2017 IEEE 25th International Requirements Engineering Conference Workshops

Traceability for Automated Production Systems: A Position Paper

Mounifah Alenazi, Nan Niu, Wentao Wang, and Arushi Gupta
Department of Electrical Engineering and Computing Systems, University of Cincinnati, USA
alenazmh@mail.uc.edu, nan.niu@uc.edu, wang2wt@mail.uc.edu, gupta2ai@mail.uc.edu

***Abstract*—Automated production systems are design-to-order, custom-built mechatronic systems that are intended to deliver automation capabilities to satisfy the stakeholder requirements in the manufacturing/production domain. Traceability has its research root in requirements engineering (RE) and is defined as "the ability to describe and follow the life of a requirement". Such descriptions and followings have mainly been scoped within a single software project's process and artifacts. We argue that, in the context of automated production systems, the scope of traceability shall go beyond the software engineering boundary into the environments which the software is operated. We outline in this position paper our aim to define a new form of the trace links and to explore the role of these links in the RE of automated production systems. In particular, we adapt a formal interaction-oriented RE framework that focuses on specifying the commitments of participants rather than the goals of each individual participant. We then integrate model checking into the framework to elaborate when, why, and how much traceability should be instrumented. We demonstrate our approach with a bench-scale automated production system where model-driven engineering practices such as constructing and evolving SysML models are adopted.**

***Index Terms*—Automated production systems, requirements traceability, systems traceability, SysML, model checking.**

I. Introduction

In their seminal work, Gotel and Finkelstein [1] defined *traceability* as "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)." Since then, much effort has been made in researching specific areas of the traceability problem, such as trace link creation, visualization, and maintenance.

The scope of current traceability work is primarily within a single software project. This is reflected in a recent definition where traceability is referred to as "the ability to interrelate any uniquely identifiable software artifact to any other, maintain required links over time, and use the resulting network to answer questions of both the software product and its development process" [2].

Scoping traceability within the software engineering boundary shapes the form of the trace links, i.e., what counts as a link and what does not. For example, in many trace link recovery approaches like [3, 4, 5], a link is composed of a source artifact (e.g., a textual requirement serving as the trace query) and a target artifact (e.g., a Java method implementing the requirement). Such a form (the "what") is tightly coupled with the "how", i.e., the use of information retrieval (IR) techniques to automatically generate the candidate links for a given trace query. The coupling, in turn, influences *when* to trace and *how much* traceability information is needed. IR-based techniques support after-the-fact tracing (the "when") and strive to retrieve all the correct links and only the correct ones, i.e., to achieve both recall and precision at a 100% level (the "how much").

The "what", "how", "when", and "how much" are all driven by the "why" [6], and for *requirements* traceability, an important "why" is to understand the fulfillment of a particular requirement or a set of requirements. In certain kinds of software-intensive systems such as a Web application, a requirement (e.g., automatic logout) can be fulfilled within software engineering. In other words, it is sufficient to establish trace links between the requirement and other software artifacts such as source code, test cases, etc. However, the requirements of some other systems cannot be satisfied by software alone. This is especially the case for automated production systems where software must integrate and coordinate well with the mechatronic environments in which it operates.

Automated production systems are the key enablers for Industry 4.0 that is aimed to fully leverage cyber-physical systems in the manufacturing/production domain [7]. An example is the special-purpose machinery for producing consumer goods like yogurt. A more complex example of automated production systems is the wood-working plant integrating hybrid processes (continuous and discrete facility modules) and logistic processes such as warehouse management [8]. Clearly, tracking the life of a requirement in these systems cannot and should not be limited only to the software artifacts. Our objective of this paper is to explore the form and the usage of trace links in automated production systems where model-driven development is practiced.

II. Background and Related Work

We use the Pick & Place Unit (PPU) to ground our discussions about automated production systems, as well as the requirements and the traceability information in these systems. The PPU is a bench-scale demonstrator derived from industrial use cases to evaluate research results at different stages of the engineering process in the mechatronic manufacturing domain [9]. Although being a simple case, the PPU is complex enough to demonstrate an excerpt of the challenges that arise

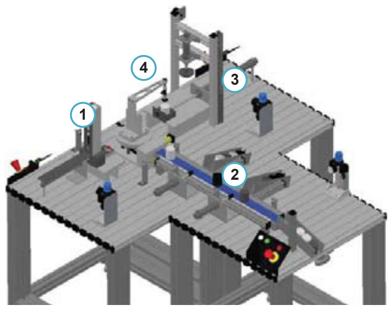
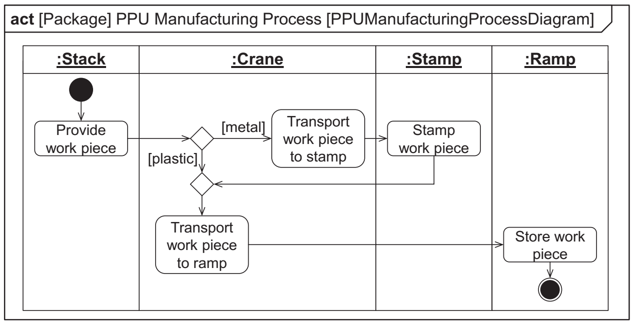


Fig. 1. High-level structural overview of the PPU: ① stack, ② ramp, ③ stamp, and ④ crane [9].



act [Package] PPU Manufacturing Process [PPUManufacturingProcessDiagram]
:Stack
:Crane
:Stamp
:Ramp
Provide work piece
[metal]
[plastic]
Transport work piece to stamp
Stamp work piece
Transport work piece to ramp
Store work piece

Fig. 2. Manufacturing process of the PPU [9].

during engineering of automated production systems in the context of Industry 4.0.

Figure 1 shows four mechatronic components that consist of sensors and actuators requiring control for operation. Although these hardware components are common in embedded systems, automated production systems like the PPU are designed specific for manufacturing/production domains. Functionality wise, the stack (Figure 1①) acts as an input storage where black plastic, white plastic, and metallic work pieces are fed for the PPU to process. The work pieces are pushed from the stack into a handover position. At the handover position, sensors are installed to identify the type of work piece provided by the stack.

The ramp (Figure 1②) serves as the work piece depot storing the outputs after the processing from the stamp (Figure 1③). Depending on the type of the work piece, different pressures are applied over the stamp. The crane (Figure 1④) transports work pieces by picking and placing them between the three positions: stack, stamp, and ramp. Note that Figure 1 shows only a high-level structural view of the PPU. More detailed ones can be found in [9], e.g., zooming in the crane shows that the work pieces are gripped or released by a vacuum gripper, which is mounted on a boom.

For the initial configuration of the PPU, the manufacturing process is illustrated in Figure 2 using a Systems Modeling Language (SysML)[1] activity diagram. SysML is now commonly used in many industry sectors and has become a *de facto* standard for systems engineering [10]. While extensively reusing UML[2], SysML also provides its own extensions like the parametric diagram. A key distinction is that SysML expresses systems engineering semantics (interpretations of modeling constructs) better than UML, allowing different engineering disciplines to externalize and communicate their concerns.

Like other automated production systems, the PPU continuously experiences the changing requirements. Because these systems are design-to-order and custom-built, it is to the best interest of all the stakeholders (customers, end users, engineering team, etc.) if the automated production systems can sustain the various kinds of changes rather than being short lived and then expensively recreated [11]. The PPU case study provides a rich set of engineering tradeoffs when the changing requirements are introduced [9]. For example, one change requires the addition of two more conveyors and the respective pushing cylinders to the ramp. Although software alone cannot fully address the changes, it plays an important and integral role in satisfying the requirements.

Software in the PPU, and in automated production systems in general, is developed on Programmable Logic Controllers (PLCs) [8]. PLCs are characterized by their cyclic data processing behavior: reading sensor input values, storing them in a process image, computing, and writing output values to control the actuators. While variants exist (e.g., PLCopen, Siemens PLC, and Rockwell PLC), PLCs currently are and will remain the state of industrial practice for some decades [8]. The coexistence and the interdependency of different engineering artifacts such as SysML models and PLC codes challenge how the PPU meets the requirements. We next survey the research on traceability related to model-driven systems development.

The use of IR methods has received much attention to automatically recover the traceability information within a software project [2]. Beyond the software boundary, the applicability of textual cues is weakened though not entirely lost. For example, Czauderna *et al.* [12] extended the probabilistic network IR models to trace how product requirements comply with regulatory standards. Nejati *et al.* [13] proposed an automated approach to identify the impact of requirements changes on system design, when the requirements and design elements are expressed using SysML models. Two steps are involved in [13]: extracting estimated set of impacted model elements via static slicing, and ranking those model elements based on textual information.

While the textual information is helpful in identifying and ranking the candidate traceability links, the scope of applicability of IR-based trace recovery techniques is the set of systems which are "textually rich". However, automated production systems have become increasingly "model rich". Abilov *et al.* [14] pointed out the lack of traceability support for model transformations and defined a general mapping scheme and rule to achieve model synchronization. Li and colleagues performed variability analysis of SysML-based

[1]http://www.omgsysml.org

[2]http://www.uml.org

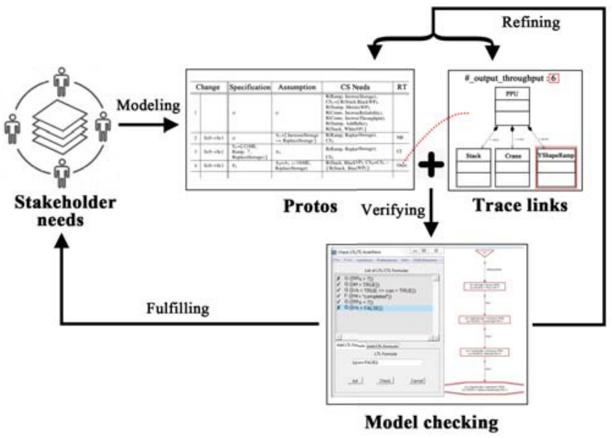


Fig. 3. Establishing and using the trace links to refine, verify, and fulfill requirements of automated production systems.

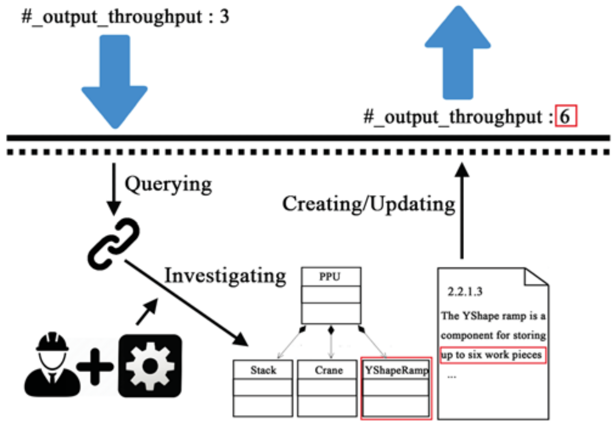


Fig. 4. Creation, maintenance, and use of the trace links.

requirements for embedded real-time systems to improve the consistency among system models [15].

In summary, identifying the traceability information via textual cues has been attempted in model-driven RE [16, 17, 18, 19]. The reported experience on the PPU suggests that deep domain knowledge is of crucial importance in linking the elements from the many heterogeneous models: A case in point is the trace link between the SysML element "expVelocity" and the Simulink variable "in1" [20]. Orthogonal to the specific way that the trace links are generated, our approach presented next embeds the traceability information in a broad array of model-driven RE activities for automated production systems.

III. Traceability for Fulfilling Requirements of Automated Production Systems

Our vision of creating, updating, and using the trace links in the RE for automated production systems is overviewed in Figure 3. To illustrate our proposed approach, we use a change scenario of the PPU where the ramp is replaced for the purpose to increase throughput [9]. Such a change is regarded as a requirement to be satisfied, and in Figure 3, this is expressed as part of the stakeholder needs that drive the entire process of our approach.

Because satisfying the requirements of automated production systems like "changing the ramp to increase throughput" involves interacting engineers (e.g., mechanical engineers and software engineers), we base our approach on an RE framework called Protos [21] that formalizes the rules of encounter in a socio-technical system. Different from traditional requirements models emphasizing the goals of the individual principals [22], Protos gives prominence to their social relationships, specifically the *commitments* that the principals have for each other. A commitment is formally expressed as a four-tuple $\mathsf{C}(debtor, creditor, antecedent, consequent)$: the debtor is committed to the creditor that if the antecedent holds, the consequent will hold.

Table I shows an example of Protos-based requirements modeling for our ramp change scenario. This example is based on the scenario change Sc0→Sc1 documented in [9]. The change set groups related needs to be satisfied by the PPU, e.g., $\mathsf{R}(\textsc{Ramp}, \textsc{IncreaseStorage})$ represents the requirement for the output of the PPU ($\textsc{Ramp}$) to increase its storage. The commitment, $\mathsf{C}(\textsc{ME}, \textsc{Ramp}, \mathrm{T}, \mathrm{ReplaceStorage})$, specifies the expectation where the mechanical engineer ($\textsc{ME}$) is committed to the $\textsc{Ramp}$ in such a way that under no specific conditions ($\textsc{True}$) the output storage shall be replaced ($\textsc{ReplaceStorage}$).

Besides the built-in modeling construct explicating the commitments, Protos allows for a series of refinements to systematically derive the specifications from the requirements [21]. Formally, refinement is a relation between problems p and p' where p' is an incremental improvement of p such that a solution for problem p' also constitutes a solution for p. The rightmost column of Table I shows the types of refinement applied at each Protos step: need refinement, commitment introduction, etc. While other refinements exist in model-driven RE [23], Protos defines its own refinement types [21].

The main benefits of Protos are the considerations of openness and autonomy [24], facilitated by a process whose starting point consists of only the needs/requirements and whose completion point is reached when the needs are empty. However, a limitation is the lack of checking of whether a commitment is met or not. While the participants should have their own autonomy, the commitment that they have of each other must be guaranteed.

To provide support for checking the actual delivery of the specified commitment, we complement Protos with the trace links of the automated production systems. Figure 4 presents our idea in which an *interface* is drawn between the information exposed to the model checker (discussed below) and the information used for the development and management of the trace links themselves. It is the interface that we consider as the new form of traceability in automated production systems. In another word, much information should not be shared among the principals in order to promote autonomy and innovation

TABLE I
COMMITMENTS AND REFINEMENTS IN PROTOS

	Specification	Assumptions	Needs in terms of change set (CS)	Refinement Type
1	ϕ	ϕ	R(Ramp, IncreaseStorage), CS_1 = {R(Stack, BlackWP) ...}	
2	ϕ	A_1 = {IncreaseStorage $\hookrightarrow$ ReplaceStorage}	R(Ramp, ReplaceStorage), CS_1	Need Refinement
3	C(ME, Ramp, $\top$, ReplaceStorage)	A_1	R(Ramp, ReplaceStorage), CS_1	Commitment Introduction II
4	C(ME, Ramp, $\top$, ReplaceStorage)	$A_2 = A_1 \cup$ O(ME, ReplaceStorage)	R(Stack, BlackWP), $CS_2 = CS_1 -$ R(Stack, BlackWP)	Onus
...	...	...	...	...

of a principal. However, once the traceability information impacts the interactions of the principals, it must be properly established and maintained. Tracing is not simply attaching a piece of information (e.g., a code snippet, a model element, etc.) from one principal to another; rather, the management of the trace links must be performed in a mutual manner.

In our example, when the mechanical engineer attempts to discharge the commitment, C(ME, Ramp, $\top$, ReplaceStorage), the throughput requirement, "number of output work pieces = 3", guides the work on the trace links. The engineer with tool support queries existing links relevant to the throughput property, investigates the changes made, and updates the traceability information. As a result, the throughput property of the PPU is re-asserted to be equal to six work pieces. In Figure 4, the change reflected in the SysML model helps justify the storage increase of the output of the PPU (Ramp). Making the relevant traceability information explicit yet just-in-time [25] underpins our ongoing work.

Properties like "#_output_throughput : 6" supported by the trace links are verified automatically in our approach. Currently, we explore the use of modeling checking tools for this purpose. This is inspired by Telang and Singh's work that preceded Protos [26] in which the syntactic properties are verified by temporal model checking tools. Note that we distinguish between the syntactic properties of the correctness and completeness of refinements in Protos and the semantic properties related to the specific automated production system (e.g., the throughput constraint of the PPU). Figure 5 shows a simple state transition diagram in which the commitment under discussion is progressed. In each state of Figure 5, we explicitly separate the syntactic state variables from the semantic ones. We implement the model using TLA+[3] as shown in Figure 6.

IV. CONCLUDING REMARKS

In this position paper, we outline an approach to making the traceability information valuable in the RE for automated production systems. The trace links, along with model checking, complement the Protos framework in that commitments can be explicitly tracked and automatically verified. Altogether, the approach provides a formal, thorough, and machine-checkable way for fulfilling the stakeholder needs and desires in the long-living and ever-changing automated production systems. As far

[3]http://lamport.azurewebsites.net/tla/tla.html

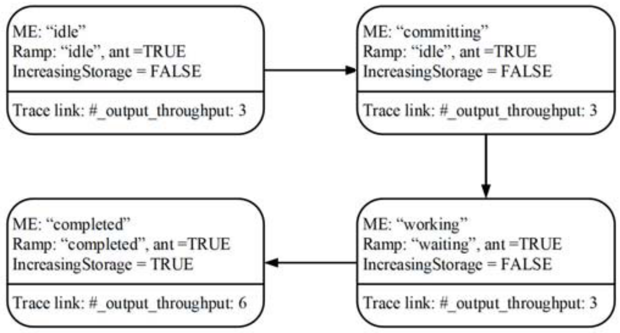
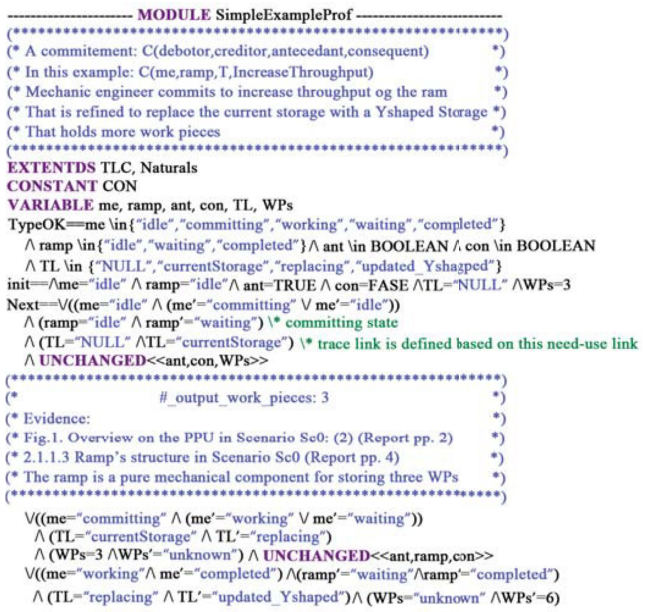

ME: "idle"
Ramp: "idle", ant =TRUE
IncreasingStorage = FALSE
Trace link: #_output_throughput: 3
ME: "committing"
Ramp: "idle", ant =TRUE
IncreasingStorage = FALSE
Trace link: #_output_throughput: 3
ME: "completed"
Ramp: "completed", ant =TRUE
IncreasingStorage = TRUE
Trace link: #_output_throughput: 6
ME: "working"
Ramp: "waiting", ant =TRUE
IncreasingStorage = FALSE
Trace link: #_output_throughput: 3

Fig. 5. State transition diagram amenable for model checking.



```
---------------------- MODULE SimpleExampleProf -------------------------
(***************************************************************)
(* A commitement: C(debotor,creditor,antecedant,consequent)        *)
(* In this example: C(me,ramp,T,IncreaseThroughput)                *)
(* Mechanic engineer commits to increase throughput og the ram     *)
(* That is refined to replace the current storage with a Yshaped Storage *)
(* That holds more work pieces                                      *)
(***************************************************************)
EXTENTDS TLC, Naturals
CONSTANT CON
VARIABLE me, ramp, ant, con, TL, WPs
TypeOK==me \in{"idle","committing","working","waiting","completed"}
   ∧ ramp \in{"idle","waiting","completed"}∧ ant \in BOOLEAN ∧ con \in BOOLEAN
   ∧ TL \in {"NULL","currentStorage","replacing","updated_Yshagped"}
init==∧me="idle" ∧ ramp="idle"∧ ant=TRUE ∧ con=FASE ∧TL="NULL" ∧WPs=3
Next==∨((me="idle" ∧ (me'="committing" ∨ me'="idle"))
   ∧ (ramp="idle" ∧ ramp'="waiting") \* committing state
   ∧ (TL="NULL" ∧TL="currentStorage") \* trace link is defined based on this need-use link
   ∧ UNCHANGED<<ant,con,WPs>>
(***************************************************************)
(*                    #_output_work_pieces: 3                    *)
(* Evidence:                                                       *)
(* Fig.1. Overview on the PPU in Scenario Sc0: (2) (Report pp. 2)  *)
(* 2.1.1.3 Ramp's structure in Scenario Sc0 (Report pp. 4)          *)
(* The ramp is a pure mechanical component for storing three WPs   *)
(***************************************************************)
   ∨((me="committing" ∧ (me'="working" ∨ me'="waiting"))
    ∧ (TL="currentStorage" ∧ TL'="replacing")
    ∧ (WPs=3 ∧WPs'="unknown") ∧ UNCHANGED<<ant,ramp,con>>
   ∨((me="working"∧ me'="completed") ∧(ramp'="waiting"∧ramp'="completed")
    ∧ (TL="replacing" ∧ TL'="updated_Yshaped")∧ (WPs="unknown" ∧WPs'=6)
```

Fig. 6. TLA+ model snippet.

as the trace links are concerned, we realize that the "what" and the "how" are coupled; however, this coupling offers insights into the "when" and the "how much". Specifically, we believe that, each time a participant delivers a commitment, the trace links shall be created, queried, or otherwise updated. While the trace links can be of different abstraction and detail, our approach suggests an interface where only the checkable properties are exposed and therefore need to be stored and maintained. Our future work includes testing more changing requirements of the PPU and other instances of automated production systems, as well as developing reusable components for TLA+ model construction and templates for temporal property specification.

References

[1] O. Gotel and A. Finkelstein, "An analysis of the requirements traceability problem," in *International Conference on Requirements Engineering (ICRE)*, Colorado Springs, CO, USA, April 1994, pp. 94–101.

[2] J. Cleland-Huang, O. Gotel, J. H. Hayes, P. Mäder, and A. Zisman, "Software traceability: trends and future directions," in *Future of Software Engineering (FOSE)*, Hyderabad, India, May-June 2014, pp. 55–69.

[3] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: the study of methods," *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 4–19, January 2006.

[4] N. Niu and A. Mahmoud, "Enhancing candidate link generation for requirements tracing: the cluster hypothesis revisited," in *International Requirements Engineering Conference (RE)*, Chicago, IL, USA, September 2012, pp. 81–90.

[5] A. Mahmoud and N. Niu, "On the role of semantics in automated requirements tracing," *Requirements Engineering*, vol. 20, no. 3, pp. 281–300, September 2015.

[6] N. Niu, W. Wang, and A. Gupta, "Gray links in the use of requirements traceability," in *International Symposium on Foundations of Software Engineering (FSE)*, Seattle, WA, USA, November 2016, pp. 384–395.

[7] B. Vogel-Heuser and D. Hess, "Industry 4.0—prerequisites and visions," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 411–413, April 2016.

[8] B. Vogel-Heuser, J. Fischer, S. Rösch, S. Feldmann, and S. Ulewicz, "Challenges for maintenance of PLC-software and its related hardware for automated production systems: selected industrial case studies," in *ICSME*, Bremen, Germany, September-October 2015, pp. 362–371.

[9] B. Vogel-Heuser, C. Legat, J. Folmer, and S. Feldmann, "Researching evolution in industrial plant automation: scenarios and documentation of the pick and place unit," Technical University Munich, Technical Report TUM-AIS-TR-01-14-02, 2014.

[10] W. Schäfer and H. Wehrheim, "The challenges of building advanced mechatronic systems," in *Future of Software Engineering (FOSE)*, Minneapolis, MN, USA, May 2007, pp. 72–84.

[11] J. Savolainen, N. Niu, T. Mikkonen, and T. Fogdal, "Long-term product line sustainability through planned staged investments," *IEEE Software*, vol. 30, no. 6, pp. 63–69, November/December 2013.

[12] A. Czauderna, J. Cleland-Huang, M. Çinar, and B. Berenbach, "Just-in-time traceability for mechatronics systems," in *International Workshop on Requirements Engineering for Systems, Services, and Systems-of-Systems (RESS)*, Chicago, IL, USA, September 2012, pp. 1–9.

[13] S. Nejati, M. Sabetzadeh, C. Arora, L. C. Briand, and F. Mandoux, "Automated change impact analysis between SysML models of requirements and design," in *International Symposium on Foundations of Software Engineering (FSE)*, Seattle, WA, USA, November 2016, pp. 242–253.

[14] M. Abilov, T. Mahmoud, J. M. Gómez, and M. Mora, "Towards an incremental bidirectional partial model synchronization between organizational and functional requirements models," in *International Model-Driven Requirements Engineering Workshop (MoDRE)*, Ottawa, Canada, August 2015, pp. 1–10.

[15] M. Li, F. Batmaz, L. Guan, A. Grigg, M. Ingham, and P. Bull, "Model-based systems engineering with requirements variability for embedded real-time systems," in *International Model-Driven Requirements Engineering Workshop (MoDRE)*, Ottawa, Canada, August 2015, pp. 36–45.

[16] M. Fockel and J. Holtmann, "A requirements engineering methodology combining models and controlled natural language," in *International Model-Driven Requirements Engineering Workshop (MoDRE)*, Karlskrona, Sweden, August 2014, pp. 67–76.

[17] O. B. Badreddin, A. Sturm, and T. C. Lethbridge, "Requirement traceability: A model-based approach," in *International Model-Driven Requirements Engineering Workshop (MoDRE)*, Karlskrona, Sweden, August 2014, pp. 87–91.

[18] N. Sannier and B. Baudry, "Toward multilevel textual requirements traceability using model-driven engineering and information retrieval," in *International Model-Driven Requirements Engineering Workshop (MoDRE)*, Chicago, IL, USA, September 2012, pp. 29–38.

[19] C. Robinson-Mallett, "An approach on integrating models and textual specifications," in *International Model-Driven Requirements Engineering Workshop (MoDRE)*, Chicago, IL, USA, September 2012, pp. 92–96.

[20] S. Feldmann, S. J. I. Herzig, K. Kernschmidt, T. Wolfenstetter, D. Kammerl, A. Qamar, U. Lindemann, H. Krcmar, C. J. J. Paredis, and B. Vogel-Heuser, "Towards effective management of inconsistencies in model-based engineering of automated production systems," in *IFAC Symposium on Information Control Problems in Manufacturing (IFAC)*, Ottawa, Canada, May 2015, pp. 916–923.

[21] A. K. Chopra, F. Dalpiaz, F. B. Aydemir, P. Giorgini, J. Mylopoulos, and M. P. Singh, "Protos: foundations for engineering innovative sociotechnical systems," in *International Requirements Engineering Conference (RE)*, Karlskrona, Sweden, August 2014, pp. 53–62.

[22] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *International Symposium on Requirements Engineering (RE)*, Toronto, Canada, August 2001, pp. 249–262.

[23] S. Teufl, W. Böhm, and R. Pinger, "Understanding and closing the gap between requirements on system and subsystem level," in *International Model-Driven Requirements Engineering Workshop (MoDRE)*, Karlskrona, Sweden, August 2014, pp. 77–86.

[24] A. K. Chopra and M. P. Singh, "From social machines to social protocols: software engineering foundations for sociotechnical systems," in *International Conference on World Wide Web (WWW)*, Montreal, Canada, April 2016, pp. 903–914.

[25] N. Niu, T. Bhowmik, H. Liu, and Z. Niu, "Traceability-enabled refactoring for managing just-in-time requirements," in *International Requirements Engineering Conference (RE)*, Karlskrona, Sweden, August 2014, pp. 133–142.

[26] P. R. Telang and M. P. Singh, "Specifying and verifying cross-organizational business models: An agent-oriented approach," *IEEE Transactions on Services Computing*, vol. 5, no. 3, pp. 305–318, 2012.