Diagnose von Inkonsistenzen in heterogenen Engineeringdaten

Stefan Feldmann und Birgit Vogel-Heuser

Zusammenfassung

Industrie 4.0 bedeutet mehr Komplexitat – nicht zuletzt auch während des Engineerings automatisierter Produktionssysteme. Essenziell für den Erfolg von Industrie 4.0-Entwicklungsprojekten ist, dass Fehler während der Entwicklung frühzeitig erkannt und behoben werden. Solche Fehler manifestieren sich in vielen Fällen durch Inkonsistenzen in den Engineeringdaten, die oftmals sehr heterogener Natur sind. Zur Adressierung dieser Problematik analysiert dieses Kapitel die Herausforderung des Managements (d. h. der Erkennung und Behebung) von Inkonsistenzen in heterogenen Engineeringdaten und stellt einen Ansatz zur Diagnose von Inkonsistenzen vor.

1 Einleitung

Die Umsetzung von Industrie 4.0 birgt ein erhöhtes Maß an Komplexität während des Engineerings und Betriebs automatisierter Produktionsanlagen. Insbesondere der interdisziplinäre Charakter von Industrie 4.0-Entwicklungsprojekten in der industriellen Produktionsautomatisierung führt zu der Beteiligung einer Vielzahl von Akteuren – beispielsweise aus den unterschiedlichen Ingenieursdisziplinen Mechanik, Elektrotechnik/Elektronik und Software, aber auch aus angrenzenden Disziplinen wie dem Projektmanagement. Diese Vielzahl an beteiligten Akteuren verwenden dabei heterogene Modellierungsansätze, Formalismen und Softwarewerkzeuge (Broy et al. 2010; Gausemeier et al. 2009) sowie unterschiedliche Abstraktionsgrade (Hehenberger et al. 2007; Rieke et al. 2012). Während der Entwicklung von automatisierten Produktionssystemen entsteht somit eine heterogene Modelllandschaft,

S. Feldmann (✉) • B. Vogel-Heuser
Lehrstuhl für Automatisierung und Informationssysteme, Technische Universität München, Garching, Deutschland
E-Mail: feldmann@ais.mw.tum.de; vogel-heuser@ais.mw.tum.de

B. Vogel-Heuser et al. (Hrsg.), *Handbuch Industrie 4.0 Bd.2*, Springer Reference Technik, DOI 10.1007/978-3-662-53248-5_91

die insbesondere von überlappenden, teilweise redundant modellierten Informationen geprägt ist. Diese Überlappungen und Redundanzen können, vor allem wenn die Modelle an geänderte System- oder Produktanforderungen angepasst werden müssen, zu Inkonsistenzen führen, die – wenn sie zu spät erkannt werden – fehlerhafte Entscheidungen hervorrufen können. Ein prominentes Beispiel für solche fehlerhaften Entscheidungen ist die Mission des Mars Climate Orbiters, die aufgrund eines Navigationsfehlers infolge einer Inkonsistenz in den verwendeten physikalischen Einheiten fehlschlug (NASA 2000). Die frühzeitige Erkennung und Behebung von Inkonsistenzen ist somit essenziell für das Engineering von automatisierten Produktionssystemen und wird darüber hinaus für Anwendungen im Kontext von Industrie 4.0, die insbesondere die Komplexität der Engineeringdaten erhöhen, zunehmend an Wichtigkeit gewinnen.

Inkonsistenzen im Engineering automatisierter Produktionsanlagen können dabei vielfältig sein. Zum einen können statische Inkonsistenzen in Strukturmodellen auftreten, sowie dynamische Inkonsistenzen in Verhaltensmodellen. Statische Inkonsistenzen können dabei einerseits Fehler mit gravierenden Auswirkungen (beispielsweise infolge der Inkompatibilitat von Einheiten oder der Über- bzw. Unterspezifikation von Parametern) implizieren, andererseits dagegen weniger gravierende Auswirkungen bewirken (beispielsweise bei Verletzungen von Namenskonventionen). Während die Erkennung von statischen Inkonsistenzen ohne Ausführung (dynamischer) Modelle, wie beispielsweise Simulationsmodelle, möglich ist, erfordert das Management von dynamischen Inkonsistenzen die Ausführung selbiger. Der Fokus dieses Kapitels liegt insbesondere auf der Erkennung statischer Inkonsistenzen, die oftmals infolge der Abhängigkeit zwischen verschiedenen, heterogenen Modellen auftreten.

Viele verfügbare Softwarewerkzeuge ermöglichen bereits die Überprüfung der Konformität modellierter Informationen gegenüber syntaktischen oder Wohlgeformtheitsregeln – eine ganzheitliche, disziplinübergreifende Definition und Überprüfung von Inkonsistenzregeln wird bisher aufgrund der Heterogenität der verschiedenen, disziplinspezifischen Informationen jedoch kaum unterstützt. Für das Engineering automatisierter Produktionssysteme werden potenzielle Entwicklungsfehler meist während Verifikations- und Validierungsphasen identifiziert. Da die Verifikation und Validierung jedoch oftmals erst in sehr späten Phasen des Entwicklungsprozesses stattfinden (Isermann 2008), kann die Erkennung und Behebung von Inkonsistenzen kostspielig sein. Ein Ansatz, der die kontinuierliche, (teil-)automatische Erkennung und Behebung von Inkonsistenzen ermöglicht, würde folglich signifikant bei der Entwicklung automatisierter Produktionssysteme unterstützen, indem das Auftreten von Inkonsistenzen kontinuierlich (teil-)automatisch überwacht und behoben wird.

Ein erster Schritt zur Adressierung der zuvor beschriebenen Problematik ist die effiziente Diagnose von Inkonsistenzen in heterogenen Engineeringdaten. Die Diagnose umfasst dabei nicht nur die *Lokalisierung* von Inkonsistenzen, sondern auch die *Identifikation* der Gründe für das Auftreten der Inkonsistenz sowie die *Klassifikation* derselben (Nuseibeh et al. 2000). Im folgenden Abschnitt wird zunächst ein Anwendungsbeispiel aufgezeigt, das die Herausforderungen des

Inkonsistenzmanagements illustriert. Darauffolgend werden im Abschnitt 3 die Anforderungen an das Management (d. h. die Erkennung und Behebung) von Inkonsistenzen in heterogenen Modelllandschaften analysiert und in Abschnitt 4 dem Stand der Forschung und Technik gegenübergestellt. Das Konzept zur Diagnose von Inkonsistenzen in heterogenen Engineeringdaten wird anhand eines Beispielszenarios in Abschnitt 5 erläutert. Abschnitt 6 diskutiert die gewonnenen Erkenntnisse und Limitationen des Ansatzes und gibt einen Ausblick auf weitere, notwendige Forschungsbedarfe. Der Beitrag greift somit bestehende Forschungsergebnisse der Autoren auf, einerseits zum konzeptionellen Framework zum Inkonsistenzmanagement (vgl. Feldmann et al. 2015a) und andererseits zum Vergleich verschiedener Ansätze zum Inkonsistenzmanagement (vgl. Feldmann et al. 2015b), und leitet Forschungsgegenstände ab, die in zukünftigen Arbeiten adressiert werden müssen.

2 Anwendungsbeispiel

Zur Illustration der Herausforderungen, die sich infolge des Inkonsistenzmanagements ergeben, wird im Folgenden ein Anwendungsbeispiel anhand eines Demonstrators der Fertigungstechnik aufgezeigt – der Pick and Place-Unit (PPU (Vogel-Heuser et al. 2014a), Abb. 1). Obwohl die PPU ein akademischer Labordemonstrator ist, bildet sie einen Teil der Herausforderungen der modellbasierten Entwicklung mechatronischer Systeme ab. Die PPU besteht aus vier Modulen: Dem *Stapel,* dem *Kran,* dem *Stempel* und der *Rampe.* Der Stapel repräsentiert die Werkstückquelle. In einem ersten Schritt wird ein Werkstück aus dem Stapel in eine Übergabeposition übergeben. Der Kran greift das Werkstück und transportiert dieses zum Stempel, in welchem das

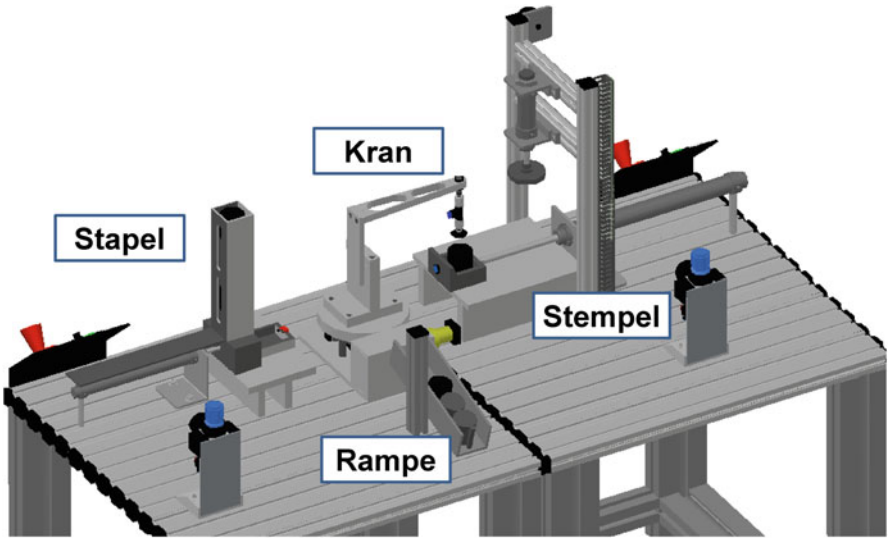

Abb. 1 Überblick über die Pick and Place-Unit (Vogel-Heuser et al. 2014a)

Werkstück eingespannt und gestempelt wird. Nachdem der Stempelprozess abgeschlossen ist, wird das Werkstück vom Kran zur Rampe transportiert.

Die Dokumentationen der PPU bilden dabei nicht nur Aspekte des „klassischen" Engineerings automatisierter Produktionssysteme ab, sondern auch weitere Aspekte, die sich beispielsweise im Rahmen von Industrie 4.0 ergeben. So existieren neben den „klassischen" Evolutionsszenarien, die sich infolge einer sequentiellen (Weiter-) Entwicklung mechatronischer Systeme ergeben (Vogel-Heuser et al. 2014a), auch parallele Evolutionsszenarien, beispielsweise zum Einbezug von Ferndiagnose-Funktionalitäten oder aber zur Realisierung von Selbstadaptivität bis hin zur Selbstheilung des Systems (Vogel-Heuser et al. 2014b). Die PPU ist somit ausreichend geeignet, um die Herausforderungen des Inkonsistenzmanagements in heterogenen Engineeringdaten im Kontext von Industrie 4.0 abzubilden.

3 Anforderungen für das Inkonsistenzmanagement

Der Begriff der *Konsistenz* wird in der Literatur vorwiegend durch sein Antonym *Inkonsistenz* definiert: Ein Modell (oder eine Menge von Modellen) wird als inkonsistent bezeichnet, wenn zwei Aussagen im Modell (oder in einer Menge von Modellen) nicht gleichzeitig erfüllbar sind (Spanoudakis und Zisman 2001), d. h. wenn sie widersprüchlich sind. Enthält das Modell keine bekannten Widersprüche, so wird es folglich als *konsistent* bezeichnet (Herzig et al. 2014). Inkonsistenzen treten beispielsweise aufgrund parallel entwickelter Modelle, schlecht verstandener Abhängigkeiten, ungenauer Anforderungen oder fehlender Informationen auf (Mens et al. 2006). Die Diagnose und Auflösung (d. h. das Management) von Inkonsistenzen ist somit notwendig, wenn Modelle in kollaborativen Projekten erstellt und gepflegt werden (Hehenberger et al. 2007). Besonders schwerwiegend sind Inkonsistenzen, deren Auftreten zu fehlerhaften Entscheidungen im Engineering führen. Insbesondere die starken Abhängigkeiten zwischen den einzelnen Disziplinen, die am Engineering automatisierter Produktionssysteme beteiligt sind, sind dabei kritisch: Die Vielzahl an beteiligten Akteuren, deren verschiedene Betrachtungsgegenstände und Aufgaben, die im Engineeringprozess adressiert werden und damit eng miteinander verbunden sind, führen zu einer Vielzahl von potenziellen Inkonsistenzen in der heterogenen Modelllandschaft. Dies führt zu Anforderungen, die von einem Ansatz zum Management von Inkonsistenzen adressiert werden müssen.

Anforderung A1 – Heterogenität der Modelle
Um die verschiedenen Betrachtungsgegenstände der an der Entwicklung automatisierter Produktionssysteme beteiligten Akteure geeignet zu adressieren, kommt eine Vielzahl an Modellierungsansätzen, Formalismen und Softwarewerkzeugen zum Einsatz – für das Beispiel der PPU beispielsweise CAD-Systeme in der mechatronischen Entwicklung, Stromlaufpläne in der elektronischen/elektrischen Entwicklung und Steuerungssoftware in der Softwareentwicklung. Dies führt zu unterschiedlichen, heterogenen Modellen, die verschiedene Formalismen und Abstraktionsgrade verwenden, um die disziplinspezifischen Aspekte des betrachteten Systems

abzubilden. Ein Anforderungsmodell, das die von der PPU zu realisierenden Funktionalitäten in den frühen Phasen des Engineerings spezifiziert, beinhaltet beispielsweise noch wenig konkrete Informationen, während ein Softwaremodell, welche das zu realisierende Steuerungsverhalten der PPU beschreibt, logische Verknüpfungen der Ein- und Ausgänge der PPU-Steuerung beschreibt. Diese Heterogenität der Modelle stellt eine zentrale Herausforderung dar (Broy et al. 2010; Gausemeier et al. 2009), da die Zusammenführung derselben schwierig ist. Folglich ist die Übersetzung der vielfältigen, heterogenen Modelle in einen gemeinsamen Repräsentationsformalismus unerlässlich, um auf dieser Basis Inkonsistenzen diagnostizieren und auflösen zu können.

Anforderung A2 – Semantische Abhängigkeiten
Die während des Engineering erstellten oder genutzten Modelle beschreiben oftmals sich überlappende Informationen. Infolgedessen entstehen semantische Abhängigkeiten zwischen den Modellen (Spanoudakis und Zisman 2001), die sich durch das Vorhandensein von redundant modellierten oder voneinander abhängigen Informationen (z. B. durch verschiedene Abstraktionsgrade) abzeichnen. Im Anwendungsbeispiel der PPU werden beispielsweise Eingänge und Ausgänge im Stromlaufplan definiert, die wiederum ihre Korrespondenz in den Ein- bzw. Ausgangsvariablen der Steuerungssoftware finden. Die Definition und die Identifikation solcher semantischen Abhängigkeiten zwischen den Modellen sind jedoch zeit- und kostenintensiv (Spanoudakis und Zisman 2001). Ein Mechanismus zur Identifikation und Definition der Abhängigkeiten zwischen konkreten Modellinstanzen ist somit essenziell für die Identifikation von Inkonsistenzen. Ein Ansatz zum Inkonsistenzmanagement in heterogenen Engineeringdaten und -modellen muss solche semantischen Abhängigkeiten somit definieren und identifizieren können.

Anforderung A3 – Diagnose von Inkonsistenzen
Um einen Ansatz zum Management von Inkonsistenzen in heterogenen Modellen – die oftmals tausende Entitäten umfassen – zu realisieren, ist ein (teil-)automatischer Ansatz erforderlich (Gausemeier et al. 2009). Dem von Nuseibeh et al. (2000) vorgeschlagenen Framework folgend beinhaltet die Diagnose von Inkonsistenzen drei zentrale Bestandteile: Das *Lokalisieren* der Inkonsistenz (Anforderung A3.1) anhand der inkonsistenten Modellelemente, die *Identifikation* der Gründe für das Auftreten der Inkonsistenz (Anforderung A3.2) und die *Klassifikation* der Inkonsistenz (Anforderung A3.3) um abzuschätzen, wie wichtig die sofortige Auflösung einer Inkonsistenz ist. Während die Verletzung einer Namenskonvention in den Softwarevariablen der Steuerungssoftware im Anwendungsbeispiel der PPU keine sofortige Auflösung der Inkonsistenz erfordert, können Inkonsistenzen zwischen den Anforderungen an die PPU und der Spezifikation des Softwareverhaltens durchaus gravierende Fehler hervorrufen.

Diese Diagnose von Inkonsistenzen ist dabei jedoch nicht trivial, da die Notwendigkeit zur Behandlung der Inkonsistenz darüber hinaus abhängig von weiteren Faktoren ist. Beispielsweise sind manche Inkonsistenzen in einigen Phasen des Engineeringprozesses völlig in Ordnung (beispielsweise wenn Entscheidungen in frühen Phasen des Engineerings noch verglichen werden müssen), in anderen

Phasen können sie jedoch grobe Fehler implizieren (beispielsweise wenn das mechatronische Systeme in Betrieb genommen werden soll). Das temporäre Zulassen von Inkonsistenzen sollte somit bei der Diagnose auch berücksichtigt werden können.

Anforderung A4 – Auflösung von Inkonsistenzen
Um Akteuren während des Engineerings automatisierter Produktionssysteme Handlungsalternativen zu bieten, die infolge einer Inkonsistenz gewählt werden können, ist die (teil-)automatische Auflösung von Inkonsistenzen erforderlich. Dies erfordert Auflösungsvorschläge (Anforderung A4.1), um den Akteuren eine Entscheidungsgrundlage zu bieten. Falls mehrere Alternativen existieren, ist die Bewertung der Auflösungsvorschlage (Anforderung A4.2) unerlässlich. Insbesondere muss dabei die Zusammenarbeit der verschiedenen, am Engineering beteiligten Interessensgruppen gesichert werden. So ist die Inkonsistenz zwischen den spezifizierten Ein- und Ausgängen der PPU in der Elektrotechnik und den in der Software verwendeten Variablen nur durch geeignete Zusammenarbeit zwischen Software- und Elektroingenieuren sicher aufzulösen.

Praktikabel ist die Auflösung von Inkonsistenzen dabei jedoch nur, wenn zuvor abgeschätzt werden kann, welche Auswirkungen die Inkonsistenz hat (A3) und welche Ressourcen zur Auflösung der Inkonsistenz zur Verfügung stehen. Daher muss insbesondere aus praktischer Sicht auch berücksichtigt werden können, welche Kosten bzw. Risiken der Behebung bzw. Vernachlässigung der Inkonsistenz gegenüberstehen.

Anforderung A5 – Visualisierung
Die reine Auflistung von Inkonsistenzen und der möglichen Auflösungsalternativen ist aufgrund des hohen Komplexitätsgrads der beteiligten Modelle in vielen Fällen nicht ausreichend. Erschwerend kommt hinzu, dass die Identifikation von semantischen Abhängigkeiten zwischen den verschiedenen Modellen (vgl. Anforderung A2) oftmals zeit- und kostenintensiv ist und detailliertes Wissen über die domänenspezifischen Modellierungselemente erfordert. Um die Akteure bei der Entscheidung, wie wichtig die unmittelbare Auflösung einer Inkonsistenz ist, zu unterstützen und die Identifikation von semantischen Abhängigkeiten zu vereinfachen (Basole et al. 2015), sind geeignete Techniken zur Visualisierung und Interaktion mit der Visualisierung erforderlich. Einige Inkonsistenzen sind zudem in vielen Fällen nicht ausschließlich auf logische Widersprüche in den Engineeringdaten und -modellen zurückzuführen: Die Anforderung liegt dann darin, ein gemeinsames Verständnis der während des Engineerings mechatronischer Systeme beteiligten Akteure zu schaffen – d. h. die Inkonsistenz im Verständnis der Akteure zu identifizieren. Die Entwicklung dieser Methoden erfordert jedoch das detaillierte Verständnis der mentalen Modelle der beteiligten Akteure.

4 Stand der Technik

In diesem Abschnitt werden die aktuellen Forschungsarbeiten im Bereich des Inkonsistenzmanagements heterogener Modelllandschaften im Engineering automatisierter Produktionssysteme vorgestellt. Hierbei werden zunächst Ansätze zur

Tab. 1 Vergleich der Ansätze zum Inkonsistenzmanagement (erweitert aus Feldmann et al. (2015b))

Forschungsgruppe / Anforderung		Anwendungsbereich	Heterogenität der Modelle (A1)	Semantische Abhängigkeiten (A2)	Diagnose (A3)			Auflösung (A4)		Visualisierung (A5)
					Lokalisation (A3.1)	Identifikation (A3.2)	Klassifikation (A3.3)	Einzeln (A4.1)	Mehrere (A4.2)	
modellbasiert	Thramboulidis	Mechatronik	+	○	–	–	–	–	–	–
	Biffl et al.	Automatisierung	+	+	○	○	○	–	–	+
	Bassi et al.	Automatisierung	+	–	○	–	–	–	–	–
beweistheoriebasiert	Finkelstein et al	Software	○	○	+	–	–	+	○	–
	Schätz et al.	Software	+	○	+	–	+	–	–	–
	Van Der Straeten et al.	Software	○	○	+	○	○	+	○	–
regelbasiert	Egyed et al.	Software	–	–	+	+	+	○	○	○
	Hegedüs et al.	System	+	○	+	○	+	+	+	–
	Herzig et al.	System	+	+	+	○	○	–	–	–
	Feldmann et al.	Mechatronik	+	○	+	+	○	–	–	+
	Vyatkin et al.	Automatisierung	–	○	○	○	○	–	–	○
synchronisationsbasiert	Vyatkin et al.	Automatisierung	–	○	○	○	○	–	–	○
	Giese et al.	Software	○	○	+	○	○	+	–	–
	Gausemeier et al.	Mechatronik	+	○	+	○	○	+	–	–
	Bill et al.	System	+	+	○	○	○	–	–	–
	Biffl et al.	System	+	+	+	○	○	–	–	+

modellbasierten Entwicklung diskutiert (Abschn. 4.1). Anschließend werden Ansätze betrachtet, welche die Diagnose und Auflösung von Inkonsistenzen fokussieren (Abschn. 4.2). Tab. 1 gibt einen Überblick über die verschiedenen Ansätze. Die wesentlichen Erkenntnisse dieses Abschnittes wurden aus (Feldmann et al. 2015b) zusammengefasst und erweitert.

4.1 Modellbasierte Entwicklung von automatisierten Produktionssystemen

Modelle abstrahieren die Realität auf die für den Modellierungszweck wesentlichen Bestandteile eines Betrachtungsgegenstands. Im Softwaredesign hat sich insbesondere die Unified Modeling Language (UML) (OMG 2011) etabliert (Secchi et al. 2007). Die interdisziplinäre Entwicklung von automatisierten Produktionssystemen involviert jedoch eine Vielzahl von Disziplinen und Akteuren (vgl. A1), deren Blickwinkel durch geeignete Modellierungskonzepte unterstützt werden müssen. Neben der Erweiterung der UML mittels Profilen, beispielsweise zur Modellierung des zustandsbasierten Verhaltens mechatronischer Systeme (MechatronicUML, Schäfer und Wehrheim 2010), etablierte sich die modellbasierte Systementwicklung (Model-Based Systems Engineering, MBSE). Aus der UML hat sich unter anderem die Systems Modeling Language (SysML) (OMG 2012) als eine weit verbreitete, grafische Modellierungssprache durchgesetzt.

Um die verschiedenen, am Engineering beteiligten Disziplinen mit ihren eigenen Terminologien und Formalismen berücksichtigen zu können, existieren in der Literatur zwei wesentliche Ansätze: (1) die Verwendung integrierter Ansätze zur

Zusammenführung der verschiedenen, disziplinspezifischen Sichten in einem Modell und (2) die Verwendung von Ansätzen, welche die Beschreibung von Abhängigkeiten zwischen den verschiedenen Modellen ermöglicht. Integrierte Ansätze finden sich beispielsweise in der Mechatronik (z. B. ein 3 + 1-Sichten-Modell mechatronischer Systeme (Thramboulidis 2013)). Zudem schlagen Biffl et al. (2015) die Anwendung von AutomationML (vgl. Deutsches Institut für Normung e.V. 2015; Drath 2010) – ein für die Interessen der Automatisierungstechnik angepasstes und standardisiertes Datenaustauschformat – vor, um die semantischen Abhängigkeiten zwischen Engineering-Modellen zu erstellen und zu verwalten. Die Abbildung aller am Engineering beteiligten Formalismen und Sprachen in eine einzelne, domänenübergreifende Modellierungssprache würde jedoch zu einem hochkomplexen Modellierungsansatz führen, der weit über die domänenspezifischen Interessen und Blickwinkel der einzelnen Akteure hinausgeht. Weitere Ansätze streben eine Abbildung zwischen den verschiedenen Sichten an (z. B. mittels eines übergeordneten SysML-Modells in der Entwicklung mechatronischer Systeme (Bassi et al. 2011)). Beide Ansätze ermöglichen es zwar, mittels semi-formaler Modellierungssprachen die verschiedenen Aspekte des mechatronischen Systems grafisch zu modellieren (vgl. A1), das Management von Inkonsistenzen (vgl. A3 und A4) steht bei diesen Ansätzen aber nicht im Fokus.

Zwar ermöglichen einige der verfügbaren Softwarewerkzeuge zur modellbasierten Systementwicklung die Überprüfung der Konformität von Modellen zu vordefinierten syntaktischen und Wohlgeformtheits-Regeln, die Identifikation und Definition von semantischen Abhängigkeiten zwischen den Modellen (vgl. A2) und das Management von Inkonsistenzen (vgl. A3 und A4) wird jedoch nicht a priori unterstützt. Daraus folgend werden fehlerhafte Entscheidungen bei der Entwicklung von automatisierten Produktionssystemen meist erst in oftmals sehr spät durchgeführten Verifikations- und Validierungsprozessen erkannt und können dadurch zu kostspieligen Änderungen führen. Um diese fehlerhaften Entscheidungen möglichst frühzeitig aufdecken und damitpotenzielle Änderungskosten minimieren zu können, ist eine (teil-)automatische und kontinuierliche Strategie für das Management von Inkonsistenzen unerlässlich. Dies ermöglicht die bessere Handhabung der Innovationszyklen und – damit verbunden – des Engineerings mechatronischer Systeme, indem Iterationen im Entwicklungsprozess und somit Mehraufwand vermieden bzw. reduziert werden. Die Kombination einer nutzerorientierten Modellierungssprache mit rechnerorientierten Verarbeitungsmechanismen ist daher unerlässlich (Brix und Reeßing 2009).

4.2 Management von Inkonsistenzen in Modellen des mechatronischen Systems

In der verwandten Literatur im Bereich des Inkonsistenzmanagements existiert eine Vielzahl von Ansätzen, die sich mit den Herausforderungen des Managements von Inkonsistenzen beschäftigen. Eine allgemeine Unterteilung dieser Ansätze kann in *beweistheoriebasierte* und *regelbasierte* Ansätze sowie *synchronisationsbasierte*

Ansätze erfolgen. Im Folgenden werden diese Ansätze unter Berücksichtigung der Anforderungen A1 bis A5 untersucht (Zusammenfassung siehe Tab. 1).

Beweistheoriebasierte Ansätze
Beweistheoriebasierte Ansätze erfordern die Notwendigkeit eines vollständigen und konsistenten formalen Systems, das der Modelllandschaft unterliegt. Ein Modell ist in diesem Falle konsistent, wenn es konsistent zum unterliegenden formalen System ist. Beispiele für solche beweistheoriebasierten Ansätze existieren insbesondere in der Domäne der Softwareentwicklung. Finkelstein et al. (1994) transformieren Softwaremodelle (Klassendiagramme, Sequenzdiagramme, etc.) in eine Prädikatenlogik erster Stufe. Semantische Abhängigkeiten werden manuell definiert (A2); Inkonsistenzen können automatisch geschlussfolgert und mittels spezifischer Regeln aufgelöst werden (A3 und A4). Ein ähnlicher Ansatz von Schätz et al. (2003) nutzt Aussagenlogik. Die Grenzen dieser Ansätze liegen jedoch in der beschränkten Ausdrucksmächtigkeit von Prädikaten- und Aussagenlogik. Zudem erfordern diese Ansätze, Modellierungssprachen in logische, formale Formalismen zu übersetzen. Van Der Straeten et al. (Van Der Straeten und D'Hondt 2006) nutzen Beschreibungslogik, um Inkonsistenzen in UML Klassen-, Zustands- und Sequenzdiagrammen zu identifizieren und aufzulösen. Unter anderem können somit fehlerhafte Relationen (z. B. Instanzen ohne Klassenzuordnung, Referenzen ohne Verweise) sowie inkompatible Verhaltensbeschreibungen identifiziert werden (Anforderungen A3 und A4). Basierend auf den Inkonsistenzen und Auflösungsvorschlagen werden Vorschläge zur Restrukturierung der Modelle generiert (Van Der Straeten und D'Hondt 2006).

Der Vorteil solcher beweistheoriebasierten Ansätze liegt darin, dass die Schlussfolgerungen, die auf Basis des unterliegenden formalen Systems gezogen werden können, logisch korrekt sind (d. h. die Konsistenz der Modelle zum unterliegenden formalen System kann formal nachgewiesen werden). Diesem Vorteil steht jedoch der Mehraufwand gegenüber, der durch die Notwendigkeit entsteht, die beteiligten Modelle (darunter auch Modelle semi-formaler oder sogar informaler Natur) in ein formales System zu überführen.

Regelbasierte Ansätze
Im Gegensatz zu beweistheoriebasierten Ansätzen, welche die Vordefinition eines vollständigen und konsistenten formalen Systems erfordern, gehen regelbasierte Ansätze von einer unvollständigen Beschreibung aus (Nuseibeh et al. 2000). Konsistenzregeln beschreiben entweder (1) die hinreichenden Bedingungen, die ein konsistentes Modell erfüllen muss (Hehenberger et al. 2007) (engl. *Positive Constraint)* oder (2) die hinreichenden Bedingungen für das Auftreten einer Inkonsistenz (Herzig et al. 2014) (engl. *Negative Constraint).* Der Ansatz von Egyed (2011) adressiert zwei wesentliche Aspekte: Zum Ersten werden inkrementelle Konsistenzprüfungen durchgeführt (d. h. nur geänderte Modellbestandteile werden geprüft). Zum Zweiten ermöglicht der Model/Analyzer-Ansatz (Egyed 2011) die Verwendung beliebiger Regelsprachen zur Formulierung der Konsistenzregeln. Die Typen von Inkonsistenzen, die identifiziert werden können (A3), hängen von der verwendeten Regelsprache (bzw. von dem für die Sprache realisierten Verarbeitungsmechanismus)

ab. Zudem fokussiert der von Egyed (2011) vorgeschlagene Ansatz insbesondere das Inkonsistenzmanagement in Software-Modellen; die weiteren, im Engineering beteiligten Modelle werden nicht betrachtet. Hehenberger et al. (2007) erweitern diesen Ansatz für Modelle mechatronischer Systeme, indem die Modellelemente mittels einer domänenübergreifenden Ontologie semantisch annotiert werden (A2). Die (manuelle) Annotation der Modellelemente führt jedoch zu (manuellem) Spezifikationsaufwand, der die Notwendigkeit einer automatischen Unterstützung hervorruft (A2). Weitere Ansätze verfolgen die Repräsentation von Inkonsistenzen als Muster in Graphen. Herzig et al. (2014) und Kovalenko et al. (2014) streben die Diagnose von Inkonsistenzen an. Die Identifikation von „Quick Fix"-Auflösungen (d. h. Schnellvorschläge zur Behebung der Inkonsistenz) wird von Hegedüs et al. (2011) angestrebt. Feldmann et al. (2014) formulieren Kriterien für die Inkompatibilität zwischen mechatronischen Modulen mittels Muster in graphbasierten Modellen. Aufgrund der Flexibilität und Anwendbarkeit für eine Vielzahl von Inkonsistenztypen ist der Ansatz, Inkonsistenzen als Muster in graphbasierten Modellen zu repräsentieren, vielversprechend. Seine Anwendbarkeit für die am Engineering beteiligten Modelle ist zu prüfen.

Obwohl regelbasierte Ansätze weniger formal sind als beweistheoriebasierte Ansätze, ermöglichen regelbasierte Ansätze die Realisierung einer höheren Flexibilität im Inkonsistenzmanagement: Regeln können ohne Wissen über ein zugrunde liegendes formales System hinzugefügt und angepasst werden – folglich können allerdings die Ergebnisse der Diagnose von Inkonsistenzen nicht als logisch korrekt bezeichnet werden (d. h. die Konsistenz der Modelle kann nicht formal nachgewiesen werden). Nichtsdestotrotz ermöglicht die hohe Flexibilität solcher regelbasierten Ansätze die vereinfachte Berücksichtigung weiterer Modelltypen und Inkonsistenztypen. Zusammenfassend wird deutlich, dass ein regelbasierter Ansatz für heterogene Modelllandschaften, wie sie im Engineering automatisierter Produktionssysteme vorliegen, erstrebenswert ist.

Synchronisationsbasierte Ansätze

Ein Forschungsbereich, der sich ähnlich zum regelbasierten Inkonsistenzmanagement durchgesetzt hat, ist das Management von Änderungen in Modellen (Hamraz et al. 2013) – und damit verbunden das Inkonsistenzmanagement durch die Transformation von geänderten Modellelementen. Lin et al. (2015) verfolgen das Änderungsmanagement in SysML-Modellen. Heterogene Modelllandschaften (A1), die verschiedenste Modellierungsformalismen involvieren, werden jedoch nicht unterstützt. Die (teil-)automatische Ableitung von Auflösungsalternativen, wenn Inkonsistenzen aufgedeckt werden (A4), steht nicht im Fokus. Giese und Wagner (2009) verwenden einen regelbasierten Transformationsansatz, der die bidirektionale Synchronisation verschiedener Softwaremodelle mittels sogenannter Triple Graph Grammars (TGGs) ermoglicht, welche bidirektionale Transformationsregeln zur Definition der Abhängigkeiten zwischen Modellelementen definieren (A2). Statt der vollständigen Transformation eines Gesamtmodells werden nur am Modell durchgeführte Änderungen transformiert. Insbesondere bei der Beteiligung einer Vielzahl von Modelltypen (wie es im Engineering automatisierter Produktionssysteme

der Fall ist), gestaltet sich dieser Ansatz als schwierig, da viele Abhängigkeiten identifiziert und definiert werden müssen (A1). Aus diesem Grunde synchronisieren Gausemeier et al. (2009) Modelle verschiedener Domänen im mechatronischen Entwurf mit einer domänenübergreifenden Systemspezifikation. Die Erweiterung dieses Ansatzes für Verhaltensmodelle mechatronischer Systeme am Beispiel von Modellen der MechatronicUML (Schäfer und Wehrheim 2010) und MATLAB/Simulink-Modellen wurde von Rieke et al. (2012) vorgestellt. Semantische Abhängigkeiten (beispielsweise die Verfeinerung von Modellelementen) können in diesem Ansatz definiert (A2) und verschiedene Auflösungsvorschlage generiert werden (A4). Die Abbildung zwischen TGGs und Technologien des Semantic Web wurde von Bill et al. (2014) vorgestellt. Ähnlich zu TGGs nutzen Moser und Biffl (2012) Linkmodelle zwischen den Datenstrukturen von Engineering-Systemen. Somit kann – zusätzlich zur Synchronisation der verschiedenen Modelle – ein Monitoring-Konzept für Inkonsistenzen (Biffl et al. 2014) erreicht werden, um Inkonsistenzregeln kontinuierlich zu validieren.

Obwohl TGGs ausreichend ausdrucksstark sind, um eine Vielzahl von Transformationsregeln zu beschreiben, ist ihre Anzahl und Komplexität stark verbunden mit der Anzahl und Komplexität der semantischen Abhängigkeiten zwischen den Modellen (A2) und der Anzahl der bekannten Inkonsistenztypen (A3 und A4). Folglich ist für das Management von Inkonsistenzen in der heterogenen Modelllandschaft im Engineering mechatronischer Systeme ein Ansatz erforderlich, der die Anzahl und Komplexität der Transformationsregeln reduziert.

Zusammenfassung des Stands der Technik
Wie in den vorherigen Ausführungen gezeigt, existiert eine Vielzahl von Ansätzen im Bereich der modellbasierten Entwicklung und im Bereich des Managements von Inkonsistenzen. Diese Ansätze stammen zumeist aus der Domäne der Software-Entwicklung. Ein holistischer Ansatz, der die verschiedenen, heterogenen Modelle, die während des Engineerings automatisierter Produktionssysteme erstellt werden, flexibel integriert und damit das Management von Inkonsistenzen in dieser heterogenen Modelllandschaft ermöglicht, existiert bisher noch nicht. Folglich ist ein Ansatz erforderlich, der durch spezifisches Wissen in den verschiedenen, am Engineering beteiligten Disziplinen sowohl die Integration der Modelle als auch das Management von Inkonsistenzen zwischen und in diesen Modellen ermöglicht.

5 Konzept zur Diagnose von Inkonsistenzen

Um einen ersten Schritt in Richtung des Inkonsistenzmanagements in heterogenen Modelllandschaften zu ermöglichen, wird in diesem Abschnitt ein konzeptueller Ansatz für die Diagnose von Inkonsistenzmanagement vorgestellt. Hierzu wird zunächst ein Beispielszenario eingeführt (Abschn. 5.1). Anschließend wird das wissensbasierte System zum Management von Inkonsistenzen aufgezeigt (Abschn. 5.2) und anhand des Beispielszenarios erläutert.

5.1 Beispielszenario anhand der Pick and Place-Unit (PPU)

Im Beispielszenario werden nun zwei verschiedene Modelle der PPU betrachtet, die repräsentativ für zwei verschiedene Disziplinen des Engineeringprozesses stehen: ein SysML4*Mechatronics*-Modell (vgl. Kernschmidt und Vogel-Heuser 2013), das repräsentativ für die mechatronische Entwicklung steht und ein MATLAB/Simulink-Modell, das genutzt wird, um den erwarteten Werkstückdurchsatz der PPU zu simulieren. Darüber hinaus wird ein Testfall betrachtet, der von einer funktionalen Anforderung an die PPU abgeleitet wurde. Sowohl das MATLAB/Simulink- als auch SysML4*Mechatronics*-Modell definieren überlappende Informationen: beispielsweise die gemessene Winkelgeschwindigkeit der Krankomponente der PPU. Die Herausforderung ist nun, die Modelle konsistent zu halten: So müssen die überlappenden Parameter zueinander konsistente Werte besitzen (vgl. (1) in Abb. 2) und bezüglich des Testfalls überprüft werden (vgl. (2) in Abb. 2). Der Begriff der *Konsistenz* der Werte kann dabei – abhangig vom konkreten Anwendungsfall – unterschiedlich interpretiert werden: In einigen Fällen kann die Identität der Parameter gefordert sein, in anderen wiederum können Unschärfen (z. B. mittels vordefinierter Toleranzgrenzen) erlaubt ein. Selbstverständlich sind diese Beispiele nur Auszüge von möglichen Inkonsistenzen – weitere Inkonsistenzen können beispielsweise infolge fehlerhafter Einheitenumrechnungen, Benennungen, usw. entstehen (Feldmann et al. 2015a).

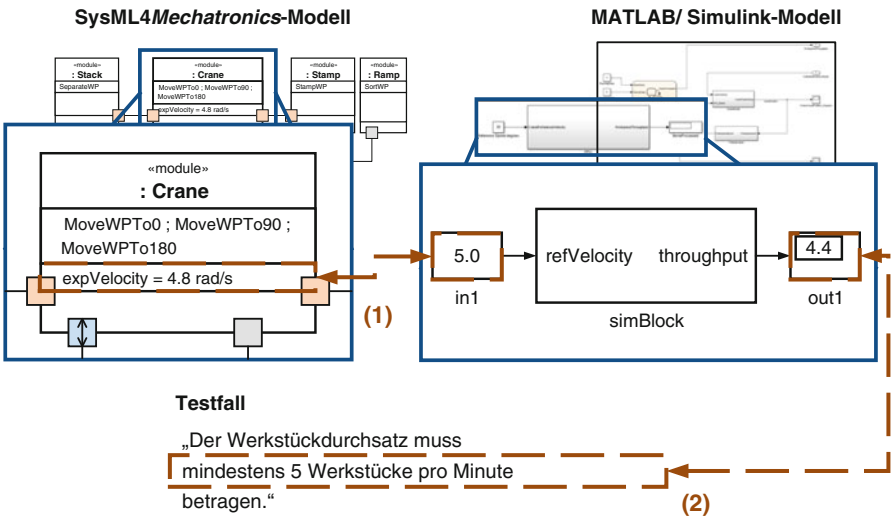

Abb. 2 Modelle im Beispielszenario der PPU (links: SysML4*Mechatronics*-Modell, rechts: MATLAB/Simulink-Modell), angepasst aus (Feldmann et al. 2015a)

5.2 Wissensbasiertes System zur Diagnose von Inkonsistenzen

Um einen ersten Schritt in Richtung des Inkonsistenzmanagements in heterogenen Engineeringdaten und -modellen zu vollziehen, strebt dieser Ansatz zunächst ausschließlich die Diagnose von Inkonsistenzen mittels eines regelbasierten Ansatzes an. Um einen solchen Ansatz zu realisieren, gibt es mitunter zwei Möglichkeiten: Eine Option ist, das Wissen über die Struktur und Semantik der verschiedenen Modelle, ebenso wie die verschiedenen Inkonsistenzen explizit in einem prozeduralen Softwaresystem zu kodieren. Die Wartung und Weiterentwicklung eines solchen Systems dagegen (beispielsweise, um weitere Modelle oder Inkonsistenzen zu integrieren) sind jedoch zeit- und kostenintensiv: Die Integration von n Modellen wurde $n \times (n-1) / 2$ bidirektionale Abbildungen (bzw. $n \times (n-1)$ unidirektionale Abbildungen) erfordern. Daher erfordert die Realisierung eines Ansatzes zum Management von Inkonsistenzen ein hohes Maß an Flexibilität und Erweiterbarkeit. Üblicherweise wird dies durch die Trennung in die *Repräsentation* von Modellen in einem einheitlichen Formalismus und die *Verarbeitung* der Modelle zur Identifikation und Auflösung von Inkonsistenzen realisiert. Dies wird durch die zweite Möglichkeit – einem *wissensbasierten System* – ermöglicht. Wissensbasierte Systeme bestehen aus zwei Komponenten: einer *Wissensbasis* und einem *Inferenzmechanismus.* Erstere dient der Repräsentation von Fakten (d. h. der Kodierung der verschiedenen Modelle). Die zweite Komponente, der Inferenzmechanismus, besteht aus einer Menge an logischen Schlussfolgerungen – oftmals *Wenn-Dann*-Regeln (oder Implikationen) – die zur Verarbeitung des Wissens genutzt werden kann.

RDF als Formalismus zur Wissensrepräsentation

Ein weit verbreiteter Formalismus zur Wissensreprasentation ist das im Kontext der Semantic Web Initiative entstandene Resource Description Framwork (RDF) (W3C 2014). RDF ermöglicht die Abbildung einer Graphenstruktur in *Subjekt-Prädikat-Objekt*-Tripeln. Eine RDF-Repräsentation der Modelle im Beispielszenario ist in Abb. 3 dargestellt. In der RDF-Abbildung des SysML4*Mechatronics*-Modells wird die Aussage „Kran ist ein Modul“ durch das Tripel *Crane rdf:type Module* repräsentiert. Analog dazu werden Aussagen über die erwartete Geschwindigkeit des Krans *(expVelocity)* formuliert, z. B. *Crane owns expVelicity* und *expVelocity value 4.8.* Wie in Abb. 3 gezeigt ist, ist die Terminologie, die genutzt wird, um ein Modell in RDF zu repräsentieren, spezifisch für den jeweiligen Modelltyp (z. B. wird der Ausdruck *value* in MATLAB/Simulink-Modellen genutzt, um ein Attribut mit einer Zahl oder einer Zeichenkette zu verbinden). Die daraus resultierenden RDF-Vokabulare sind in RDF-Namensräumen organisiert und definieren verschiedene (modellspezifische) RDF-Klassen, -Instanzen und -Eigenschaften.

Gleichermaßen können Überlappungen zwischen Modellen in RDF dargestellt werden. Im Beispiel dient ein RDF-Namensraum *overlaps* zur Definition von solchen Überlappungen, beispielsweise um anzugeben, dass zwei Modellelemente die inhaltlich identische Information widerspiegeln. Konkret definiert der Ausdruck

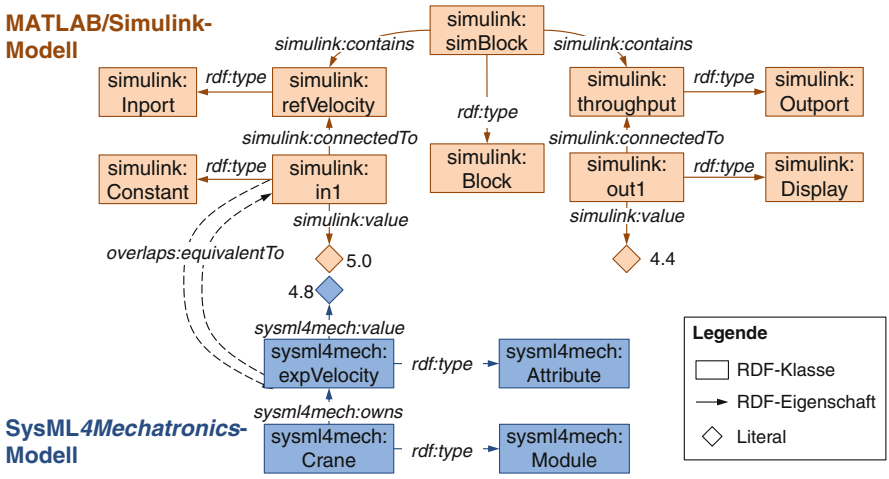

Abb. 3 RDF-Repräsentation der Modelle im Beispielszenario, angepasst aus (Feldmann et al. 2015a)

sysml4mech:expVelocity overlaps:equivalentTo simulink:in1, dass die beiden Attribute semantisch äquivalent sind (siehe Abb. 3). Solche Zusammenhänge können a-priori definiert oder (mittels entsprechender Schlussfolgerungsmechanismen) automatisch identifiziert werden.

SPARQL zur Definition und Identifikation von Inkonsistenzen

Neben RDF zur Wissensrepräsentation existieren Mechanismen, um auf die RDF-Daten zuzugreifen. SPARQL Protocol and RDF Query Language (W3C 2013) ermöglicht es beispielsweise, in RDF gespeicherte Informationen abzufragen und zu verändern. Der Hauptbestandteil des Standards ist die SPARQL Query Langage (im Weiteren SPARQL genannt). SPARQL ähnelt in vielerlei Hinsicht der bekannten Structured Query Language (SQL), die von den meisten relationalen Datenbanksystemen unterstützt wird.

Eine SPARQL-Anfrage besteht aus drei essenziellen Bestandteilen: Der Definition der *Namensräume,* die in der Anfrage verwendet werden sollen, der Definition des *Ausgabeformats* der Abfrage sowie der Definition des *Graph-Musters,* das aus dem RDF-Graphen extrahiert werden soll. SPARQL ermöglicht dabei vier verschiedene Ausgabeformate:

- *SELECT*-Anfragen geben die Werte von Anfragevariablen zurück (beispielsweise in tabellarischer Form),
- *ASK*-Anfragen liefern die Booleschen Werte „wahr“ bzw. „falsch“ zurück – je nachdem ob eine Lösung für die Anfrage existiert oder nicht,
- *CONSTRUCT*-Anfragen ermöglichen die Substitution von Anfrageergebnissen mit definierten Schablonen, und

Anfrage 1

```
@PREFIX simulink: <http://simulink>
@PREFIX sysml4mech: <http://sysml4mechatronics>
@PREFIX overlaps: <http://overlaps>
SELECT * WHERE {
 ?x  overlaps:equivalentTo ?y .
 ?x sysml4mech:value ?xVal . ?y simulink:value ?yVal .
 BIND ( (ABS(?xVal - ?yVal)/(?xVal + ?yVal) > 0.1 )
  AS ?isInconsistent ) }
```

Anfrage 2

```
@PREFIX simulink: <http://example.org/simulink>
SELECT * WHERE {
 simulink:out1  simulink:value ?tVal .
 BIND ( (?tVal < 5 ) AS ?isInconsistent ) }
```

Abb. 4 Inkonsistenzregeln für das Beispielszenario formuliert in SPARQL-Anfragen

- *DESCRIBE*-Anfragen geben einzelne RDF-Graphen zurück, welche die für die Anfrageergebnisse relevanten Daten beinhalten. Da die „Relevanz" der Daten stark abhängig vom Anwendungskontext ist, beinhaltet SPARQL (W3C 2013) keine normative Beschreibung der Ausgabe, die von DESCRIBE-Anfragen erzeugt wird.

Unter der Anforderung, Inkonsistenzen als *Wenn* (Bedingung) *Dann* (Aktion)-Regeln zu formulieren, sind SPARQL SELECT-Anfragen für die Definition von Inkonsistenzregeln geeignet.

In Abb. 4 sind zwei Beispiele für solche Anfragen dargestellt. Eine Art von Inkonsistenz im Beispielszenario bezieht sich auf zwei Modellelemente, deren Werte identisch sein sollen. Diese Modellelemente werden im Beispiel als inkonsistent betrachtet, wenn ihre Werte um mehr als 10 % voneinander abweichen. Anfrage 1 formuliert dieses Kriterium: Existieren zwei Entitäten x und y, welche die Relation *overlaps:equivalentTo* haben und die Werte *xVal* bzw. *yVal* besitzen, so wird das Resultat des logischen Ausdrucks $|xVal - yVal|/(xVal + yVal) > 0.1$ an die Variable *isInconsistent* gebunden. Im vorliegenden Beispiel erkennt die Anfrage somit, dass *expVelocity* und *in1* konsistent sind. Eine zweite Anfrage dient der Überprüfung, ob der zuvor definierte Testfall erfüllt ist oder nicht. Daher wird in Anfrage 2 verglichen, ob der Wert *tVal* von *simulink:out1* (d. h. der Wert des simulierten Durchsatzes) kleiner als 5 ist und bindet das Ergebnis an *isInconsistent.* Diese Bedingung ist in diesem Fall verletzt, da das MATLAB/Simulink-Modell einen Werkstückdurchsatz von 4.4 ermittelt.

6 Diskussion der Ergebnisse und Forschungsgegenstande für zukünftige Arbeiten

Mit Hilfe der Repräsentation von heterogenen Engineeringdaten und deren semantischer Abhängigkeiten (Anforderungen A1 und A2) in RDF-Graphen sowie der Definition von Inkonsistenzregeln mittels SPARQL-Anfragen (A3) lässt sich eine Vielzahl an Inkonsistenztypen identifizieren (Feldmann et al. 2015a). Da Softwarewerkzeuge und -schnittstellen zur Erstellung, Verarbeitung und Anfrage von Daten im Kontext des Semantic Web bereits existieren sowie eine Vielzahl von Ansätzen zur Visualisierung solcher Graphen verfügbar sind (A3–A5), ist auch die Realisierung des Inkonsistenzmanagements mittels dieser Technologien greifbar. Des Weiteren zeigen Performanz- und Skalierbarkeitsanalysen, dass sich RDF-basierte Datenbanken auch für die Anfrage und Verarbeitung großer Datenmengen eignen (siehe z. B. Rohloff et al. 2007).

Gleichzeitig steht der Ansatz zum Inkonsistenzmanagement mittels graphbasierter Repräsentationsformalismen einer zentralen Annahme gegenüber: Die Diagnose von Inkonsistenzen mittels Mustererkennung (z. B. mittels SPARQL) erfordert, dass die zu betrachtenden Modelle in einem graphbasierten Formalismus vorliegen. Zwar ist dies für die technischen Modelle im Engineering automatisierter Produktionssysteme der Fall; allerdings ergeben sich hieraus einige weitere Forschungsgegenstände, die in zukünftigen Arbeiten adressiert werden müssen. Diese werden im Folgenden vorgestellt.

Forschungsgegenstand G1 – Domänenübergreifender Repräsentationsformalismus
Die Heterogenität der Modelle sowie die verschiedenen Abstraktions- und Formalisierungsgrade (z. B. informal in Form von natürlich-sprachlichen Anforderungsdokumenten bzw. formal in zustandsraumbasierten Dynamikmodellen, die mathematisch in Differenzialgleichungen beschreibbar sind) stellen das Inkonsistenz- management vor eine große Herausforderung. Die Anwendbarkeit von RDF zur Repräsentation graphbasierter Modelle konnte am Beispielszenario gezeigt werden. Die Integration von weiteren Modellen – insbesondere solcher, die nicht a priori in graphbasierter Form vorliegen wie beispielsweise textuelle Anforderungsspezifikationen – muss in zukünftigen Forschungsarbeiten erfolgen.

Forschungsgegenstand G2 – Identifikation und Definition von semantischen Abhängigkeiten
Im Rahmen des Beispielszenarios wurde davon ausgegangen, dass semantische Abhängigkeiten a-priori definiert wurden. Die manuelle Spezifikation dieser semantischen Abhängigkeiten zwischen den Modellen (beispielsweise Äquivalenz-, Verfeinerungs- und Abstraktionsbeziehungen), ist jedoch aufgrund der Vielzahl an Abhängigkeiten im Engineering automatisierter Produktionssysteme zeit- und kostenaufwändig. Existierende Ansätze nutzen beispielsweise probabilistische Verfahren (z. B. Herzig und Paredis 2014); deren Anwendbarkeit auf Modelle im Engineering automatisierter Produktionssysteme muss jedoch gepruft werden.

Forschungsgegenstand G3 – Verarbeitungsmechanismen zur Diagnose und Auflösung von Inkonsistenzen
Ist ein gemeinsamer Repräsentationsformalismus sowie ein Konzept zur Identifikation und Definition von semantischen Abhängigkeiten verfügbar, so können geeignete Verarbeitungsmechanismen zur Diagnose und Auflösung von Inkonsistenzen realisiert werden. Im Beispielszenario wurde die Anwendbarkeit von RDF und SPARQL für die Diagnose von einfachen, logischen Inkonsistenzen gezeigt. Die Erweiterung des Ansatzes um verschiedenartige Inkonsistenztypen, sowie um die Beantwortung der Frage, wie sich Inkonsistenzen im Engineering automatisierter Systeme auswirken und wie der Aufwand zur Auflösung anhand vordefinierter Kriterien abgeschätzt werden kann (z. B. Grad der Auswirkung in den Modellen, Konsequenzen der Inkonsistenz), muss in zukünftigen Arbeiten erfolgen.

Neben den Mechanismen zur Diagnose von Inkonsistenzen erfordert das Engineering automatisierter Produktionssysteme auch die Generierung von Auflösungsalternativen, um die Entscheidungsfindung der beteiligten Akteure zu unterstützen. Im Besonderen müssen die möglichen Auflösungsalternativen (teil-)automatisch identifiziert und anhand definierter Kriterien (z. B. Art und Anzahl involvierter Modellelemente) priorisiert werden. Anhand der diagnostizierten Inkonsistenzen und priorisierten Auflösungsvorschläge kann den Akteuren eine Entscheidungsgrundlage geboten werden.

Forschungsgegenstand G4 – Unterstützungsmethodik zum Inkonsistenzmanagement
Eine vollständig automatische Diagnose und Auflösung von Inkonsistenzen ist jedoch weder möglich noch erstrebenswert: Ideen werden von den am Engineering beteiligten Akteuren kreiert, indem Alternativen erstellt, evaluiert und revidiert werden. Folglich muss in vielen Fällen der jeweilige Akteur selbst entscheiden können, welche Handlung aufgrund des Auftretens einer Inkonsistenz erforderlich ist, indem er z. B. abwägt, ob eine Auflösung der Inkonsistenz erforderlich/gewünscht ist oder nicht. Infolgedessen muss untersucht werden, in welchen Fällen eine automatische Verarbeitung im Inkonsistenzmanagement möglich ist und an welchen Stellen manuell eingegriffen werden muss. Ist diese Fragestellung beantwortet, so kann eine Unterstützungsmethodik zum Inkonsistenzmanagement in heterogenen Modelllandschaften erarbeitet werden, welche die geeignete Synergie zwischen (teil-)automatischer Verarbeitung der Modelle sowie manuellem Eingriff der jeweiligen Akteure erlaubt. Eine solche Unterstützungsmethodik besteht dabei aus drei zentralen Bestandteilen: (1) *Handlungsempfehlungen,* welche die Einführung und Realisierung eines Ansatzes zum Inkonsistenzmanagement in heterogenen Modelllandschaften vereinfachen, (2) *Richtlinien,* um insbesondere Inkonsistenzen, die durch einen automatischen Ansatz nicht erkannt oder aufgelöst werden können, abzudecken, sowie (3) *Visualisierungstechniken,* welche durch geeignete Darstellungen und Interaktionen die manuelle Auflösung von Inkonsistenzen disziplinübergreifend ermöglichen.

Danksagung Die Autoren danken der Deutschen Forschungsgemeinschaft (DFG) für die Förderung dieser Arbeit als Teil des Sonderforschungsbereichs 768: Zyklenmanagement von Innova-

tionsprozessen – verzahnte Entwicklung von Leistungsbündeln auf Basis technischer Produkte (SFB 768). Des Weiteren danken die Autoren Christiaan J.J. Paredis, Sebastian J.I. Herzig und Ahsan Qamar (Georgia Institute of Technology) für ihre Unterstützung und fruchtbaren Diskussionen.

Literatur

Basole R, Qamar A, Park H, Paredis C, McGinnis L (2015) Visual analytics for early-phase complex engineered system design support. IEEE Computer Graphics and Applications 35(2):41–51. doi:10.1109/MCG.2015.3

Bassi L, Secchi C, Bonfe M, Fantuzzi C (2011) A SysML-based methodology for manufacturing machinery modeling and design. IEEE/ASME Transactions on Mechatronics 16(6):1049–1062. doi:10.1109/TMECH.2010.2073480

Biffl S, Winkler D, Mordinyi R, Scheiber S, Holl G (2014) Efficient monitoring of multi-disciplinary engineering constraints with semantic data integration in the multi-model dash-board process. In: IEEE International Conference on Emerging Technology and Factory Automation, S 1–10. doi:10.1109/ETFA.2014.7005211

Biffl S, Maetzler E, Wimmer M, Luder A, Schmidt N (2015) Linking and versioning support for automationml: a model-driven engineering perspective. In: IEEE International Conference on Industrial Informatics, S 499–506. doi:10.1109/INDIN.2015.7281784

Bill R, Steyskal S, Wimmer M, Kappel G. (2014) On synergies between model transformations and semantic web technologies. In: International Conference on Model Driven Engineering Languages and Systems – workshop on multi-paradigm modeling, S 31–40. Valencia

Brix T, Reeßing M (2009) Domain spanning design tools for heterogeneous systems. In: International Conference on Engineering Design, Palo Alto, US-CA

Broy M, Feilkas M, Herrmannsdoerfer M, Merenda S, Ratiu D (2010) Seamless model-based development: from isolated tools to integrated model engineering environments. Proceedings of the IEEE 98(4):526–545. doi:10.1109/JPROC.2009.2037771

Deutsches Institut für Normung e.V. (2015) Datenaustauschformat für Planungsdaten industrieller Automatisierungssysteme – AutomationML – Teil 1: Architektur und allgemeine Festlegun gen

Drath R (Hrsg) (2010) Datenaustausch in der Anlagenplanung mit AutomationML: Integrationvon CAEX, PLCopen XML und COLLADA. Springer

Egyed A (2011) Automatically detecting and tracking inconsistencies in software design models. IEEE Transactions on Software Engineering 37(2):188–204. doi:10.1109/TSE.2010.38

Feldmann S, Kernschmidt K, Vogel-Heuser B (2014) Combining a SysML-based modeling approach and semantic technologies for analyzing change influences in manufacturing plant models. In: CIRP Conference on Manufacturing Systems, S 451–456. doi:10.1016/j.procir.2014.01.140

Feldmann S, Herzig SJI, Kernschmidt K, Wolfenstetter T, Kammerl D, Qamar A, Lindemann U, Krcmar H, Paredis CJJ, Vogel-Heuser B (2015a) Towards effective management of inconsistencies in model-based engineering of automated production systems. In: IFAC Symposium on Information Control Problems in Manufacturing. doi:10.1016/j.ifacol.2015.06.200

Feldmann S, Herzig SJI, Kernschmidt K, Wolfenstetter T, Kammerl D, Qamar A, Lindemann U, Krcmar H, Paredis CJJ, Vogel-Heuser B (2015b) A Comparison of Inconsistency Management Approaches Using a Mechatronic Manufacturing System Design Case Study. In: 10th IEEE International Conference on Automation Science and Engineering (CASE), Gotheburg, Sweden

Finkelstein AC, Gabbay D, Hunter A, Kramer J, Nuseibeh B (1994) Inconsistency handling in multiperspective specifications. IEEE Transactions on Software Engingeering 20(8):569–578

Gausemeier J, Schäfer W, Greenyer J, Kahl S, Pook S, Rieke J (2009) Management of cross-domain model consistency during the development of advanced mechatronic systems. In: International Conference on Engineering Design, Palo Alto, US-CA

Giese H, Wagner R (2009) From model transformation to incremental bidirectional model synchronization. Software and Systems Modeling 8(1):21–43. doi:10.1007/s10270-008-0089-9

Hamraz B, Caldwell NHM, Clarkson PJ (2013) A holistic categorization framework for literature on engineering change management. Systems Engineering 16(4):473–505. doi:10.1002/sys.21244

Hegedus A, Horvath A, Rath I, Branco M, Varro D (2011) Quick fix generation for DSMLs. In: IEEE symposium on visual languages and human-centric computing, S 17–24. doi:10.1109/VLHCC.2011.6070373

Hehenberger P, Egyed A, Zeman K (2007) Consistency checking of mechatronic design models. In: ASME international design engineering technical conferences and computers and information in engineering conference. doi:10.1115/DETC2010–28615

Herzig SJI, Paredis CJJ (2014) Bayesian reasoning over models. In: Workshop model-driven Eng., verification, and validation

Herzig SJ, Qamar A, Paredis CJ (2014) An approach to identifying inconsistencies in model-based systems engineering. In: Conference on Systems Engineeirng Research 28:354–362. doi:10.1016/j.procs.2014.03.044

Isermann R (2008) Mechatronic systems – innovative products with embedded control. Control Eng Practice 16(1):14–29

Kernschmidt K, Vogel-Heuser B (2013) An interdisciplinary SysML based modeling approach for analyzing change influences in production plants to support the engineering. In: IEEE International Conference on Automation Science and Engineering, S 1113–1118. doi:10.1109/CoASE.2013.6654030

Kovalenko O, Serral E, Sabou M, Ekaputra FJ, Winkler D, Biffl S (2014) Automating cross-disciplinary defect detection in multi-disciplinary engineering environments. In: Janowicz K, Schlobach S, Lambrix P, Hyvönen E (Hrsg) Knowledge engineering and knowledge management. Lecture notes in computer science, Bd 8876. Springer, S 238–249

Lin HY, Sierla S, Papakonstantinou N, Shalyto A, Vyatkin V (2015) Change request management in model-driven engineering of industrial automation software. In: IEEE International Conference on Industrial Informatics, S 1186–1191. doi:10.1109/INDIN.2015.7281904

Mens T, Van Der Straeten R, D'Hondt M (2006) Detecting and resolving model inconsistencies using transformation dependency analysis. In: Model driven engineering languages and systems. Lecture notes in computer science, Bd 4199. Springer, S 200–214. doi:10.1007/11880240_15

Moser T, Biffl S (2012) Semantic Integration of Software and Systems Engineering Environments. IEEE Transactions on Systems, Man and Cybernetics – Part C 42(1):38–50. doi:10.1109/TSMCC.2011.2136377

NASA: Report on Project Management in NASA: Phase II of the Mars Climate Orbiter Mishap Report (2000). ftp://ftp.hq.nasa.gov/pub/pao/reports/2000MCOJMIB_Report.pdf. Zugegriffen am 26.01.2016

Nuseibeh B, Easterbrook S, Russo A (2000) Leveraging inconsistency in software development. Computer 33(4):24–29. doi:10.1109/2.839317

OMG: Systems Modeling Language (SysML), Version 1.3 (2012). http://www.omg.org/spec/SysML/1.3. Zugegriffen am 26.01.2016

OMG: Unified Modeling Language (UML), Version 2.4.1 (2011). http://www.omg.org/spec/UML/2.4.1/. Zugegriffen am 26.01.2016

Rieke J, Dorociak R, Sudmann O, Gausemeier J, Schäfer W (2012) Management of cross-domain model consistency for behavioral models of mechatronic systems. In: International Design Conference

Rohloff K, Dean M, Emmons I, Ryder D, Sumner J (2007) An evaluation of triple-store technologies for large data stores. In: OTM Confederated International Conference on On the Moveto Meaningful Internet Systems, S 1105–1114

Schäfer W, Wehrheim H (2010) Model-driven development with mechatronic UML. In: Engels G, Lewerentz C, Schäfer W, Schürr A, Westfechtel B (Hrsg) Graph transformations and

model-driven engineering, Lecture notes in computer science, Bd 5765. Springer, S 533–554. Doi:10.1007/978-3-642-17322-6_23

Schätz B, Braun P, Huber F, Wisspeintner A (2003) Consistency in model-based development. In: IEEE international conference and workshop on the engineering of computer-based systems, S 287–296. doi:10.1109/ECBS.2003.1194810

Secchi C, Bonfe M, Fantuzzi C (2007) On the Use of UML for Modeling Mechatronic Systems. IEEE Transactions on Automation Science and Engineering 4(1):105–113. doi:10.1109/TASE.2006.879686

Spanoudakis G, Zisman A (2001) Inconsistency management in software engineering: Survey and open research issues. In: Chang SK (Hrsg) Handbook of software engineering and knowledge engineering. World Scientific, Singapore, S 329–380

Thramboulidis K (2013) Overcoming mechatronic design challenges: the 3 + 1 SysML-view model. J Comput Sci Techn 1(1):6–14

Van Der Straeten R, D'Hondt M (2006) Model refactorings through rule-based inconsistency resolution. In: ACM Symposium on Applied Computing, S 1210–1217. doi:10.1145/1141277.1141564

Vogel-Heuser B, Legat C, Folmer J, Feldmann S (2014) Researching evolution in industrial plant automation: scenarios and documentation of the pick and place. Unit. Tech. Rep. TUM-AIS-TR-01-14-02, Technische Universitat München. https://mediatum.ub.tum.de/node?id=1208973. Zugegriffen am 26.01.2016

Vogel-Heuser B, Legat C, Folmer J, Rosch S (2014b) Challenges of Parallel Evolution in Production Automation Focusing on Requirements Specification and Fault Handling. at – Automatisierungstechnik 62(11):758–770

W3C: Resource Description Framework (RDF) (2014). http://www.w3.org/RDF/. Zugegriffen am 26.01.2016

W3C: SPARQL Protocol and RDF Query Language 1.1 overview (2013). http://www.w3.org/TR/sparql11-overview/. Zugegriffen am 26.01.2016