incorrect, reaching almost 300W, which is way above the maximum power threshold of 220W for Intel servers. This leads to a wrong temperature computation (and a fake thermal emergency, surpassing T_{intel}^{max}), which is propagated to the calculation of cooling power consumption. As this situation does not result into any C++ error, the only way of checking the correctness of the result is via logging or debugging. However, the constraints M&S architecture detects a violation on the constraint and aborts the simulation.

Inlet temperature configuration: We modify the input configuration file of the simulator to force the inlet temperature of the servers (i.e., the cooling set-point of the data room) at a value which is above the admissible range (i.e., 35^oC). In this case, both the Intel and the SPARC server violate the T^{max} threshold. The constraints M&S architecture allows to easily spot the situation, enabling to backtrace the error. Furthermore, a second constraint can be added directly on the cooling subsystem to prevent such a situation to happen, as it also impacts the functionality of the cooling system.

5 CONCLUSIONS AND FUTURE WORK

In this work we presented a Constraint Modeling and Simulation architecture implemented as an extension of the xDEVS simulation engine. This architecture provides a transparent and straightforward way to validate the behavior of DEVS complex systems. This architecture is conceived as part of the DEVS simulation layer, making the validation process orthogonal to the model design. Also, the proposed architecture allows us to check mathematical properties over different system components, even if they have been defined at different levels of the DEVS hierarchical structure. The set of constraints is defined through JSON text files. The structure and syntax of these files have been detailed. Moreover, the development of the DEVS constraint checker has been provided.

The implementation of the Constraint M&S architecture has been tested on a complex DEVS model, the SFIDE data center model presented in (Penas, Zapater, Risco-Martín, and Ayala 2017). This use case proves the utility of the proposed validation layer.

As future work, we plan to extend our specification, for example including wildcards in the definition of variables and constraints, or the utilization of model classes additionally to model names (to generalize even more the use of each constraint). The use of states as operators is not planned for the moment, as this aspect requires serious restrictions in the implementation of the DEVS simulation engine.

ACKNOWLEDGMENTS

This work has been partially supported by the Education and Research Council of the Community of Madrid (Spain), under research grant P2018/TCS-4423, and by the EC H2020 RECIPE (GA No. 801137) project.

REFERENCES

Bengtsson, J., K. Larsen, F. Larsson, P. Pettersson, and W. Yi. 1996. "UPPAAL — a tool suite for automatic verification of real-time systems". In *Hybrid Systems III*, edited by R. Alur, T. A. Henzinger, and E. D. Sontag, pp. 232–243. Berlin, Heidelberg, Springer Berlin Heidelberg.

Di Filippo, M., C. Damiani, R. Colombo, D. Pescini, and G. Mauri. 2016. "Constraint-based modeling and simulation of cell populations". In *Italian Workshop on Artificial Life and Evolutionary Computation*, pp. 126–137. Springer.

Gholami, S., and H. S. Sarjoughian. 2017. "Modeling and verification of network-on-chip using constrained-DEVS". In *Proceedings of the Symposium on Theory of Modeling & Simulation*, pp. 9. Society for Computer Simulation International.

Hwang, M. H., and B. P. Zeigler. 2009. "Reachability graph of finite and deterministic DEVS networks". *IEEE Transactions on Automation Science and Engineering* vol. 6 (3), pp. 468–478.

Jensen, K. 2013. *Coloured Petri nets: basic concepts, analysis methods and practical use*, Volume 1. Springer Science & Business Media.

Koomey, J. 2011. "Growth in Data center electricity use 2005 to 2010". Technical report, Analytics Press, Oakland, CA.

Mittal, S. 2014. "Model engineering for cyber complex adaptive systems". In *EMSS*.

Mittal, S., and J. L. Risco-Martín. 2017. *Guide to Simulation-Based Disciplines: Advancing Our Computational Future*, Chapter Simulation-based Complex Adaptive Systems, pp. pp. 127–151.

Pasqua, R., D. Foures, V. Albert, and A. Nketsa. 2012. "From sequence diagrams uml 2. x to fd-devs by model transformation". In *European Simulation and Modelling Conference*, pp. pp–37.

Penas, I., M. Zapater, J. L. Risco-Martín, and J. L. Ayala. 2017. "SFIDE: A Simulation Infrastructure for Data Centers". In *Proceedings of the Summer Simulation Multi-Conference*, SummerSim '17, pp. 34:1–34:12. San Diego, CA, USA, Society for Computer Simulation International.

Risco-Martín, J. L., S. Mittal, J. C. Fabero, M. Zapater, and R. Hermida. 2017. "Reconsidering the performance of DEVS modeling and simulation environments using the DEVStone benchmark". *SIMULATION* vol. 93 (6), pp. 459–476.

Saadawi, H., and G. Wainer. 2013. "Principles of discrete event system specification model verification". *Simulation* vol. 89 (1), pp. 41–67.

Sargent, R. G. 2011. "Verification and Validation of Simulation Models". In *Proceedings of the Winter Simulation Conference*, WSC '11, pp. 183–198, Winter Simulation Conference.

Zapater, M., A. Turk, J. M. Moya, J. L. Ayala, and A. K. Coskun. 2015, Dec. "Dynamic workload and cooling management in high-efficiency data centers". In *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*, pp. 1–8.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation. Integrating Discrete Event and Continuous Complex Dynamic Systems*. 2 ed. Academic Press.

AUTHOR BIOGRAPHIES

KEVIN HENARES is a Ph.D. candidate at the Complutense University of Madrid (UCM). His work focuses on the development of robust modeling and simulation methodologies to study the behavior of complex systems. His email address is khenares@ucm.es.

MARINA ZAPATER is a Post-Doctoral researcher in ESL-EPFL since 2016. She received her Ph.D. degree in Electronic Engineering from Universidad Politécnica de Madrid, Spain, in 2015, and a M.Sc. in Telecommunication Engineering and a M.Sc. in Electronic Engineering, both from Universitat Politècnica de Catalunya (UPC), Spain, in 2010. Her research interests include thermal and power optimization of heterogeneous servers, and energy efficiency in data centers. Her email address is marina.zapater@epfl.ch.

JOSÉ L. RISCO-MARTÍN received his Ph.D. from Complutense University of Madrid, and currently is Associate Professor in the Department of Computer Architecture and Automation at Complutense University of Madrid. His research interests include computer aided design, optimization and discrete event simulation. He can be reached at jlrisco@ucm.es.