Dynamische Co-Simulation von Automatisierungssystemen und ihren Komponenten im Internet der Dinge

T. Jung, M.Sc.; Dr.-Ing. **N. Jazdi**; Univ.-Prof. Dr.-Ing. **M. Weyrich**,
Universität Stuttgart Institut für Automatisierungstechnik und Softwaresysteme, Stuttgart

Kurzfassung

Heute wird Simulation während der Entwicklung und Inbetriebnahme von Automatisierungssystemen genutzt, allerdings weitet sich die Nutzung von Simulation durch die Einführung des „Digitalen Zwillings“ auch auf den Betrieb aus. Durch diese neuen Einsatzmöglichkeiten der Simulation ergeben sich, neue Herausforderungen, gerade bei der Simulation von IoT-Systemen, da diese dynamische und heterogene Systeme sind. Durch die hohe Dynamik (Ein- und Austreten der Komponenten) und Heterogenität (stark unterschiedliche Komponenten) in IoT-Systemen ist es nicht mehr möglich alle Komponenten mit allen relevanten Aspekten in einem Simulationstool zu simulieren, weshalb eine Co-Simulation benötigt wird. Um der hohen Dynamik des realen Systems gerecht zu werden, wird auch ein Konzept benötigt um „Plug-and-Simulate“ zu ermöglichen.

In diesem Beitrag werden zuerst existierende Ansätze zur Simulation von IoT-Systemen sowie zur Co-Simulation vorgestellt und auf ihre Tauglichkeit hinsichtlich der Simulation von heterogenen und dynamischen IoT-Systemen untersucht. Anschließend wird ein neuartiges Konzept vorgestellt, das ein Eintreten von neuen, zur Entwurfszeit unbekannten Modellen zur Laufzeit der Simulation in eine Co-Simulation ermöglicht. Abschließend wird eine prototypische Umsetzung des Konzepts am Beispiel der Co-Simulation eines IoT-Warenlagers vorgestellt.

Abstract

Nowadays simulation is used during development and commissioning of automation systems, but by the introduction of the “Digital Twin” the use of simulation is expanded towards operation. New challenges arise through those new applications of simulation, especially for simulating IoT-systems, as such systems are very dynamic and heterogeneous. Because of the dynamic (entering and leaving of components during runtime) and heterogeneity (different components) it will not be possible to simulate all

components with all relevant aspects in a single simulation, hence a co-simulation is needed. Additionally to do justice to the dynamic of the real systems, a concept is needed to realize “Plug-and-Simulate”.

This contribution first compares different co-simulation approaches for the simulation of dynamic and heterogeneous IoT-systems. Afterwards a new concept is introduced, which enables a dynamic entering of, during development, unknown models during runtime into a co-simulation. Finally, a prototypical implementation of the concept is presented by the example of an IoT-warehouse.

1. Einleitung

Durch die zunehmende Vernetzung und die Verwendung von cyber-physischen Systemen entstehen immer mehr ad-hoc Netzwerke und IoT-Systeme, welche zur Laufzeit ihre Struktur und Aufgaben verändern, indem neue Komponenten eintreten oder andere Komponenten austreten. Allerdings bestehen diese Systeme aus sehr heterogenen Komponenten von verschiedenen Herstellern und aus verschiedenen Domänen. Um den Ein- und Austritt von Komponenten zu erleichtern wurden „Plug-and-Play“-Konzepte vorgestellt.

Parallel zu dieser Entwicklung weitet sich die Nutzung von Simulation von der Entwicklung von Systemen auf den Betrieb von Systemen aus. Insbesondere die Einführung des „Digitalen Zwillings“ ermöglicht eine Simulation, welche ständig parallel zum realen System läuft. Bei einer Simulation von IoT-Systemen parallel zum realen System übertragen sich die Herausforderungen der Dynamik und Heterogenität auch auf die Simulation des IoT-Systems. Daher wird um eine einfache Nutzung der Simulation zu ermöglichen ein „Plug-and-Play“-Konzept, „Plug-and-Simulate“, für die Simulation benötigt.

In [1] werden mehrere Simulationskonzepte zur Simulation von IoT-Systemen untersucht und geschlussfolgert, dass es nicht möglich sein wird, ein IoT-System mit allen relevanten Aspekten in einem einzelnen Simulationstool zu simulieren.

Daher wird ein neuartiges Konzept zur Simulation von IoT-Systemen benötigt, welches folgende Anforderungen erfüllen muss:

A1: Der Ein- und Austritt von Modellen zur Simulationszeit muss möglich sein, um Systeme simulieren zu können, in denen ein Ein- und Austritte von Komponenten zur Laufzeit stattfindet [1].

A2: Es muss möglich sein, verschiedene Simulationstools zu verwenden um die IoT-Komponenten zu modellieren, da heterogene IoT-Systeme aus verschiedenen Komponenten bestehen, welche in unterschiedlichen Simulationstools modelliert werden müssen, um alle relevanten Aspekte berücksichtigen zu können.

A3: Das Simulationskonzept muss während jeder Lebenszyklusphase eines Systems einsetzbar, da eine durchgängige Simulation über den gesamten Lebenszyklus immer mehr an Bedeutung gewinnt [2].

A4: Das Simulationskonzept muss domänenunabhängig sein, da in IoT-Systemen eine Vernetzung über mehrere Domänen hinweg stattfindet.

A5: Es muss möglich sein, den Modellen Intelligenz hinzufügen zu können, da der „Digitale Zwilling“ mehr ist als eine ultra-realistische Simulation der Realität. Der „Digitale Zwilling“ kann durch das Hinzufügen von Intelligenz auch zur Entscheidungsunterstützung, Systemoptimierung und für Predictive Maintenance genutzt werden.

In Kapitel 2 werden zunächst existierende Co-Simulationsansätze untersucht und in Hinblick auf die Erfüllung der genannten Anforderungen bewertet. Kapitel 3 stellt ein neuartiges Konzept zur Co-Simulation von IoT-Systemen vor und Kapitel 4 zeigt eine prototypische Umsetzung dieses Konzepts anhand eines Warenlagerszenarios. Abschließend evaluiert Kapitel 5 anhand dieser prototypischen Realisierung das Konzept.

2. Existierende Co-Simulationsansätze

Es wurden sowohl domänenspezifische und domänenübergreifende Co-Simulationsstandards als auch Stand-alone-Ansätze, welche Simulationstools koppeln, untersucht.

2.1 Functional Mock-Up Interface (FMI)

Functional Mock-Up Interface bietet neben einem Standard für Co-Simulationen auch die Möglichkeit Modelle zwischen Simulationstools auszutauschen [3]. Ein Nachteil von FMI ist allerdings, dass die Simulationstools FMI unterstützen müssen, wenn sie an einer FMI-Co-Simulation teilnehmen möchten [4]. Zudem ist der Datenaustausch zwischen den Teilsimulationen auf diskrete Zeitpunkte beschränkt. Dieser Datenaustausch wird von einem Masteralgorithmus koordiniert, welcher auch für die Synchronisierung der Teilsimulationen, auch Simulationsslaves genannt, zuständig ist. Um an einer Co-Simulation teilnehmen zu können, müssen die einzelnen Teilsimulation durch Functional Mock-Up Units repräsentiert werden, welche eine Schnittstelle zu den Simulationstools realisieren [3].

Es existiert eine Vielzahl an Realisierungen von Co-Simulationen mit FMI, wie beispielsweise [5], wo die Fähigkeiten von FMI diskutiert werden. Hierbei wird aufgezeigt, dass der Masteralgorithmus vor Simulationsbeginn Informationen über jede Teilsimulation benötigt, wodurch „Plug-and-Simulate“ nicht praktikabel mit FMI ist.

2.2 High Level Architecture (HLA)

HLA ist eine vom amerikanischen Verteidigungsministerium entwickelte Co-Simulationsarchitektur für verteilte und parallele Simulationen [6]. Eine Co-Simulation mit HLA, Föderation genannt, besteht aus Föderaten, den einzelnen Teilsimulationen, und der Run-Time-Infrastructure (RTI), einer zentralen Einheit zur Koordination der Co-Simulation. In Interface-Specifications werden die Schnittstelle zwischen den Föderaten und der RTI definiert und ein Object-Model-Template definiert die Informationen, die zwischen den Föderaten ausgetauscht werden. Ein Satz an HLA-Regeln beschreibt Anforderungen, die eine Simulation erfüllen muss, um an einer Co-Simulation teilzunehmen. Die RTI kann als Simulationsmaster angesehen werden und regelt neben dem Datenaustausch zwischen den Föderaten auch deren Synchronisierung [7]. Trotz dem, dass HLA den Eintritt von Föderaten zur Laufzeit erlaubt [8], muss für jeden neuen Föderat ein Federation Agreement geschrieben werden, welches domänen- und anwendungsfallspezifisch ist. Es existieren mehrere Realisierungen von RTIs, sowohl kommerzielle als auch frei verfügbare [9].

2.3 Co-Simulation mit OPC UA

OPC UA ist ein serviceorientierter Machine-to-Machine-Kommunikationsstandard, welcher die Übertragung von Prozessdaten und deren maschinenlesbaren Beschreibungen ermöglicht [10]. In [11] wird OPC UA dazu verwendet, eine Co-Simulation zu realisieren, indem jede Teilsimulation durch eine Schnittstelle an einen generischen Adapter angeschlossen, welcher jeweils einen OPC-UA-Server und einen OPC-UA-Client enthält. Dieser Adapter kommuniziert mittels OPC UA mit einem zentralen Server, welcher ebenfalls aus einem OPC-UA-Server und einem OPC-UA-Client besteht. Jeder Simulator muss sich an diesem Server registrieren, wobei der erste registrierte Simulator der Simulationsmaster ist und alle folgenden Simulatoren Simulationsslaves sind. Der Master koordiniert und synchronisiert die Co-Simulation. Falls der Master die Co-Simulation verlässt übernimmt ein anderer Simulator die Rolle des Masters.

2.4 Co-Simulation mit OSGi

OSGi ist ein Framework der Open-Services-Gateway-initiative, welches, basierend auf Java, die Entwicklung eines dynamischen Komponentensystems mit dynamisch kombinierbaren und wiederverwendbaren Komponenten ermöglicht. Um die Komplexität zu reduzieren, werden in OSGi die einzelnen Komponenten in sogenannten Bundels gekapselt und interagieren über Services miteinander [12]. Diese Bundels können zur Laufzeit geladen oder entfernt werden und können sich auf einem oder verteilt auf mehreren Rechnern befinden [13]. Der dynamische Austausch von Bundels ermöglicht eine dynamische Co-

Simulation, wie sie in [14] umgesetzt wurde. Hier wird jede Simulation durch ein Bundel repräsentiert, welches durch einen Simulation-Coupler an das Simulationstool angebunden ist. Der Simulation-Coupler verwendet OPC um eine Synchronisierung und einen Datenaustausch zu ermöglichen.

2.5 Domänenspezifische Ansätze

Neben den domänenübergreifenden Ansätze und Standards gibt es noch eine Vielzahl an domänenspezifischen Co-Simulationsansätzen. Einer der bekanntesten ist CAPE-OPEN (Computer-Aided Process Engineering) welcher in der Prozessindustrie eingesetzt wird [15], welcher aber keinen dynamischen Austausch von Simulationen zur Laufzeit ermöglicht [8]. Neben CAPE-OPEN gibt es noch ein breites Spektrum an Ansätzen wie beispielsweise EPOCHS [16], Mosaik [17] und ADEVS [18], welche zur Simulation von Energienetzen verwendet werden. Da diese Standards und Ansätze allerdings alle domänenspezifisch sind erfüllen sie nicht die Forderung nach einem domänenübergreifenden Konzept.

2.6 Vergleich der Ansätze

Tabelle 1 vergleicht die vorgestellten Ansätze hinsichtlich der in Kapitel 1 aufgestellten Anforderungen. Erfüllt ein Ansatz eine Anforderung komplett, ist dies mit „+“ gekennzeichnet, wird eine Anforderung nur teilweise erfüllt, ist dies mit „0“ gekennzeichnet und wird eine Anforderung nicht erfüllt, ist dies mit „-“ gekennzeichnet.

Tabelle 1: Vergleich der Co-Simulationsansätze

Co-Simulationsansatz	A1	A2	A3	A4	A5
FMI	-	0	+	0	-
HLA	+	-	+	0	-
OPC UA	+	0	+	+	0
OSGi	+	0	+	+	0
domänenspezifisch	0	-	+	-	-

Aus Tabelle 1 wird ersichtlich, dass kein Ansatz alle Anforderungen völlig erfüllt. Nur mit HLA, OPC UA und OSGi ist es möglich zur Laufzeit Simulatoren in die Co-Simulation einzubinden. Bei HLA muss allerdings für jede neue Simulation ein neues Federation Agreement geschrieben werden. FMI und HLA unterstützen zwar eine Vielzahl an Simulationstools, wurden aber für spezifische Domänen entwickelt und sind daher nicht gänzlich domänenübergreifend. Keiner der Ansätze bietet zudem die Möglichkeit Intelligenz

hinzuzufügen. Insbesondere bei FMI und HLA wären größere Anpassungen nötig, da beide Standards sind, die diese Möglichkeit nicht vorsehen. Bei den Ansätzen mit OPC UA und OSGi wäre es prinzipiell möglich, Intelligenz hinzuzufügen, allerdings wären auch hier Anpassungen am Konzept nötig.
Da keiner der präsentierten Ansätze mehr als 3 der genannten Anforderungen erfüllt, wird ein neuer Ansatz zur Co-Simulation von IoT-Systemen benötigt.

3. Dynamische, agentenbasierte Co-Simulation

Der Ansatz eines Agentensystems wurde gewählt um eine dynamische Co-Simulation zu realisieren, da Agenten zur Laufzeit in ein Agentensystem eintreten können, domänen-lebenszyklusphasenunabhängig sind und die Möglichkeit Intelligenz hinzuzufügen vorsehen [19]. Das Konzept Agenten zur Co-Simulation zu verwenden, wurde bereits in [1] vorgestellt. Hierbei wird jede IoT-Komponente in einer eigenen Simulation simuliert und durch einen Agenten repräsentiert, siehe Bild 1. Durch die Modellierung und Simulation der IoT-Komponenten in individuellen Simulationen und durch die Repräsentation durch Agenten, können die einzelnen Simulationen zur Laufzeit ein- und austreten, wie die realen IoT-Komponenten. Der Datenaustausch findet über das Agentensystem statt.

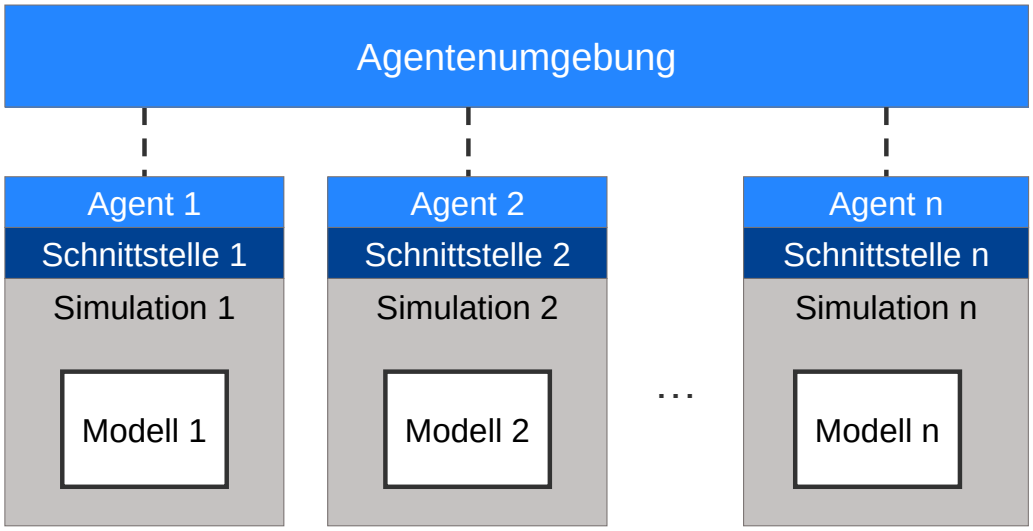

Bild 1: Agentenbasierte Co-Simulation von IoT-Systemen

3.1 Schnittstelle zwischen Agenten und Simulationstools

Um einen Datenaustausch zwischen den einzelnen Simulationen zu ermöglichen, wird eine Schnittstelle zwischen den Simulationstools und den Agenten benötigt. Die Daten, die unter den Simulationen ausgetauscht werden müssen können in kommunikationsorientierte und prozessorientierte Daten unterteilt werden. Kommunikationsorientierte Daten stellen die Nachrichten, die im IoT-System über Kommunikationskanäle versendet werden, dar und die

prozessorientierten Daten simulieren die prozessorientierte Interaktion im IoT-System. Die prozessorientierten Interaktionen in einem IoT-System sind die physikalischen Wechselwirkungen im IoT-System. Erhitzt beispielsweise eine Heizung einen Raum, so hat dies eine Auswirkung auf einen sich im Raum befindlichen Temperatursensor. Somit interagieren Heizung und Temperatursensor über den Prozess Erhitzen miteinander. Für beide Arten der Interaktion wird eine Schnittstelle zwischen Agenten und Simulationstools benötigt, siehe Bild 2. Eine Unterteilung der Interaktionen ist sinnvoll, da die kommunikationsorientierte Interaktion über die im realen System verwendeten Kommunikationsprotokolle genau spezifiziert sind, die prozessorientierte Interaktion hingegen abstrahiert werden muss, da für die physikalische Welt keine Spezifikation vorliegt.
Um eine gewisse Wiederverwendbarkeit der Schnittstellen zu ermöglichen, werden die Schnittstellen in einen spezifischen und einen generischen Teil unterteilt, siehe Bild 2. Der spezifische Teil ist auf das jeweilige Simulationstool angepasst und muss für jedes Tool einmal entwickelt werden. Werden mehrere Modelle mit dem gleichen Tool modelliert, kann dieser spezifische Teil für jedes dieser Modelle wiederverwendet werden. Der generische Teil bildet die Schnittstelle nach oben zum Agenten und kann für alle Simulationstools verwendet werden.

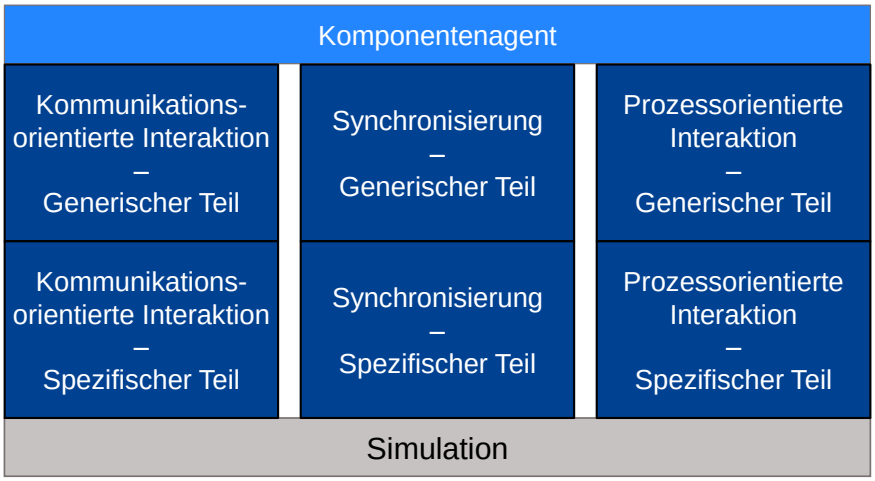

Bild 2: Schnittstelle zwischen Agent und Simulation

3.2 Datenaustausch, Synchronisierung und Nachrichtenreduktion

Um die prozessorientierte Interaktion zu simulieren, wird zusätzlich eine Simulation der Umgebung benötigt. Diese Simulation wird gleich wie die Simulationen der IoT-Komponenten in die Co-Simulation eingebunden und dient als Vermittler für den prozessorientierten Datenaustausch, siehe Bild 3.
Zusätzlich wird eine Synchronisierung zwischen den einzelnen Simulationen benötigt. Diese muss zentral erfolgen, da Informationen über den Simulationsfortschritt jeder Teilsimulation benötigt werden. Die Synchronisierung erfolgt über einen Taktgeberagenten, siehe Bild 3,

der Synchronisierungsnachrichten an die einzelnen Co-Simulationsteilnehmer versendet. Um diese Synchronisierungsnachrichten verarbeiten zu können, wird zusätzlich eine Synchronisierungsschnittstelle zwischen den Agenten und den Simulationstools benötigt, über die die Agenten den Zeitfortschritt in den einzelnen Simulationstools beeinflussen können, siehe Bild 2. Diese Synchronisierungsschnittstelle wird ebenfalls, aus den oben genannten Gründen, in einen spezifischen und einen allgemeinen Teil unterteilt.
Zusätzlich wurde ein Pattern-Detection-Algorithmus basierend auf der Levenshtein-Distanz in den Agenten implementiert um das Nachrichtenaufkommen zu reduzieren. Ohne diesen Algorithmus, würde jeder Agent seine vom Simulationstool weiterzuleitenden Nachrichten an alle anderen Agenten weiterleiten, was zu Skalierungsproblemen führen kann. Daher wurde ein Feedbackmechanismus in die Schnittstellen eingebaut, der meldet, ob ein empfangene Nachricht von dem Modell verwertet werden kann oder nicht. Falls eine Nachricht nicht verwendbar war, wird der Inhalt der Nachricht mithilfe der Levenshtein-Distanz analysiert und in einer Liste gespeichert. Falls ähnliche Nachrichten immer nicht verwendbar sind, wird dem Versender dieses Nachrichtentyps mitgeteilt, dass er diese nicht mehr an diesen speziellen Empfänger senden muss.

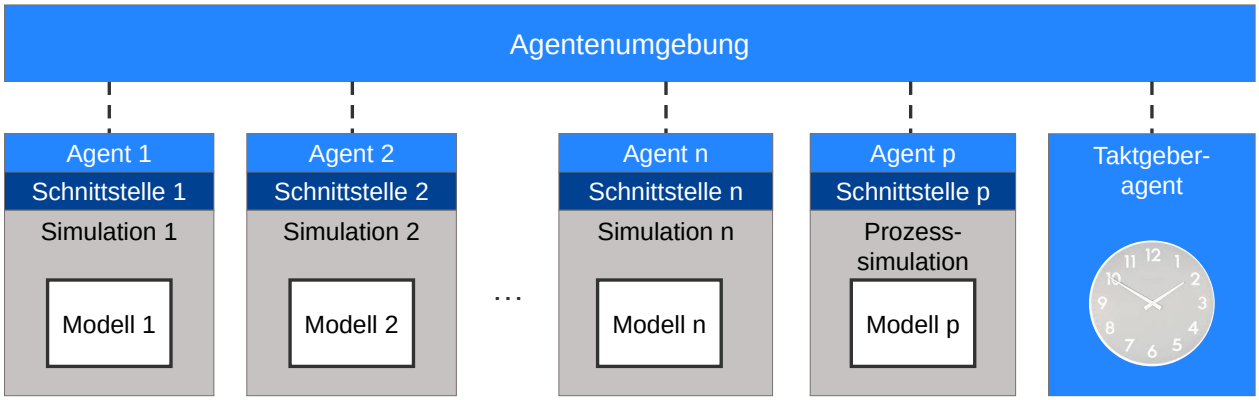

Bild 3: Datenaustausch und Synchronisierung

Somit besteht die Co-Simulation neben den Simulationen der IoT-Komponenten aus einer Simulation der Umgebung. Zusätzlich hierzu existiert ein Taktgeberagent zur Synchronisierung der Co-Simulation. Dieses Konzept wurde prototypisch anhand eines IoT-basierten Warenlagers umgesetzt.

4. Simulation eines Warenlagers

4.1 Szenario

Um das beschriebene Konzept zu evaluieren, wurde ein IoT-basiertes Warenlager simuliert, bestehend aus einem Gabelstapler, einer Heizeinheit, Temperatursensor, Waren,

Lagerregale und Umgebung, welche in verschiedenen Simulationstools simuliert wurden. Falls eine Ware in das Warenlager eintritt, kommuniziert sie mit den Lagerregalen um einen Lagerplatz zu erhalten. Die Lagerregale nehmen Kontakt zu den Temperatursensoren auf, um die momentane Temperatur mit den Temperaturanforderungen der Ware abzustimmen und geben dann dem Gabelstapler Bescheid, in welches Lagerregal die Ware eingelagert werden soll. Zudem kommuniziert die Heizung mit den Temperatursensoren um die Umgebung auf der gewünschten Temperatur zu halten.

4.2 Umsetzung

Die Waren und das Warenlager wurden in OpenModelica modelliert und simuliert, die Temperatursensoren und der Gabelstapler in MATLAB Simulink und die Heizeinheit sowie die Umgebung wurden mit Java modelliert. Das Agentensystem selbst wurde mit BDI-Agenten über Jadex realisiert. Es wurden BDI-Agenten verwendet um die beschriebene Nachrichtenreduktion zu realisieren. Die Modelle können zur Laufzeit durch den Benutzer über eine Benutzeroberfläche eingebunden und entfernt werden. Zuvor müssen die Simulationen allerdings ebenfalls durch den Benutzer im entsprechenden Simulationstool gestartet werden. Wenn der Nutzer ein Modell einfügt, wird ein Agent erzeugt, welcher dann die entsprechende Schnittstelle startet und diese mit der ausgewählten Simulation verbindet. Entfernt der Nutzer die Simulation wieder aus der Co-Simulation, so werden der Agent und die Schnittstelle gelöscht.

Da MATLAB Simulink nur eine Verbindung je Simulationsinstanz zulässt, mussten die spezifischen Schnittstellen zu einer spezifischen Schnittstelle zusammengefasst werden, allerdings blieben die generischen Schnittstellen getrennt. Die Kommunikation der spezifischen Schnittstelle mit der MATLAB-Simulink-Simulation findet über MATLAB-Befehle statt. Hierzu wird der Output-Port von MATLAB regelmäßig auf neue Nachrichten hin abgefragt und ankommende Nachrichten werden über einen Push-Mechanismus an MATLAB weitergegeben. Bei OpenModelica funktioniert die Kommunikation über einen CORBA-Server, der bei jeder Simulation mitgestartet wird. Die Kommunikation mit den Java-Modellen wurde über einen einfachen Methodenaufruf realisiert.

Bisher wurde der Taktgeberagent noch nicht implementiert und somit konnte bisher noch keine Synchronisierung der Simulationen realisiert werden. Daher wurde ein sequentielles Szenario gewählt, in dem keine parallelen Abläufe stattfinden, wodurch eine Synchronisierung für dieses Szenario nicht nötig war.

5. Evaluierung

Bild 4 zeigt einen Screenshot des in Kapitel 4.2 beschriebenen Prototyps. Dieser wurde hinsichtlich der in Kapitel 1 aufgestellten Anforderungen evaluiert und bewertet:

- A1: Es ist möglich, zur Laufzeit Modelle in die Co-Simulation ein- und austreten zu lassen.
- A2: Es wurden MATLAB-, OpenModelica- und Java-Modelle verwendet um die IoT-Komponenten zu modellieren und somit verschiedene Simulationstools.
- A3 und A4: Beide Anforderungen sind erfüllt, da das Konzept weder domänen- noch lebensphasenabhängig ist.
- A5: Es wurde eine Möglichkeit zur Nachrichtenreduktion präsentiert, was eine Form der Intelligenz in den Agenten darstellt.

Somit wurden alle Anforderungen, insbesondere in Bezug auf die Dynamik und Heterogenität von dem vorgestellten Konzept erfüllt.

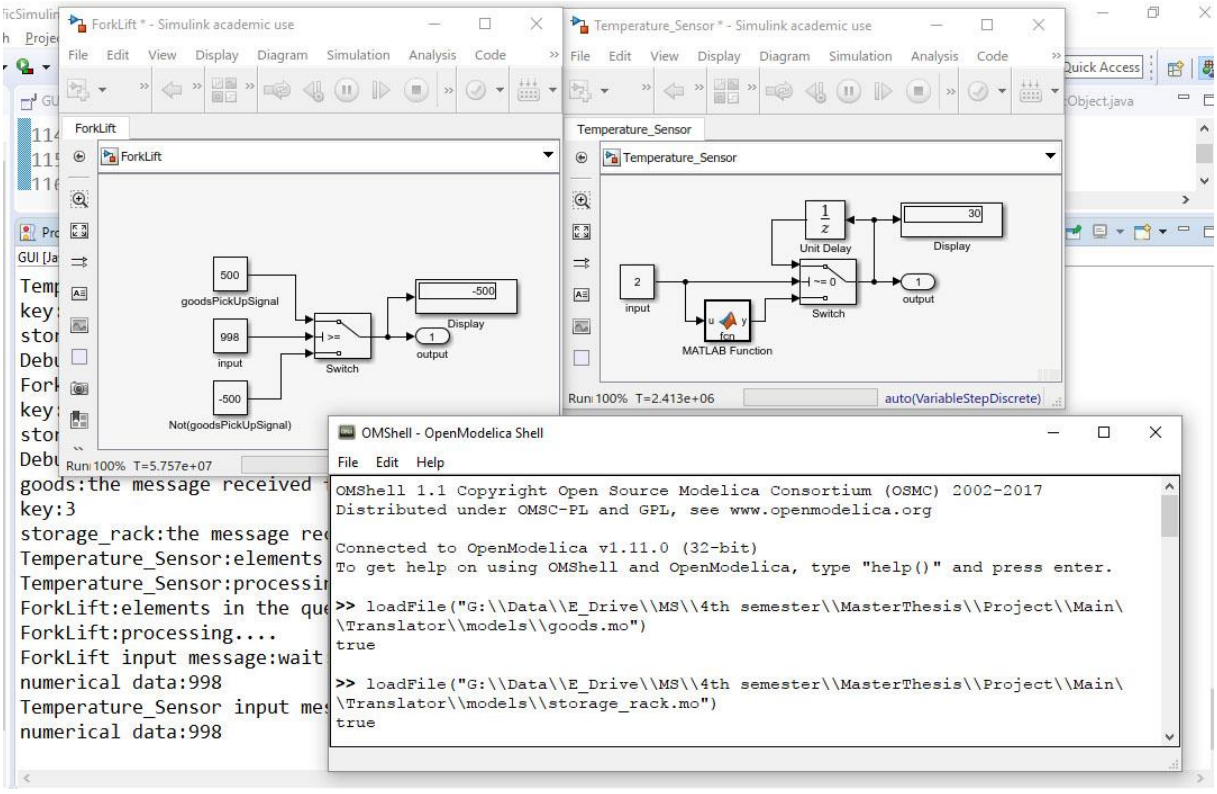

Bild 4: Screenshot des Prototyps mit MATLAB Simulink und OpenModelica

6. Zusammenfassung und Ausblick

Folgende Punkte wurden in diesem Beitrag vorgestellt und diskutiert:

- Es wurden Anforderungen an eine Simulation von IoT-Systemen aufgestellt unter Berücksichtigung der Dynamik und Heterogenität von IoT-Systemen. Aus diesen

Anforderungen wurde gefolgert, dass eine „Plug-and-Simulate“-fähige Co-Simulation benötigt wird.

- Es wurden verschiedene Co-Simulationsansätze vorgestellt und hinsichtlich ihrer Tauglichkeit zur Simulation von IoT-Systemen bewertet. Insbesondere wurde untersucht, ob diese Ansätze „Plug-and-Simulate“-fähig sind.
- Ein neuartiges, agentenbasiertes Konzept zur Co-Simulation von IoT-Systemen wurde vorgestellt. Hierbei wird jede IoT-Komponente in einer eigenen Simulation simuliert, welche gegenüber den anderen Simulationen von einem Agenten vertreten wird.
- Das vorgestellte Konzept wurde prototypisch, anhand eines Warenlagerszenarios realisiert und anschließend anhand der aufgestellten Anforderungen evaluiert.

Im nächsten Schritt wird der vorgestellt Taktgeberagent realisiert und evaluiert, um eine Synchronisierung der Teilsimulationen zu ermöglichen. Hierdurch wird es möglich auch parallele Abläufe in IoT-Systemen zu simulieren.

7. Literatur

[1] Jung, T.; Jazdi, N.; Weyrich, M.: A Survey on Dynamic Simulation of Automation Systems and Components in the Internet of Things. 22nd IEEE ETFA, Limassol, Zypern, 2017.

[2] Mathias Oppelt, Mike Barth, and Leon Urbas. 2015. The Role of Simulation within the Life-Cycle of a Process Plant - Results of a global online survey.

[3] Blockwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., and Viel, A. 2012. Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 173–184.

[4] Bertsch, C., Ahle, E., and Schulmeister, U. 2014. The Functional Mockup Interface - seen from an industrial perspective. In Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 27–33.

[5] Bastian, J., Clauß, C., Wolf, S., and Schneider, P. 2011. Master for Co-Simulation Using FMI. In . Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 115–120.

[6] Fujimoto, R. M. op. 2000. Parallel and distributed simulation systems. Wiley series on parallel and distributed computing. John Wiley & Sons.

[7] 2010. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Framework and Rules. IEEE, Piscataway, NJ, USA.

[8] Oppelt, M., Wolf, G., and Urbas, L. 2014. Capability-analysis of co-simulation approach-es for process industries. In ETFA'2014. 19th IEEE ETFA : September 16-19, 2014 : Barcelona, Spanien. IEEE,

[9] Möller, B. 2012. The HLA Tutorial: A practical guide for developing distributed simulations.

[10] OPC Foundation. 2008. OPC-UA Specification.

[11] Hensel, S., Graube, M., Urbas, L., Heinzerling, T., and Oppelt, M. 2016. Co-simulation with OPC UA. In Proceedings, 2016 IEEE 14th INDIN. Palais des Congrès du Futuroscope, Futuroscope - Poitiers, Frankreich, 19-21 Juli, 2016. IEEE, Piscataway, NJ, 20–25.

[12] The OSGi Alliance. 2014. OSGi Core.

[13] McAffer, J., VanderLei, P., and Archer, S. J. op. 2010. OSGi and Equinox. Creating highly modular Java systems. The eclipse series. Addison-Wesley, Upper Saddle River.

[14] Oppelt, M., Drumm, O., Lutz, B., and Gerrit Wolf Siemens, A. G. 2013. Approach for integrated simulation based on plant engineering data. In ETFA 2013. September 10-13, 2013, Cagliari, Italien. IEEE, Piscataway.

[15] Peshev, D. and Livingston, A. G. 2013. OSN Designer, a tool for predicting organic solvent nanofiltration technology performance using Aspen One, MATLAB and CAPE OPEN. Chemical Engineering Science.

[16] Hopkinson, K., Wang, X., Giovanini, R., Thorp, J., Birman, K., and Coury, D. 2006. EPOCHS. A Platform for Agent-Based Electric Power and Communication Simulation Built From Commercial Off-the-Shelf Components. IEEE Trans. Power Syst. 21, 2, 548–558.

[17] Schutte, S., Scherfke, S., and Troschel, M. 2011. Mosaik: A framework for modular simulation of active components in Smart Grids. In SGMS 2011. 2011 IEEE First International Workshop on Smart Grid Modeling and Simulation: [17 Okt. 2011, Brüssel, Belgien]. IEEE.

[18] Nutaro, J., Kuruganti, P. T., Miller, L., Mullen, S., and Shankar, M. 2007. Integrated Hybrid-Simulation of Electric Power and Communications Systems. In 2007 IEEE Power Engineering Society General Meeting, Tampa, FL 24-28 Juni. IEEE Xplore, Piscataway, N.J.

[19] Weiss, G. 2001, ©1999. Multiagent systems. A modern approach to distributed artificial intelligence. MIT Press, Cambridge.