Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

AUTOMATED MODEL VERIFICATION USING AN EQUIVALENCE TEST ON A REFERENCE MODEL

Akin Akbulut
Stephan Abke

Heinz Nixdorf Institut
University of Paderborn
Fürstenallee 11
33102 Paderborn, GERMANY

Christoph Laroque

Business Computing
University of Applied Sciences Zwickau
Scheffelstrae 39
08066 Zwickau, GERMANY

ABSTRACT

In this article, a cross-tooling method for automated model verification is presented using a reference model. Furthermore, the method is implemented in teaching, using a web platform. The method bases on Yücesan and Schruben (1992), demonstrating a procedure for the examination of a structural and behavioral equivalence of two simulation models based on Simulation Graph models. However, Simulation Graph models are subject to an event-oriented modeling world view. Since current simulation tools use a process-oriented - easier to understand - modeling world view, a simple queuing model shows how transformation from a process-oriented world view (Simio 2017, AnyLogic 2017) takes place in an event-oriented world view (Simulation Graph models). A further step then checks the structural equivalence via an isomorphic mapping on the resulting planar graphs.

1 INTRODUCTION

The University of Paderborn offers an introduction course in Modeling and Simulation for Material Flow Systems. This course is aimed at undergraduate students at the Faculty of Economics and thus also at students, with a major in business studies or business engineering, who have no or little knowledge of programming. The course content includes the theory of modeling and concludes by completion of a simulation study in production and logistics – this also requires the implementation of a computer model. The task of the simulation study includes a fictitious description of a manufacturing system, a problem definition and an objective. The choice of a simulation application, in which the computer model is implemented, will be chosen by the students independently. We regularly observe that students who are located in the business studies or business engineering can easily create an adequate concept model for a given problem, but they are very unsure about the implementation of a simple computer model. As a consequence, the lecturers' hours are frequently visited by participants to discuss whether a computer model is correct or whether a concept model has been correctly transformed into a computer model. The problem space addressed here is shown in Figure 1. Since the simulation study is carried out on a fictitious manufacturing system, the methods for verification & validation described in the literature are not applicable so that students can not compare their model results (for example from a control calculation) with a real system, see e.g. Balci (1998), Knaak et al. (2005), Rabe et al. (2008). A possibility with the existing data to check whether a concept model has been correctly transformed into a computer model can be done by a comparison with a sample solution or a reference model. However, a manual verification is time-consuming because the structure, all functions, and all other attributes of both models must be compared. This fact led us to the question of how an automated approach for model verification must be built up and which prerequisites must apply in order to create a method that can be used for teaching. Our contribution in

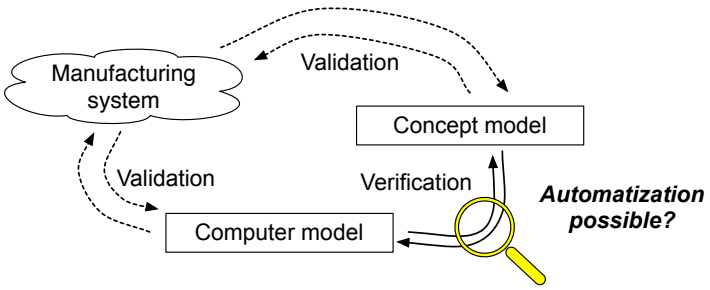

Figure 1: Process of Modeling.

this paper is a method that is capable of comparing two computer models and make a statement whether both models model the same manufacturing system. Since the students use different simulation tools, the method must also be able to compare models from two different simulation tools.

2 RELATED WORK

An explicit check on equality of two simulation models is described in the literature as equivalence of models and is associated with abstraction techniques for simulation models. In this context, it is checked whether an abstracted model calculates still the same or similar results as before its abstraction, see e.g. Sevinc (1991), Zeigler et al. (2000), Johnson et al. (2005). Overstreet (1982) differentiates two different equivalences: An equivalence of the structure (or structural equivalence) and an external equivalence. Accordingly, there is a structural equivalence when the sets A and B, by which two models are described, are identical. External equivalence exists if the inputs U and the calculated outputs Y are identical for two models. Sargent (1988) provides another definition. He considers the values of the state variables calculated by the events $(S_{e_1}, S_{e_2}, ..., S_{e_n})$ and the discrete time steps at which the events occur $(T_{e_1}, T_{e_2}, ..., T_{e_n})$. If the values and the time steps are identical for both models, there is a behavioral equivalence according to Sargent. Another definition in Systems Science gives Klir (1991) for systems in general with

> "A homomorphic relation (or homomorphism) between two systems is contingent upon a function from relevant entities of one system (the original) onto the corresponding entities of the other system (the modeling system) under which the relation among the entities is preserved. If the function, which is called a homomorphic junction, is bijective, the relation is preserved completely (we say that the two systems are isomorphic);"

Zeigler et al. (2000) differentiates between five different Level of morphism. Form level 0 (Observation frame) where the inputs, outputs and time bases of two systems (or models) can be put into correspondence, to level 4 (Coupled component) where components of the systems can be placed into correspondence so that corresponding components are equal. However, Yücesan and Schruben (1992) have demonstrated how behavioral equivalence can be determined using Simulation Graph models. They examine the structure of two Simulation Graph models through an isomorphic mapping between two models. If an isomorphic mapping exists, then the models are structurally equivalent. In a next step, they prove that structural equivalence also implies behavior equivalence. The proof is made by contraposition. This proof puts us in a position to compare two models structurally and to make a decision about whether both models have the same output (behavior equivalence), without executing the models. Apart from these theoretical treatments, there is no other literature demonstrating a practical application. Also Simulation Graph models are subject to an event-oriented modeling world view. However, current simulation tools use a process-oriented modeling world view, and there is no literature demonstrating an algorithm to transform from one world view to another world view. Closing this knowledge gap, we present a method with an application for the verification of simulation models, based on Yücesan and Schruben (1992), in order to use in teaching.

Therefore, we will introduce the modeling technique of the Simulation Graph models next. The method is extended so that models of current simulation applications can be analyzed and, in addition, a comparison of models from different simulation tools becomes possible.

2.1 Simulation Graph Models

Simulation Graph models are a mathematical form of Event Graph models, therefore we will introduce this modeling technique first. In Event Graphs, simulation models are represented using directed graphs, see Figure 2 (Schruben 1983). Events are represented by nodes. Each node has its own state variables and is associated with a set of state changes that occur in its own variables as soon as a node is activated. Event Graphs focus on the individual events in a system to be modeled. Entities, for example a discrete material flow, are only implicitly presented for this reason. Relations between events are represented by directed edges between two nodes. Edges can have logical and temporal expressions. A logical expression (i) determines the conditions that must be met for an edge to be passed. A temporal expression t describes how much time passes when an edge is passed. The illustrated event graph can be interpreted as follows:

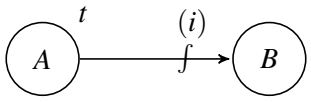

Figure 2: Structure of an Event Graph Model.

"Whenever an event A occurs and the condition (i) is satisfied, the event B is scheduled in t time units". In literature, Event Graphs are used in different variations to create event-discrete simulation models (see e.g. Law and Kelton 2000, Sargent 1988, Schruben 1991). Schruben describes Simulation Graph models as an explicitly mathematical form of Event Graphs and defines them as

$$\mathscr{S} = (\mathscr{F}, \mathscr{C}, \mathscr{T}, \Gamma, G). \tag{1}$$

The first four sets define the entities in a Simulation Graph model, see (1). These are represented by $\mathscr{F}$ the event functions, $\mathscr{C}$ the conditions, $\mathscr{T}$ the time delays, and γ the execution priorities defined as $\mathscr{F} = \{f_v : \ STATES \rightarrow STATES \mid \forall v \in V(G)\}$, the set of event functions associated with vertex v. $\mathscr{C} = \{c_e : \ STATES \rightarrow \{0,1\} \mid \forall e \in E(G)\}$, the set of edge conditions with a mapping from $STATES$ to $0 = false$ if a condition is not satisfied, and $1 = true$ if a condition is heeded. $\mathscr{T} = \{t_e : \ STATES \rightarrow \mathscr{R}^+ \mid \forall e \in E(G)\}$, the set of edge delay times with a mapping from $STATES$ to a nonnegative real number. $\Gamma = \{\gamma_e : \ STATES \rightarrow \mathscr{R}^+ \mid \forall e \in E(G)$, the set of event execution priorities, where $STATES$ is defined as in Zeigler (1984). To conclude, G is a directed graph as ordered triple $(V(G), E(G), \Psi_G)$ and describes the structure of a Simulation Graph model.

3 TRANSFORMATION

The challenge in transformation is that a model is translated from one modeling concept into another modeling concept. Current simulation tools, e.g. AnyLogic (2017) and Simio (2017), describe models with process blocks. A component in such a model is a complete process, e.g. A assembly cell which represents a assembly process. This modeling concept is called a process-oriented world view (Law and Kelton 2000). However, Simulation Graph models are using an event-oriented world view. A model component in such a graph represents a single event, e.g. an "arrival of material on a assembly cell" (Page and Kreutzer 2005). A process usually consists of a large number of individual events, so the process "assembly" in the event-oriented world view can be represented by the events "Arrival of material at assembly cell", "Processing start in assembly cell" and "Processing end in assembly cell". However, this is only one of many ways to describe a process with its inherent events. Information about the actual events aggregated in a process is hidden in the event list of the respective simulation tools (at runtime). Since AnyLogic and

Simio do not have the ability to analyze their event lists, this work has been used to manually determine which events are hidden behind a process in a simulation tool, see Table 1. The table is structured as follows: If a process type has a restriction regarding its resources, e.g. resources must exist in sufficient quality and/ or quantity, it will be modeled with two types of events, see *Work process*. An event type for the arrival, in which a resource is blocked and an event type for a departure, in which a resource is released again. If a process type has a restriction on its capacity to hold entities, it is described with an event type arrival, see *Queue*. The process types *Source* and *Sink* are always described with the event type arrival.

Table 1: Transformation of Elementary Process Types in Event Types.

Process type	**Behavior**	**Event type**
Source	generate entity forward entity	Arrival
Sink	take entity destroy entity	Arrival
Queue	take entity & swap in swap out entity & forward entity	Arrival
Work process/ Assembly	take entity bind resource & schedule delay unbind resource & forward entity	Arrival Departure

Using this table, the new Simulation Graph model is created in three steps using Algorithm 1 in Figure 3. We are briefly explaining its functionality:

1. **Initialization:** From a source model, all process blocks are extracted and assigned to the set K. All links from the source model are assigned to the set L. Then the sets $\mathscr{S}$ for Simulations Graph model, $\mathscr{F}$ for event functions, $\mathscr{C}$ for constrains, $\mathscr{T}$ for time delays, Γ for execution priorities, E for edges, V for vertexes, G for the directed graph, and Z for the state variables are declared and initialized with the empty set, see lines 1 to 3 in Algorithm 1. After we separated the links and process blocks into two different sets K and L, we are now able to process them one by one.
2. **Vertexes & local properties:** For each process block in set $k \in K$ form the source model, the corresponding events are created for each type defined in the *switch-case* statements, see lines 5 to 25. The local properties of the events are then formed and parameterized with the values from each corresponding process block. Local properties are all properties which are in no direct relation to a predecessor process block, e.g. in lines 7 to 11 the event type source contains one vertex v, one state variable z, one looping edge $e \leftarrow (v,v)$, one delay time $t_e \leftarrow k.delay$ associated with the edge e, and one event function $f_v = \{z \leftarrow z + k.arrival\}$ associated with the vertex v.
3. **Edges & global properties:** For each link $l \in L$ from the source model, a directed edge is formed in the Simulation Graph model to correctly transfer the model relations, see lines 25 to 26. After all events ($\forall v \in V$) have been generated with the local properties and the directed edges have been created, in a last step, all properties that are directly dependent on a predecessor process block are set. Therefore we start with the second event vertex $v_2 \in V$, to ensure that there is always one predecessor vertex, see line 28. This relates all event functions, conditions, and time delays which are set in dependence on a preceding process block, e.g. in lines 30 to 34 the event type service has a event function $f_{v_k} = \{z_{k-1,S} \leftarrow z_{k-1,S} - 1\}$ to increment one entity from his predecessor, when it gets activated. Also there are two constraints; First to ensure enough (R > 0) resources $c_{(v_{k-1},v_k)}$ = $\{z_{k,R} > 0\}$ before activating vertex v; And second to ensure enough entities to process (S > 0) in the states variables of the predecessor vertex v, with $c_{(v_{k+1},v_k)} = \{z_{k-1,S} > 0\}$.

Algorithm 1 in Figure 3 is an example implementation for the simulation tool AnyLogic. The algorithm follows the described three steps and can also be adapted for each further process-oriented simulation tool using this approach. The algorithm builds a new Simulation Graph model from a source model. The process

building blocks source, delay, assembler, service, queue, and sink (Process Modeling Library, AnyLogic 2017) are handled.

3.1 Example of a Transformation

To clarify the procedure, a simple queuing model is described that has been implemented with the tools AnyLogic and Simio. Now both implementations will be transformed into separate Simulation Graph models. The queuing model consists of a material arrival (x piece, an interval a), a waiting area (capacity y pieces), and a service station that operates sequentially (resource 1, processing time s). A correct computer model in the simulation tool Anylogic contains a source, a queue, a delay, and a sink process block as shown in Figure 4. The model implemented in AnyLogic given the algorithm in Figure 3 as source model generates the following Simulation Graph model $\mathscr{S}_A$ with

$$G_A(V) = \{A_0, A_1, A_2, A_3, A_4, A_5\}, \tag{2}$$

$$G_A(E) = \{E_{(A_0,A_1)}, E_{(A_1,A_1)}, E_{(A_1,A_2)}, E_{(A_2,A_3)}, E_{(A_3,A_4)}, E_{(A_4,A_3)}, E_{(A_4,A_5)}\}, \tag{3}$$

$$\Psi_A = \{(A_0,A_1),(A_1,A_1),(A_1,A_2),(A_2,A_3),(A_3,A_4),(A_4,A_3),(A_4,A_5)\}, \tag{4}$$

$$\mathscr{F}_A = \{f_{A_0}; f_{A_1}; f_{A_2}; f_{A_3}; f_{A_4}; f_{A_5}\} = \{$$

$$z_1 \leftarrow z_1 + x; z_2 \leftarrow z_2 + x, z_1 \leftarrow z_1 - x; z_3 \leftarrow 0, z_2 \leftarrow z_2 - 1; z_3 \leftarrow 1, z_4 \leftarrow z_4 + 1; z_5 \leftarrow z_5 + 1, z_4 \leftarrow z_4 - 1\}, \tag{5}$$

$$\mathscr{C}_A = \{C_{(A_1,A_2)}; C_{(A_2,A_3)}; C_{(A_4,A_3)}\} = \{z_2 < y; z_3 > 0; z_2 > 0\}, \tag{6}$$

$$\mathscr{T}_A = \{t_{(A_1,A_1)}; t_{A_3,A_4}\} = \{a, s\}, \tag{7}$$

$$\Gamma_A = \emptyset. \tag{8}$$

The sets (2), (3) and (4) are creating the directed graph and thus the structure of the Simulation Graph model. The sets (5), (6), (7) and (8) are defining the behavior of the Simulation Graph model. In the simulation tool Simio, an adequate model contains the process blocks source, server and sink, see figure 5. Due page restrictions, we have only described one algorithm for transforming a process-oriented model into an event-oriented model for reasons of demonstrating the implementation. So, a adapted algorithm for the simulation tool Simio should generate a Simulation Graph model $\mathscr{S}_B$ with the following sets

$$G_B(V) = \{B_0, B_1, B_2, B_3, B_4, B_5\}, \tag{9}$$

$$G_B(E) = \{E_{(B_0,B_1)}, E_{(B_1,B_1)}, E_{(B_1,B_2)}, E_{(B_2,B_3)}, E_{(B_3,B_4)}, E_{(B_4,B_3)}, E_{(B_4,B_5)}\}, \tag{10}$$

$$\Psi_B = \{(B_0,B_1),(B_1,B_1),(B_1,B_2),(B_2,B_3),(B_3,B_4),(B_4,B_3),(B_4,B_5)\}, \tag{11}$$

$$\mathscr{F}_B = \{f_{B_0}; f_{B_1}; f_{B_2}; f_{B_3}; f_{B_4}; f_{B_5}\} = \{$$

$$z_1 \leftarrow z_1 + x; z_2 \leftarrow z_2 + x, z_1 \leftarrow z_1 - x; z_3 \leftarrow 0, z_2 \leftarrow z_2 - 1; z_3 \leftarrow 1, z_4 \leftarrow z_4 + 1; z_5 \leftarrow z_5 + 1, z_4 \leftarrow z_4 - 1\}, \tag{12}$$

$$\mathscr{C}_B = \{C_{(B_1,B_2)}; C_{(B_2,B_3)}; C_{(B_4,B_3)}\} = \{z_2 < y; z_3 > 0; z_2 > 0\}, \tag{13}$$

$$\mathscr{T}_B = \{t_{(B_1,B_1)}; t_{(B_3,B_4)}\} = \{a, s\}, \tag{14}$$

$$\Gamma_B = \emptyset. \tag{15}$$

The sets (9), (10) and (11) are also shaping the directed graph and thus the structure of the Simulation Graph model. The sets (12), (13), (14) and (15) are defining the behavior of the Simulation Graph model.

Data: Source Model
Result: Simulation Graph Model $\mathscr{S}$

$K \leftarrow$ Process Blocks $\subseteq$ Source Model;
$L \leftarrow$ Links $\subseteq$ Source Model;
$\mathscr{S}, \mathscr{F}, \mathscr{C}, \mathscr{T}, \Gamma, E, V, G, Z \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** $|K|$ **do**
  switch $k_i.type$ **do**
    case *Source* **do**
      $V \leftarrow V \cup \{v_i \leftarrow k_i.Typ\}$;
      $Z \leftarrow Z \cup \{z_{i,x\leftarrow S} \leftarrow 0\}$;
      $E \leftarrow E \cup \{e_i \leftarrow (v_i, v_i)\}$;
      $\mathscr{T} \leftarrow \mathscr{T} \cup \{t_{e_i} \leftarrow k_i.delay\}$;
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_i}\} = \{z_i \leftarrow z_i + k_i.arrival\}$;
    case *Service* | *Assembler* | *Delay* **do**
      $V \leftarrow V \cup \{v_{i,x\leftarrow 1} \leftarrow k_i.type\} \cup \{v_{i,x\leftarrow 2} \leftarrow k_i.type\}$;
      $Z \leftarrow Z \cup \{z_{i,x\leftarrow R} \leftarrow 0\} \cup \{z_{i,x\leftarrow S} \leftarrow 0\}$;
      $E \leftarrow E \cup \{e_{i,x\leftarrow 1} \leftarrow (v_{i,1}, v_{i,2})\}$;
      $E \leftarrow E \cup \{e_{i,x\leftarrow 2} \leftarrow (v_{i,2}, v_{i,1})\}$;
      $\mathscr{T} \leftarrow \mathscr{T} \cup \{t_{e_{i,1}} \leftarrow k_i.delay\}$;
      $z_{i,R} \leftarrow k_i.resources$;
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_{i,1}}\} = \{z_{i,R} \leftarrow z_{i,R} - 1\}$;
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_{i,2}}\} = \{z_{i,R} \leftarrow z_{i,R} + 1,\ z_{i,S} \leftarrow z_{i,S} + 1\}$;
    case *Queue* | *Sink* **do**
      $V \leftarrow V \cup \{v_i \leftarrow k_i.type\}$;
      $Z \leftarrow Z \cup \{z_{i,x\leftarrow R} \leftarrow 0\} \cup \{z_{i,x\leftarrow S} \leftarrow 0\}$;
      $z_{i,R} \leftarrow k_i.capacity$;
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_i}\} = \{z_{i,S} \leftarrow z_{i,S} + 1\}$;
for $j \leftarrow 1$ **to** $|L|$ **do**
  $E \leftarrow E \cup \{e_{h\leftarrow|E|+1} \leftarrow (v_x \leftarrow \text{Source}(l_j),\ v_y \leftarrow \text{Target}(l_j))\}$;
for $k \leftarrow 2$ **to** $|V|$ **do**
  switch v_k **do**
    case *Service* | *Assembler* | *Delay* **do**
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_k}\} = \{z_{k-1,S} \leftarrow z_{k-1,S} - 1\}$;
      $\mathscr{C} \leftarrow \mathscr{C} \cup \{c_{(v_{k-1},v_k)}\} = \{z_{k,R} > 0\}$;
      $\mathscr{C} \leftarrow \mathscr{C} \cup \{c_{(v_{k+1},v_k)}\} = \{z_{k-1,S} > 0\}$;
      $k \leftarrow k + 1$
    case *Sink* **do**
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_k}\} = \{z_{k-1,S} \leftarrow z_{k-1,S} - 1\}$;
      $\mathscr{C} \leftarrow \mathscr{C} \cup \{c_{(v_{k-1},v_k)}\} = \{z_{k,S} < z_{k,R}\}$;
$G \leftarrow (V, E)$;
$\mathscr{S} \leftarrow (\mathscr{F}, \mathscr{C}, \mathscr{T}, \Gamma, G)$;

Figure 3: Algorithm 1 - Transformation in a Simulation Graph Model.

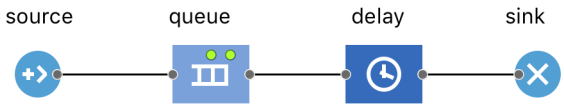

Figure 4: Model A - Simple Queuing Model in AnyLogic.

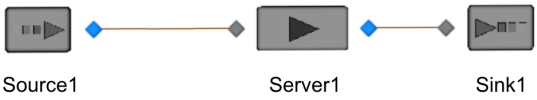

Figure 5: Model B - Simple Queuing Model in Simio.

4 EQUIVALENT SIMULATION MODELS

Two Simulation Graph models $\mathscr{S}_A = (\mathscr{F}_A, \mathscr{C}_A, \mathscr{T}_A, \Gamma_A, G_A)$ and $\mathscr{S}_B = (\mathscr{F}_B, \mathscr{C}_B, \mathscr{T}_B, \Gamma_B, G_B)$ are isomorph to each other $\mathscr{S}_A \approx \mathscr{S}_B$, if there is a bijective mapping between $\mathscr{S}_A$ and $\mathscr{S}_B$ with (Yücesan and Schruben 1992)

$$\Theta : V(G_A) \to V(G_B),$$
$$\Phi : E(G_A) \to E(G_B),$$
$$\Lambda : \mathscr{F}_A \to \mathscr{F}_B,$$
$$\Omega : \mathscr{C}_A \to \mathscr{C}_B,$$
$$\chi : \mathscr{T}_A \to \mathscr{T}_B,$$
$$\Delta : \Gamma_A \to \Gamma_B.$$

The sets Θ, Φ, Λ, Ω, χ, Δ form an isomorphism between the simulation graph models $\mathscr{S}_A$ and $\mathscr{S}_B$ (Bondy and Murty 1976). The first two mappings Θ and Φ check a match in the structure between the vertexes and edges of the graphs. Next, the remaining four sets will be explained in detail. The mapping $\Lambda : \mathscr{F}_A \to \mathscr{F}_B$ checks for a match between the event functions, which are responsible for the calculation of all the system states. For two event functions to be considered equivalent, their calculations must produce identical results. In cases where a function has a stochastic component, the generated random numbers need not be identical, but they must be subject to identical stochastic distributions. The mapping $\Omega : \mathscr{C}_A \to \mathscr{C}_B$ checks for a match between the conditions of two different models. The terms for the expressions and the binary operators must not be identical for the conditions in the different simulation graph models. It is only important that the conditions assume identical Boolean values as soon as an identical model state exists in two different simulation graph models. The mapping $\chi : \mathscr{T}_A \to \mathscr{T}_B$ checks for a match between the delay times. If the delay times are deterministic, then they must be identical in the different models. If the delay times are stochastic, then it must be ensured that they are subject to the same stochastic distributions, so that they can be classified as equivalent. The mapping $\Delta : \Gamma_A \to \Gamma_B$ checks for a match between the execution priorities.

Since the execution priorities are not considered in this work, both sets are empty. In general, there is currently no efficient algorithm known for checking isomorphism between two graphs (Deiser et al. 2016, S.95). However, a Simulation Graph model is a planar graph (Yücesan 1989), i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints. For the examination of an isomorphism in planar graphs, different algorithms were presented in the literature. The implementation presented here is based on our theoretical description, we made first. The graphs G of the Simulation Graph models are first checked to determine whether they are planar and then to check whether an isomorphic mapping exists between them. For this purpose, we use the method as described by Kukluk et al. (2004). If the mapping Θ and Φ can be created, the function *checkIsomorphism* gives *true* as result. Then the sets $\mathscr{F}$, $\mathscr{C}$, $\mathscr{T}$ and Γ are checked for a match. For this purpose, all assignments ($\leftarrow$) and binary operations

$(+,-,<,=,>)$ are checked with regular expressions for string analysis whether they exist in the same quantity in both sets. If this is the case, the *equals* function returns the Boolean value *true* as result. After termination the Algorithm 2 in Figure 6 gives five boolean variables as result. Only if all variables contain the value *true* the test for isomorphism for two Simulation Graph models is complete and successful.

Data: Simulation Graph Model $\mathscr{S}_A$, Simulation Graph Model $\mathscr{S}_B$
Result: Boolean, Boolean, Boolean, Boolean, Boolean
1 iso, fn, constr, times, prio $\leftarrow false$;
2 iso $\leftarrow checkIsomorphsim(G(\mathscr{S}_A), G(\mathscr{S}_B))$;
3 fn $\leftarrow equals(\mathscr{F}(\mathscr{S}_A), \mathscr{F}(\mathscr{S}_B))$;
4 constr $\leftarrow equals(\mathscr{C}(\mathscr{S}_A), \mathscr{C}(\mathscr{S}_B))$;
5 times $\leftarrow equals(\mathscr{T}(\mathscr{S}_A), \mathscr{T}(\mathscr{S}_B))$;
6 prio $\leftarrow equals(\Gamma(\mathscr{S}_A), \Gamma(\mathscr{S}_B))$;
7 **return** iso, fn, constr, times, prio

Figure 6: Algorithm 2 - Isomorphic Simulation Graph Models.

4.1 Example of an Isomorphism Test

After model A and model B have been transformed into separate Simulation Graph models $\mathscr{S}_A$ and $\mathscr{S}_B$, it can be checked whether there is an isomorphism between both models with:

$$\Theta : V(G_A) \rightarrow V(G_B) = \{\Theta(A_0) = B_0, \Theta(A_1) = B_1, \Theta(A_2) = B_2, \Theta(A_3) = B_3, \Theta(A_4) = B_4, \Theta(A_5) = B_5\} \quad (16)$$

$$\Phi : E(G_A) \rightarrow E(G_B) = \{\Phi((A_0,A_1)) = (B_0,B_1), \Phi((A_1,A_1)) = (B_1,B_1), \Phi((A_1,A_2)) = (B_1,B_2),$$
$$\Phi((A_2,A_3)) = (B_2,B_3), \Phi((A_3,A_4)) = (B_3,B_4), \Phi((A_4,A_3)) = (B_4,B_3), \Phi((A_4,A_5)) = (B_4,B_5)\} \quad (17)$$

$$\Lambda : \mathscr{F}_A \rightarrow \mathscr{F}_B = \{\Lambda(f_{A_0}) = f_{B_0}, \Lambda(f_{A_1}) = f_{B_1}, \Lambda(f_{A_2}) = f_{B_2}, \Lambda(f_{A_3}) = f_{B_3}, \Lambda(f_{A_4}) = f_{B_4}, \Lambda(f_{A_5}) = f_{B_5}\} \quad (18)$$

$$\Omega : \mathscr{C}_A \rightarrow \mathscr{C}_B = \{\Omega(C_{(A_1,A_2)}) = C_{(B_1,B_2)}, \Omega(C_{(A_2,A_3)}) = C_{(B_2,B_3)}, \Omega(C_{(A_4,A_3)}) = C_{(B_4,B_3)}\} \quad (19)$$

$$\chi : \mathscr{T}_A \rightarrow \mathscr{T}_B = \{\chi(t_{(A_1,A_1)}) = t_{(B_1,B_1)}, \chi(t_{(A_3,A_4)}) = t_{(B_3,B_4)}\} \quad (20)$$

$$\Delta : \Gamma_A \rightarrow \Gamma_B = \{\Delta(\emptyset) = \emptyset\} \quad (21)$$

Therefore we try with Algorithm 2 in line 2 to assert the equation (16) and (17). Then in line 2 to 6, the algorithm is trying with regular expressions for string analysis to establish the equations (18), (19), (20), and (21). When the algorithm terminates, only if all variables contain the value *true* the test for isomorphism for two Simulation Graph models is successful. After an isomorphic mapping has been identified between $\mathscr{S}_A$ and $\mathscr{S}_B$, we assume that model A and model B are equivalent.

5 WEB PLATFORM

The method was implemented as an application and integrated into a web platform. The structure corresponds to a classic SaaS (Software as a Service) architecture, as shown in Figure 7. Students can access different tasks on the platform via a Web browser. Each of these tasks contain a description of a manufacturing system, a problem position, an objective, and a reference model. The reference model is a sample solution, but the student is not given access to it. The students are asked to implement a computer model for a task and to save it after completion. Students themselves choose the simulation tool with which they implement their model. A computer model stored in this way is uploaded for verification into the web platform and read in with an XML parser. The model is transformed with an algorithm into a Simulation Graph model.

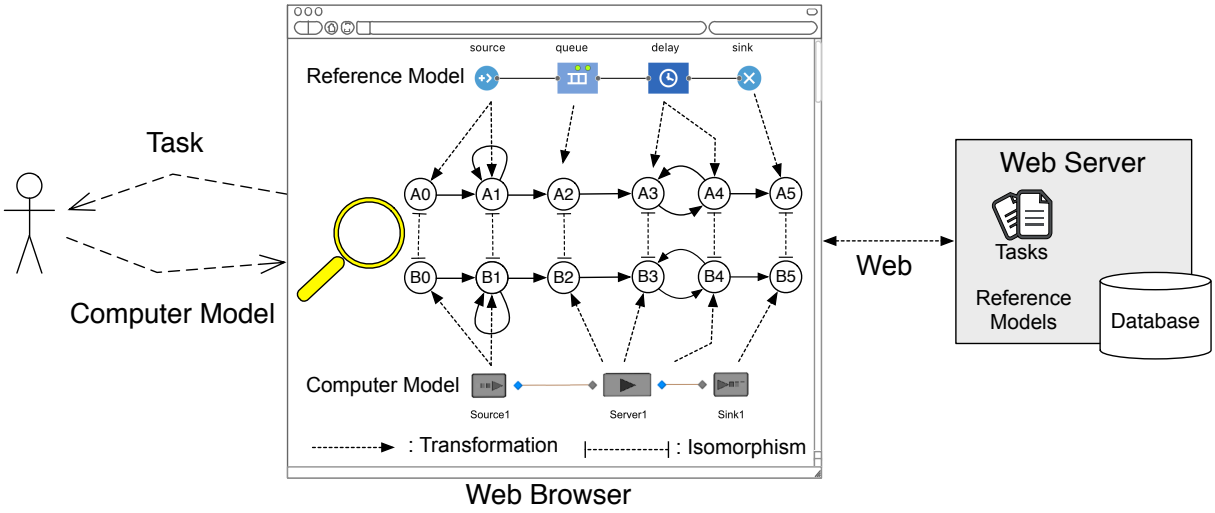

Figure 7: Overview of Implementation.

With Algorithm 2, the model is checked together with a reference model from a database for an existence of a isomorphic mapping between them. The algorithms are executed on client side using the programming language JavaScript. Once an isomorphic mapping is identified ($\mathscr{S}_A \approx \mathscr{S}_B$), model A and model B are assumed to be structurally equivalent. Since it is shown in Yücesan and Schruben (1992) that a structural equivalence also implies a behavioral equivalence, it is assumed that Simulation Graph models identified in this way, calculate the same results Y when they get same inputs Y – therefore a computer model is correctly implemented for a given task.

6 CONCLUSION

In this article, a cross-tool-based method for automated model verification was presented using a reference model. Furthermore, implementation took place in teaching, using a web platform. For this method, we refer to the work of Yücesan and Schruben (1992) in which a procedure for the examination of a structural and behavioral equivalence of two simulation models was demonstrated. It is based on Simulation Graph models. However, Simulation Graph models are subject to an event-oriented modeling world view. Since current simulation tools use a process-oriented modeling world view and process-oriented model blocks are easier to understand for beginners, a simple queuing model shows how a transformation from a process-oriented world view (Simio 2017, AnyLogic 2017) into an event-oriented world view (Simulation Graph) can take place. As a further step, structural equivalence was tested via an isomorphic mapping to the resulting planar graph. During the test, it is assumed that two simulation models (even if implemented in different simulation tools) must have the same structure, functions, conditions, and time delays so that they can be identified as (structure) equivalent. These conditions, which lead to a positive test result, make the method very strict and inflexible. For this reason, there is currently still a need for further research in order to allow identifying models that do not have the same structure (structure equivalence) but calculate the same results (external equivalence).

REFERENCES

AnyLogic 2017. “Simulation Tool”. www.xjtek.com/AnyLogic. [Online; accessed 2017-04-27].

Balci, O. 1998. “Verification, Validation and Testing”. In *Handbook of Simulation - Principles, Methodology, Advances, Applications, and Practice*, edited by J. Banks, 335–393. N.Y., USA: Wiley.

Bondy, J., and U. Murty. 1976. *Graph Theory with Applications*. North-Holland, N.Y., USA: Macmillan.

Deiser, O., C. Lasser, V. Elmar, and W. D.. 2016. *12 x 12 Schlüsselkonzepte zur Mathematik - Diskrete Mathematik*. Berlin, Germany: Springer.

Johnson, R. T., J. W. Fowler, and G. T. Mackulak. 2005. "A Discrete Event Simulation Model Simplification Technique". In *Proceedings of the 2005 Winter Simulation Conference*, edited by M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 2172–2176. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Klir, G. J. 1991. *Facets of Systems Science*. International Federation for Systems Research International Series on Systems Science and Engineering. Boston, MA, USA: Springer US.

Knaak, N., B. Page, and W. Kreutzer. 2005. "Validation, Verification, and Testing of Simulation Models". In *The Java Simulation Handbook - Simulating Discrete Event Systems with UML and Java*, edited by B. Page and B. Kreutzer, 195–231. Aachen, Germany: Shaker.

Kukluk, J. P., L. B. Holder, and D. J. Cook. 2004. "Algorithm and experiments in testing planar graphs for isomorphism.". *Journal of Graph Algorithms and Applications* 8 (2): 313–356.

Law, A., and W. Kelton. 2000, 4. *Simulation Modeling and Analysis*. 3 ed. McGraw-Hill.

Overstreet, C. 1982. *Model Specification and Analysis for Discrete Event Simulation*. Ph. D. thesis, Virginia Tech., Blacksburg, Virginia, USA.

Page, B., and W. Kreutzer. 2005. *The Java Simulation Handbook - Simulating Discrete Event Systems with UML and Java*. Aachen, Germany: Shaker.

Rabe, M., S. Spieckermann, and S. Wenzel. 2008. *Verifikation und Validierung für die Simulation in Produktion und Logistik - Vorgehensmodelle und Techniken*. Berlin, Germany: Springer.

Sargent, R. 1988, October. "Event Graph Modeling for Simulation with an Application to Flexible Manufacturing Systems". *Management Science* 34 (10): 1231–1251.

Schruben, L. 1983, November. "Simulation Modeling with Event Graphs". *Communications of the ACM Journal* 26 (11): 957–963.

Schruben, L. 1991. *Sigma: A Graphical Simulation System*. Thomson South-Western, Mason, USA.

Sevinc, S. 1991. "Theories of Discrete Event Model Abstraction". In *Proceedings of the 1991 Winter Simulation Conference*, edited by B. L. Nelson, W. D. Kelton, and G. M. Clark, 1115–1119. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Simio 2017. "Simulation Tool". http://www.simio.com. [Online; accessed 2017-04-21].

Yücesan, E. 1989. *Simulation graphs for the design and analysis of discrete event simulation models*. Ph. D. thesis, Cornell University, Ithaca, USA.

Yücesan, E., and L. Schruben. 1992, January. "Structural and Behavioral Equivalnce of Simulation Models". *ACM Transitions on Modeling and Computer Simulation* 2 (1): 82–103.

Zeigler, B. P. 1984. *Theory of Modelling and Simulation*. Melbourne, USA: Krieger Publishing Company.

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. San Diego, USA: Academic Press.

AUTHOR BIOGRAPHIES

AKIN AKBULUT is research associate and PhD student at the department of Business Computing, especially CIM at the Heinz Nixdorf Institute, University of Paderborn. He studied business computing at the University of Paderborn. His email address is Akin.Akbulut@hni.upb.de.

STEPHAN ABKE is research associate and PhD student at the department of Business Computing, especially CIM at the Heinz Nixdorf Institute, University of Paderborn. His email address is Stephan.Abke@hni.upb.de.

CHRISTOPH LAROQUE studied business computing at the University of Paderborn, Germany. Since 2013 he is Professor of Business Computing at the University of Applied Sciences Zwickau, Germany. His email address is Christoph.Laroque@fh-zwickau.de.