Simulation in Produktion
und Logistik 2017
Sigrid Wenzel & Tim Peter (Hrsg.)
kassel university press, Kassel 2017

Automatisierte Modellverifikation mithilfe einer Äquivalenzprüfung an einem Referenzmodell für einen Einsatz in der Lehre

Automated Model Verification Using an Equivalence Test on a Reference Model for Use in Teaching

Akin Akbulut, Wilhelm Dangelmaier, Universität Paderborn, Paderborn (Germany), Akbulut@hni.upb.de, Wilhelm.Dangelmaier@hni.upb.de

Christoph Laroque, Westsächsische Hochschule Zwickau, Zwickau (Germany), Christoph.Laroque@fh-zwickau.de

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 Yucesan and Schruben (1992), demonstrating a procedure for the examination of a structural and behavioural equivalence of two simulation models based on Simulation Graph models. However, Simulation Graph models are subject to an event-oriented modelling world view. Since current simulation tools use a process-oriented - easier to understand - modelling world view, a simple queuing model shows how transformation from a process-oriented world view (Simio, AnyLogic) 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 Einleitung

An der Universität Paderborn wird eine Einführungsveranstaltung zur Modellierung und Simulation von Materialflusssystemen angeboten. Die Veranstaltung richtet sich an die Bachelorstudiengänge der Fakultät Wirtschaftswissenschaften und somit auch an Wirtschaftswissenschaftler und Wirtschaftsingenieure, die keine bis geringe Programmierkenntnisse besitzen. Die Inhalte der Veranstaltung beginnen mit der Theorie zur Modellbildung und schließen mit der Durchführung einer Simulationsstudie in Produktion und Logistik - dies erfordert auch die Implementierung eines Computermodells. Die von den Studierenden durchzuführende Studie beinhaltet eine Beschreibung eines geplanten Fertigungssystems, eine Problemstellung und eine Zielsetzung. Die Auswahl einer Simulationsanwendung, in der das Computermodell implementiert wird, treffen die Studierenden dabei eigenständig. Regelmäßig beobachten die Dozenten, dass Studierende, die in den

Ingenieur- und Wirtschaftswissenschaften verortet sind, problemlos ein adäquates Konzeptmodell für eine vorgegebene Problemstellung erstellen können, jedoch bei der Implementierung eines selbst einfachen Computermodells stark verunsichert sind Daher werden die Sprechstunden der Dozenten häufig von den Veranstaltungsteilnehmern besucht, um zu besprechen, ob ein Computermodell korrekt ist bzw. ob ein Konzeptmodell korrekt in ein Computermodell transformiert wurde. Da die Simulationsstudie an einem nichtexistierenden Fertigungssystem ohne Ergebnisdaten durchgeführt wird, sind die in der Literatur beschriebenen Verfahren (z. B. Rabe et al. 2008) zur Verifikation und Validierung nur bedingt anwendbar, da Studierende ihre Modellergebnisse (z. B. aus einer Kontrollrechnung) nicht einem Realsystem gegenüberstellen können. Eine Möglichkeit mit den vorhandenen Daten zu prüfen, ob ein Konzeptmodell korrekt in ein Computermodell transformiert wurde, kann über einen Vergleich mit einer Musterlösung bzw. einem Referenzmodell erfolgen. Eine solche manuelle Verifikation stellt jedoch eine zeitaufwendige Prozedur dar, weil die Struktur, alle Funktionen und alle weiteren Attribute beider Modelle miteinander verglichen werden müssen. Dieser Umstand hat zu der Frage geführt, wie ein automatisierter Ansatz für die Modellverifikation aufgebaut sein muss und welche Voraussetzungen gelten müssen, damit eine Methode entsteht, die in der Lehre eingesetzt werden kann. Eine solche Methode muss zwei Computermodelle miteinander vergleichen und eine Aussage treffen, ob beide Modelle ein und dasselbe zugrundeliegende Fertigungssystem modellieren. Da die Studierenden unterschiedliche Simulationsanwendungen benutzen, muss die Methode zusätzlich die Fähigkeit besitzen, Modelle aus zwei unterschiedlichen Simulationsanwendungen miteinander zu vergleichen.

2 Stand der Technik

Eine explizite Prüfung auf Gleichheit von zwei Simulationsmodellen wird in der Literatur als Äquivalenz von Modellen beschrieben und kommt im Zusammenhang mit Abstraktionstechniken für Simulationsmodelle vor. In diesem Kontext wird geprüft, ob ein abstrahiertes Modell immer noch dieselben oder ähnliche Ergebnisse berechnet wie vor einer Abstraktion. Overstreet (1982) differenziert zwei unterschiedliche Äquivalenzen: Eine Äquivalenz der Struktur (bzw. Strukturäquivalenz) und eine externe Äquivalenz. Demnach besteht eine Strukturäquivalenz, wenn die Mengen A und B, durch die zwei Modelle beschrieben werden, identisch sind. Eine externe Äquivalenz besteht, wenn die Eingaben U und die berechneten Ausgaben Y bei zwei Modellen identisch sind. Eine weitere Definition liefert Sargent (1988). Er betrachtet die von den Ereignissen berechneten Ausprägungen der Zustandsvariablen (S_{e_1}, S_{e_2}, ..., S_{e_n}) und die diskreten Zeitpunkte, an denen die Ereignisse auftreten (T_{e_1}, T_{e_2}, ..., T_{e_n}). Sind Ausprägungen und Zeitpunkte für beide Modelle identisch, so besteht nach Sargent eine Verhaltensäquivalenz. Yücesan und Schruben (1992) zeigen wie eine Verhaltensäquivalenz mithilfe von sogenannten Simulationsgraphen nachgewiesen werden kann. Interessant ist, dass der Nachweis über einen Umweg erfolgt. Sie prüfen die Struktur zweier Simulationsgraphen über eine isomorphe Abbildung zwischen zwei Modellen. Existiert eine solche Abbildung, so sind die Modelle strukturäquivalent. In einem nächsten Schritt beweisen sie, dass eine Strukturäquivalenz auch eine Verhaltensäquivalenz impliziert. Außer diesen theoretischen Behandlungen, existiert keine weitere Literatur, die auch eine praktische

Anwendung demonstriert. Um diese Wissenslücke zu schließen, stellen wir eine Methode mit einer Anwendung zur Verifikation von Simulationsmodellen vor, die auf Yücesan und Schruben (1992) aufbaut, um sie im Rahmen der Lehre einzusetzen. Aus diesem Grund soll als nächstes die Modellierungstechnik der Simulationsgraphen vorgestellt werden. Die Methode wird erweitert, sodass Modelle von aktuellen Simulationsanwendungen analysiert werden können, und zusätzlich ein Vergleich von Modellen aus unterschiedlichen Simulationsanwendungen möglich wird.

2.1 Simulationsgraphen

Simulationsgraphen sind eine mathematische Form von Ereignisgraphen, aus diesem Grund wird erst diese Modellierungstechnik erklärt. In Ereignisgraphen werden Simulationsmodelle mithilfe von gerichteten Graphen dargestellt, siehe Abbildung 1 (vgl. Schruben 1983). Ereignistypen werden durch Knoten dargestellt. Jeder Knoten besitzt eigene Zustandsvariablen und steht in Verbindung mit einer Reihe von Zustandsveränderungen, die in den eigenen Variablen durchgeführt werden, sobald ein Knoten aktiviert wird. Der Fokus bei Ereignisgraphen liegt auf den einzelnen Ereignissen in einem zu modellierenden System. Die Entitäten, wie z. B. ein diskreter Materialfluss, werden aus diesem Grund nur implizit dargestellt. Die Relationen zwischen den Ereignissen werden durch gerichtete Kanten zwischen zwei Knoten dargestellt. Kanten können logische und zeitliche Ausdrücke besitzen. Ein logischer Ausdruck (i) bestimmt die Bedingungen, die erfüllt werden müssen, damit eine Kante passiert werden kann. Ein zeitlicher Ausdruck t beschreibt, wie viel Zeit vergeht, wenn eine Kante passiert wird. Der abgebildete Ereignisgraph lässt sich wie folgt interpretieren: Immer wenn ein Ereignis A stattfindet und wenn die Bedingung (i) erfüllt ist, dann wird das Ereignis B in t Zeiteinheiten eingeplant.

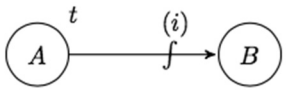

Abbildung 1: *Aufbau eines Ereignisgraphen*

Schruben beschreibt Simulationsgraphen als eine explizit mathematische Form von Ereignisgraphen und definiert sie als S = (F, C, T, Γ, G). Die ersten vier Mengen F, C, T, Γ definieren die Entitäten in einem Simulationsgraphen und damit das explizite Modellverhalten. Diese werden durch F die Ereignisfunktionen, C die Bedingungen, T die Zeitverzögerungen und Γ die Ausführungsprioritäten dargestellt. Der gerichtete Graph G als geordnetes Triple $(V(G), E(G), \Psi_G)$ beschreibt die Struktur eines Simulationsgraphen mit $V(G)$, um die Ereignisknoten zu beschreiben, $E(G)$, um die Kanten zu beschreiben, und der Menge Ψ_G, um die Verbindungen von Quellknoten nach Zielknoten zu beschreiben (s. Yücesan und Schruben 1992).

3 Transformation

Die Herausforderung bei einer Transformation liegt darin, dass ein Modell von einem Modellierungsparadigma in ein anderes Modellierungsparadigma übersetzt wird. Aktuelle Simulationswerkzeuge, wie z. B. AnyLogic und Simio, beschreiben Modelle

mithilfe von Prozessbausteinen. Eine Komponente in so einem Modell ist ein vollständiger Prozess, wie z. B. eine Montagezelle, die einen Montageprozess darstellt. Dieses Modellierungskonzept wird als prozessorientierte Weltsicht bezeichnet. Simulationsgraphen hingegen verwenden eine ereignisorientierte Weltsicht. Eine Modellkomponente in einem solchen Graphen stellt ein einzelnes Ereignis dar, wie z. B. eine „Ankunft von Material an einer Montagezelle“ (Law und Kelton 2000). Ein Prozess besteht i. d. R. aus vielen einzelnen Ereignissen, und so kann der Prozess „Montagezelle“ in der ereignisorientierten Weltsicht durch die Ereignisse „Ankunft von Material an Montagezelle“, „Bearbeitungsbeginn in Montagezelle“ und „Bearbeitungsende in Montagezelle“ beschrieben werden. Dies ist nur eine von vielen Möglichkeiten, einen Prozess über seine Ereignisse darzustellen. Die Information darüber, welche Ereignisse tatsächlich in einem Prozess zusammengefasst werden, liegt in den Ereignislisten in den jeweiligen Simulationswerkzeugen (zur Laufzeit) verborgen. Da AnyLogic und Simio keine Möglichkeiten bieten, ihre Ereignislisten zu analysieren, wird in dieser Arbeit manuell festgelegt, welche Ereignistypen hinter einem Prozesstyp verborgen liegen (s. Tabelle in Abb. 2). Die Tabelle wird anhand nachfolgendem Schema aufgebaut: Kann ein Prozesstyp eine Restriktion bezüglich ihrer Ressourcen besitzen (z. B. Ressourcen müssen in ausreichender Qualität und/oder Quantität vorhanden sein, um eine Bearbeitung in einem Prozessblock beginnen zu können), so wird er mit zwei Ereignistypen modelliert (siehe Arbeitsvorgang): ein Ereignistyp für die Ankunft, in der eine Ressource blockiert wird, und ein Ereignistyp für einen Abgang, in der eine Ressource wieder frei gegeben wird. Besitzt ein Prozesstyp eine Restriktion bezüglich seiner Kapazität zur Aufnahme von Entitäten, so wird er mit einem Ereignistypen Ankunft beschrieben (siehe Puffer). Die Prozesstypen Quelle und Senke werden immer mit einem Ereignistypen Ankunft beschrieben.

Prozesstyp	**Verhalten**	**Ereignistyp**
Quelle	Entität erzeugen Entität weiterleiten	Ankunft
Senke	Entität entgegennehmen Entität vernichten	Ankunft
Puffer	Entität entgegennehmen & einlagern Entität auslagern & weiterleiten	Ankunft
Arbeitsvorgang	Entität entgegennehmen Ressource binden Zeitverzögerung einplanen Ressource freigeben Entität weiterleiten	Ankunft Abgang

Abbildung 2: *Transformation elementarer Prozesstypen in Ereignistypen*

Mithilfe dieser Tabelle wird anschließend mit Algorithmus 1 (s. Abb. 3) in drei Stufen der neue Simulationsgraph kreiert. Die Funktionsweise des Algorithmus wird im Folgenden kurz erläutert:

1. *Initialisierung:* Aus einem Ausgangsmodell werden alle Prozessbausteine extrahiert und der Menge K zugewiesen. Alle Verknüpfungen aus dem Ausgangsmodell werden der Menge L zugewiesen. Anschließend werden die Mengen S, F,

C, T, Γ, E, V, G, Z deklariert und mit der leeren Menge initialisiert (Zeilen 1–3). Die Mengen S, F, C, T, Γ, E, V, G werden analog zu der Definition eines Simulationsgraphen gebildet. Die Menge Z hält die Zustandsvariablen eines Modells, die mit den Ereignisfunktionen F im Verlauf eines Simulationsexperiments berechnet werden.

2. *Knoten & lokale Eigenschaften:* Für jeden Prozessbaustein K werden die fest gelegten Ereignisknoten je Typ gebildet. Anschließend werden die lokalen Eigenschaften der Ereignisknoten gebildet und mit den Werten aus einer Modellkomponente parametrisiert. Lokale Eigenschaften sind alle Eigenschaften, die in keiner Beziehung zu einem vor- oder nachgestellten Prozessbaustein stehen (Zeilen 4–25).
3. *Kanten & globale Eigenschaften:* Für jede Verknüpfung L aus dem Ausgangsmodell wird entsprechend eine gerichtete Kante in dem Simulationsgraphen gebildet, um die Relationen korrekt zu übertragen. Nachdem alle Ereignisknoten mit den lokalen Eigenschaften generiert und die Verbindungen untereinander gesetzt wurden, werden in einem letzten Schritt alle Eigenschaften gesetzt, die in einer Abhängigkeit zu einem Vorgänger stehen. Dies betrifft die Ereignisfunktionen, Bedingungen und Zeitverzögerungen, die in Abhängigkeit von einem vorher geschalteten Prozessbaustein gesetzt werden (Zeilen 26–37).

Algorithmus 1 (Abb. 3) ist eine exemplarische Implementierung für AnyLogic. Der Algorithmus folgt den beschriebenen drei Stufen und kann auch für jedes weitere prozessorientierte Simulationswerkzeug in dieser Vorgehensweise programmiert werden. Der Algorithmus baut aus einem Ausgangsmodell einen neuen Simulationsgraphen S auf. Dabei werden die Prozessbausteine Source, Delay, Assembler, Service, Queue und Sink (Process Modeling Library) behandelt.

Data: Ausgangsmodell
Result: Simulationsgraph $\mathscr{S}$
$K \leftarrow$ Modellkomponenten $\subseteq$ Ausgangsmodell;
$L \leftarrow$ Verknüpfungen $\subseteq$ Ausgangsmodell;
$\mathscr{S}, \mathscr{F}, \mathscr{C}, \mathscr{T}, \Gamma, E, V, G, Z \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** $|K|$ **do**
  switch $k_i.Typ$ **do**
    case *Source* **do**
      $V \leftarrow V \cup \{v_i \leftarrow k_i.Typ\}$;
      $Z \leftarrow Z \cup \{z_{i,x\leftarrow S} \leftarrow 0\}$;
      $E \leftarrow E \cup \{e_i \leftarrow (v_i,\, v_i)\}$;
      $\mathscr{T} \leftarrow \mathscr{T} \cup \{t_{e_i} \leftarrow k_i.Zeitverzögerung\}$;
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_i}\} = \{z_i \leftarrow z_i + k_i.AnkunftMaterialInStk\}$;
    case *Delay* | *Assembler* | *Service* **do**
      $V \leftarrow V \cup \{v_{i,x\leftarrow 1} \leftarrow k_i.Typ\} \cup \{v_{i,x\leftarrow 2} \leftarrow k_i.Typ\}$;
      $Z \leftarrow Z \cup \{z_{i,x\leftarrow R} \leftarrow 0\} \cup \{z_{i,x\leftarrow S} \leftarrow 0\}$;
      $E \leftarrow E \cup \{e_{i,x\leftarrow 1} \leftarrow (v_{i,1},\, v_{i,2})\}$;
      $E \leftarrow E \cup \{e_{i,x\leftarrow 2} \leftarrow (v_{i,2},\, v_{i,1})\}$;
      $\mathscr{T} \leftarrow \mathscr{T} \cup \{t_{e_{i,1}} \leftarrow k_i.Zeitverzögerung\}$;
      $z_{i,R} \leftarrow k_i.RessourcenInStk$;
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_{i,1}}\} = \{z_{i,R} \leftarrow z_{i,R} - 1\}$;
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_{i,2}}\} = \{z_{i,R} \leftarrow z_{i,R} + 1,\, z_{i,S} \leftarrow z_{i,S} + 1\}$;
    case *Queue* | *Sink* **do**
      $V \leftarrow V \cup \{v_i \leftarrow k_i.Typ\}$;
      $Z \leftarrow Z \cup \{z_{i,x\leftarrow R} \leftarrow 0\} \cup \{z_{i,x\leftarrow S} \leftarrow 0\}$;
      $z_{i,R} \leftarrow k_i.KapazitätInStk$;
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_i}\} = \{z_{i,S} \leftarrow z_{i,S} + 1\}$;
for $j \leftarrow 1$ **to** $|L|$ **do**
  $E \leftarrow E \cup \{e_{h\leftarrow |E|+1} \leftarrow (v_x \leftarrow \text{Quelle}(l_j),\, v_y \leftarrow \text{Ziel}(l_j))\}$;
for $k \leftarrow 2$ **to** $|V|$ **do**
  switch v_k **do**
    case *Delay* | *Assembler* | *Service* **do**
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_k}\} = \{z_{k-1,S} \leftarrow z_{k-1,S} - 1\}$;
      $\mathscr{C} \leftarrow \mathscr{C} \cup \{c_{(v_{k-1},v_k)}\} = \{z_{k-1,R} > 0\}$;
      $\mathscr{C} \leftarrow \mathscr{C} \cup \{c_{(v_{k+1},v_k)}\} = \{z_{k,R} > 0\}$;
      $k \leftarrow k + 1$
    case *Queue* | *Sink* **do**
      $\mathscr{F} \leftarrow \mathscr{F} \cup \{f_{v_k}\} = \{z_{k-1,S} \leftarrow z_{k-1,S} - 1\}$;
      $\mathscr{C} \leftarrow \mathscr{C} \cup \{c_{(v_{k-1},v_k)}\} = \{z_{k,S} < z_{k,R}\}$;
$G \leftarrow (V, E)$;
$\mathscr{S} \leftarrow (\mathscr{F}, \mathscr{C}, \mathscr{T}, \Gamma, G)$;

Abbildung 3: *Algorithmus 1 - Transformation in einen Simulationsgraphen*

Um das Vorgehen zu verdeutlichen, wird hier ein einfaches Warteschlangenmodell beschrieben, das mit den Werkzeugen AnyLogic und Simio implementiert ist. Beide Implementierungen sollen in Simulationsgraphen transformiert werden. Das Warteschlangenmodell besteht aus einer Materialankunft (x Stück in einem Intervall a), einem Warteraum (Kapazität y Stück) und einer Bedienstation die sequenziell arbeitet (Ressource 1, Bearbeitungszeit s). Ein korrektes Computermodell in dem

Simulationswerkzeug AnyLogic besteht, wie in Abbildung 4 (Links) dargestellt, aus den Prozessbausteinen vom Typ Source, Queue, Delay und Sink.

Abbildung 4: *(Links) Modell A – Warteschlangenmodell in AnyLogic; (Rechts) Modell B – Warteschlangenmodell in Simio*

Ein solches Modell als Ausgangsmodell für Algorithmus 1 erzeugt einen Simulationsgraphen S_A wie in Abbildung 5 dargestellt wird.

$G_A(V)= \{A_0, A_1, A_2, A_3, A_4, A_5\}$,
$G_A(E)= \{E_{(A_0,A_1)}, E_{(A_1,A_1)}, E_{(A_1,A_2)}, E_{(A_2,A_3)}, E_{(A_3,A_4)}, E_{(A_4,A_3)}, E_{(A_4,A_5)}\}$,
$\Psi_A=\{(A_0, A_1), (A_1, A_1), (A_1, A_2), (A_2, A_3), (A_3, A_4), (A_4, A_3), (A_4, A_5)\}$,
$\mathscr{F}_A= \{f_{A_0}; f_{A_1}; f_{A_2}; f_{A_3}; f_{A_4}; f_{A_5}\} = \{ z_1 \leftarrow z_1 + x; z_2 \leftarrow z_2 + x, z_1 \leftarrow z_1 - x;$
$z_3 \leftarrow 0, z_2 \leftarrow z_2 - 1; z_3 \leftarrow 1, z_4 \leftarrow z_4 + 1; z_5 \leftarrow z_5 + 1, z_4 \leftarrow z_4 - 1 \}$,
$\mathscr{C}_A = \{C_{(A_1,A_2)}; C_{(A_2,A_3)}; C_{(A_4,A_3)}\} = \{z_2 < y; z_3 > 0; z_2 > 0\}$,
$\mathscr{T}_A = \{t_{(A_1,A_1)}; t_{A_3,A_4} \} = \{\text{a, s}\}$,
$\Gamma_A = \emptyset$.

Abbildung 5: *Darstellung von Modell A als Simulationsgraphen S_A*

In dem Simulationswerkzeug Simio besteht ein adäquates Modell aus den Prozessbausteinen Source, Server und Sink (Abb. 4, Rechts). Ein auf Simio angepasster Algorithmus erzeugt einen Simulationsgraphen S_B wie in Abbildung 6 dargestellt wird.

$G_B(V)= \{B_0, B_1, B_2, B_3, B_4, B_5\}$,
$G_B(E)= \{E_{(B_0,B_1)}, E_{(B_1,B_1)}, E_{(B_1,B_2)}, E_{(B_2,B_3)}, E_{(B_3,B_4)}, E_{(B_4,B_3)}, E_{(B_4,B_5)}\}$,
$\Psi_B=\{(B_0, B_1), (B_1, B_1), (B_1, B_2), (B_2, B_3), (B_3, B_4), (B_4, B_3), (B_4, B_5)\}$,
$\mathscr{F}_B= \{f_{B_0}; f_{B_1}; f_{B_2}; f_{B_3}; f_{B_4}; f_{B_5}\} = \{ z_1 \leftarrow z_1 + x; z_2 \leftarrow z_2 + x, z_1 \leftarrow z_1 - x;$
$z_3 \leftarrow 0, z_2 \leftarrow z_2 - 1; z_3 \leftarrow 1, z_4 \leftarrow z_4 + 1; z_5 \leftarrow z_5 + 1, z_4 \leftarrow z_4 - 1 \}$,
$\mathscr{C}_B = \{C_{(B_1,B_2)}; C_{(B_2,B_3)}; C_{(B_4,B_3)}\} = \{z_2 < y; z_3 > 0; z_2 > 0\}$,
$\mathscr{T}_B = \{t_{(B_1,B_1)}; t_{(B_3,B_4)}\} = \{\text{a, s}\}$,
$\Gamma_B = \emptyset$.

Abbildung 6: *Darstellung von Modell B als Simulationsgraphen S_B*

4 Äquivalente Simulationsmodelle

Zwei Simulationsgraphen $S_A = (F_A, C_A, T_A, \Gamma_A, G_A)$ und $S_B = (F_B, C_B, T_B, \Gamma_B, G_B)$ sind isomorph zueinander $S_A \approx S_B$, wenn eine bijektive Abbildung zwischen S_A und S_B wie folgt existiert (Yücesan und Schruben 1992):

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

Abbildung 7: *Bijektive Abbildung zwischen S_A und S_B*

Die Abbildungen Θ, Φ, Λ, Ω, χ, Δ formen zusammen eine Isomorphie zwischen den Simulationsgraphen S_A und S_B. Die ersten zwei Abbildungen Θ und Φ prüfen eine Übereinstimmung in der Struktur zwischen den Knoten und Kanten der Graphen. Als nächstes werden die verbleibenden vier Abbildungen im Detail erklärt.

Die Abbildung Λ: $F_A \rightarrow F_B$ prüft eine Übereinstimmung zwischen den Ereignisfunktionen, die für die Berechnung der Systemzustände verantwortlich sind. Damit zwei Ereignisfunktionen als gleichwertig eingestuft werden, müssen ihre Berechnungen identische Ergebnisse liefern. In den Fällen, in denen eine Funktion eine stochastische Komponente besitzt, müssen die generierten Zufallszahlen nicht identisch sein, sie müssen jedoch identischen stochastischen Verteilungen unterliegen.

Die Abbildung Ω: $C_A \rightarrow C_B$ prüft eine Übereinstimmung zwischen den Bedingungen von zwei unterschiedlichen Modellen. Dabei müssen die Bezeichnungen für die Ausdrücke und die binären Operatoren für die Bedingungen in den unterschiedlichen Simulationsgraphen nicht identisch sein. Wichtig ist ausschließlich, dass die Bedingungen identische booleschen Werte annehmen, sobald in zwei unterschiedlichen Simulationsgraphen ein identischer Modellzustand vorliegt.

Die Abbildung χ: $T_A \rightarrow T_B$ prüft, ob eine Übereinstimmung zwischen den Verzögerungszeiten vorliegt. Wenn die Verzögerungszeiten deterministisch sind, müssen sie in den unterschiedlichen Modellen identisch sein. Wenn die Verzögerungszeiten stochastisch sind, muss sichergestellt werden, dass sie denselben stochastischen Verteilungen unterliegen, damit sie als gleichwertig eingestuft werden können.

Die Abbildung Δ: $\Gamma_A \rightarrow \Gamma_B$ prüft eine Übereinstimmung zwischen den Ausführungsprioritäten. Da die Ausführungsprioritäten in dieser Arbeit nicht beachtet werden, sind beide Mengen leer.

Allgemein ist hinzuzufügen, dass für die Prüfung einer Isomorphie zwischen zwei Graphen derzeit kein effizienter Algorithmus bekannt ist (Deiser et al. 2016). Simulationsgraphen stellen jedoch planare Graphen dar, d. h. diese Graphen können auf einer Ebene dargestellt werden, ohne dass sich Kanten schneiden (Yücsesan 1989). Für die Prüfung einer Isomorphie bei planaren Graphen werden in der Literatur unterschiedliche Algorithmen vorgestellt. Die hier vorgestellte Implementierung (s. Abb. 8) orientiert sich an der theoretischen Beschreibung. Die Simulationsgraphen werden erst dahingehend geprüft, ob sie planar sind, und dann wird geprüft, ob eine isomorphe Abbildung zwischen ihnen besteht. Hierzu wird die Methode verwendet, wie sie von Kukluk et al. (2004) vorgestellt wird. Wenn die Abbildungen Θ und Φ hergestellt werden können, liefert die Funktion *checkIsomorphism* eine positive Rückmeldung. Anschließend werden die Mengen F, C, T und Γ auf Übereinstimmung geprüft. Hierzu werden alle Zuweisungen ($\leftarrow$) und binäre Operationen ($+$, $-$, $<$, $=$, $>$) geprüft, ob sie in gleicher Anzahl in beiden Mengen existieren. Trifft dies zu, so gibt die Funktion *equals* eine positive Rückmeldung. Der Algorithmus 2 (s. Abb. 8) gibt nach Abschluss fünf boolesche Variablen aus. Erst wenn alle Variablen den Wert „true“ enthalten, so ist eine Prüfung vollständig und erfolgreich abgeschlossen.

Data: Simulationsgraph $\mathscr{S}_A$, Simulationsgraph $\mathscr{S}_B$
Result: Boolean, Boolean, Boolean, Boolean, Boolean
1 a, b, c, d, e $\leftarrow false$;
2 a $\leftarrow checkIsomorphsim(G(\mathscr{S}_A), G(\mathscr{S}_B))$;
3 b $\leftarrow equals(\mathscr{F}(\mathscr{S}_A), \mathscr{F}(\mathscr{S}_B))$;
4 c $\leftarrow equals(\mathscr{C}(\mathscr{S}_A), \mathscr{C}(\mathscr{S}_B))$;
5 d $\leftarrow equals(\mathscr{T}(\mathscr{S}_A), \mathscr{T}(\mathscr{S}_B))$;
6 e $\leftarrow equals(\Gamma(\mathscr{S}_A), \Gamma(\mathscr{S}_B))$;
7 **return** a, b, c, d, e

Abbildung 8: *Isomorphie Simulationsgraphen*

Wenn Modell A und Modell B in separate Simulationsgraphen S_A und S_B transformiert sind, kann mit Algorithmus 2 geprüft werden, ob eine isomorphe Abbildung zwischen beiden Modellen hergestellt werden kann (s. Abb. 8). Nachdem eine isomorphe Abbildung zwischen S_A und S_B identifiziert werden konnte, wird angenommen, das Modell A und Modell B struktur- und verhaltensäquivalent sind.

5 Implementierung und Einbettung in eine Webplattform für eine Anwendung in der Lehre

Die Methode wird als Anwendung implementiert und in eine Webplattform integriert. Der Aufbau entspricht einer klassischen SaaS (Software as a Service) Architektur, wie sie in Abbildung 6 dargestellt wird. Studierende können auf der Plattform über einen Browser auf unterschiedliche Aufgabenstellungen zugreifen. Jede dieser Aufgabenstellungen besteht aus einer fiktiven Beschreibung eines Fertigungssystems, einer Problemstellung, einer Zielsetzung und einem Referenzmodell. Das Referenzmodell stellt eine Musterlösung dar, auf die die Studierenden jedoch keinen Zugriff erhalten. Die Auswahl einer Simulationsanwendung, in der ein Computermodell implementiert wird, treffen die Studierenden eigenständig.

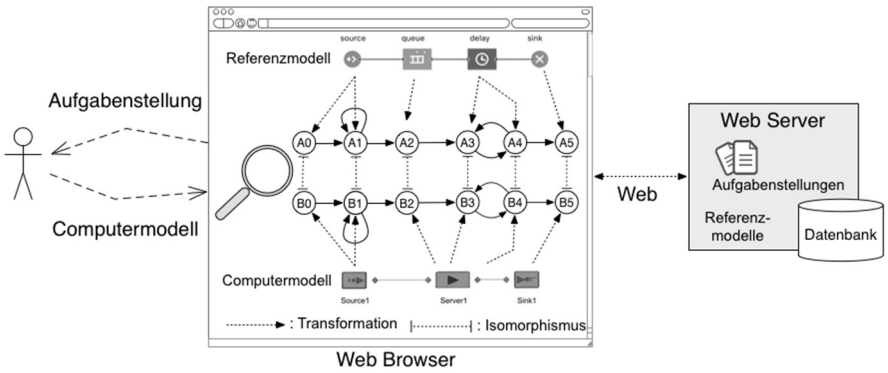

Abbildung 9: *Aufbau der Implementierung*

Die Studierenden werden aufgefordert für eine Aufgabenstellung ein Computermodell zu implementieren und nach der Fertigstellung abzuspeichern. Ein so gespeichertes Computermodell wird für die Verifikation in die Web-Plattform hochgeladen und mit einem XML-Parser eingelesen. Das eingelesene Modell wird mit dem Algorithmus 1 in einen Simulationsgraphen transformiert. Mit Algorithmus 2 wird das Modell mit einem Referenzmodell aus einer Datenbank auf eine bestehende isomorphe Abbildung geprüft. Die Algorithmen werden dabei clientseitig ausgeführt, dazu kommt die Programmiersprache JavaScript zum Einsatz. Sobald eine Isomorphie identifiziert werden kann ($S_A \approx S_B$), wird angenommen, dass Modell A und Modell B strukturäquivalent sind. Da Yücesan und Schruben (1992) gezeigt haben, dass aus einer Strukturäquivalenz auch eine Verhaltensäquivalenz folgt, wird angenommen, dass so identifizierte Simulationsmodelle zur Laufzeit dieselben Ergebnisse Y bei denselben Eingaben U berechnen – ergo ein Computermodell für eine vorgegebene Aufgabenstellung korrekt implementiert ist.

6 Zusammenfassung

In dieser Arbeit wird eine werkzeugübergreifende Methode zur automatisierten Modellverifikation mithilfe eines Referenzmodells vorgestellt und gezeigt, wie die Methode mit einer Web-Plattform in der Lehre eingesetzt werden kann. Für diese Methode greifen wir auf die Arbeit von Yücesan und Schruben (1992) zurück, in der basierend auf Simulationsgraphen ein Vorgehen für die Prüfung einer Struktur- und Verhaltensäquivalenz von zwei Simulationsmodellen demonstriert wird.

Literatur

Deiser, O.; Lasser, C.; Vogt, E.; Werner, D.: 12 x 12 Schlüsselkonzepte zur Mathematik - Diskrete Mathematik. Berlin, Heidelberg: Springer 2016.

Kukluk, J.P.; Holder, L.B.; Cook, D.J.: Algorithm and experiments in testing planar graphs for isomorphism. J. Graph Algorithms Appl. 8 (2004), 2, S. 313–356.

Law, A.M.; Kelton, W.D.: Simulation Modeling and Analysis. 3. McGraw-Hill, 2000.

Overstreet, C.M.: Model Specification and Analysis for Discrete Event Simulation. Blacksburg, Virginia, Virginia Tech., Diss., 1982.

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

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

Schruben, L.: Simulation Modeling with Event Graphs. Commun. ACM 26 (1983) 11, S. 957–963.

Yücesan, E.; Schruben, L.: Structural and Behavioral Equivalence of Simulation Models. ACM Transitions on Modeling and Computer Simulation 2 (1992) 1, S. 82–103.

Yücesan, E.: Simulation graphs for the design and analysis of discrete event simulation models. Ithaca, N.Y., Cornell University, Diss., 1989.