Modeling and verifying behavioral constraints for automation systems

Benjamin Brandenbourger
CAX-Service GmbH
Siedlungsweg 14, 85258 Weichs, Germany
brandenbourger@cax-service.de

Milan Vathoopan, Alois Zoitl
fortiss GmbH
Guerickestr. 25, 80805 Munich, Germany
{vathoopan, zoitl}@fortiss.org

***Abstract*—A technical plant behavior using a functional description forms the basis for common understanding of different disciplines and is used for the draft of hardware and software. Correlations between specific functions of the automation components and system can be allowed or denied in the behavior model describing the tended reaction. This contribution proposes an approach for modeling and verifying formal behavioral constraints of automation systems. The approach is realized either with finite state machines or with regular expressions. Furthermore, this work presents a methodology to convert verbalized constraints into finite state machines or regular expressions. The approach is evaluated by means of a pick & place unit.**

I. Introduction

The manufacturing industry currently experiences drastic changes shifting mass production towards customer-specific products [1]. With the increasing, dynamic, and variant nature of customer requirements [2], companies are forced to react with fast refitting machines. Cyber-physical systems with functional interfaces are one response to this requirement [3].

However, the Plug&Produce approach does not implicitly support equipment protection. Malfunction of an automation system causing unintended behavior or even physical damage due to faulty programming can occur at any time. Yet using the system's functionalities together with a behavior model such as presented in [4] opens up new possibilities. Specific sequences of functionalities can be permitted or blocked. As a consequence, modeling system- or application-specific constraints retains the monitored system in a protected, defined status and prevents unintended behavior.

The practical application of the approach is diverse, as it is independently applicable: One use case is continuously checking the activity of the monitored system during runtime and detecting program sequences leading to a harmful behavior. This use case could find approval, for example, in the education sector and is comparable to the restriction of a robot's working space in order to avoid physical crashes with the periphery. Another application is planing and pre-checking program sequences before a new code is downloaded on the automation system. Furthermore, the approach supports equipment protection for a common resource which is accessed by several parallel running application sequences. The risk of a common resource running into a dead-lock is also eliminated when using the severest level of restriction. Additionally, the behavior of spatially distributed component system can be modeled and checked.

In this work, an approach for modeling and verifying complex sequences of operations of automation systems is presented. The approach supports simultaneous call of functions which is required e.g. for interpolation.

The remainder of this paper is structured as follows: Section II gives an overview of available work in the field of modeling constraints in behavior models. An application example in form of a pick & place unit is presented in Section III. Section IV describes the theoretical approach for modeling behavioral constraints. Different levels of restriction of the behavior model, depending on the envisioned level of severeness, are introduced. The theoretical approach is analyzed on the one hand with finite state machines and on the other hand with regular expressions. In Section V the evaluation of the approach is presented. Finally, Section VI concludes the paper.

II. Related Work

Hanisch et al. [5] put forward a method for formal synthesis of discrete supervisory controllers which is based on an automaton model of the plant. The model describes the plant's uncontrolled behavior and a specification of states or state sequences of the plant which must be prevented by the controller. Moreover, Hanisch presents together with Vyatkin et al. [6] a new framework for design and validation of industrial automation systems based on systematic application of formal methods. Preusse et al. [7] propose a graphic-based method which enables the user to create a formal specification of behavior description with the use of symbolic timing diagrams. The authors of this work follow a functional approach for controlling the plant's actuators.

The same functional approach is also taken up again by Brandenbourger et al. [8] by creating an integrated mechatronic model using a metamodel implemented in AutomationML. The system's functions are encapsulated in so-called skills which start a procedure when triggered. Together with a 3D-model, the same authors introduce in [4] a behavior model of automation components using cross-domain interdependencies.

Ramadge et al. [9] follow in their work a similar approach as presented in this contribution. However, the derivation of the system's behavioral states is not taken into account. Furthermore, different interdependent levels of application, such as arbitrary or reasonable aaplications, are not considered.

De Schutter et al. [10] present in their work an approach focusing on constraint-based programming of robot systems that can also be applied on plant automation. The vector-valued functions are comparable to the functional approach of the presented work. However, the geometry uncertainty is not categorized into different interdependent levels of application.

An automatic synthesis of control strategies for complex dynamical systems is presented in the work of Wongpiromsarn et al. [11]. The resulting system is used for planning and continuously checking the correct behaviors. The authors express the desired properties in the language of linear temporal logic which is used to construct a finite state automaton. Hence, the envisioned state automaton is build up from the scratch whereas the approach followed in the presented work consists of erasing undesirable transitions out of a total state automaton.

An alternative approach concentrating on spatially distributed component systems is proposed by Blech et al. [12] with a framework for modeling and checking the behavior. Vogel-Heuser et al. [13] present the core concepts for PLC-statecharts - an adaptation of UML-statecharts - which can be used as a visual programming language for PLCs. The defined formal behavioral semantics sets the basis for an automatic transformation of PLC-statecharts into timed automata which are analyzed by a model-checker. However, these approaches mainly concentrate on the model itself or its possible application methodology. The evolution of the model and a real application scenario considering the model are not in the scope of these works. Besides, the combination of regular expressions with a functional approach in the automation domain has not yet been examined.

III. Application Example

Figure 1 depicts a pick & place unit with an initial position consisting of two retracted linear drives and an open gripper. The pick & place unit is used for transferring workpieces from one tray to the other. Following the functional approach presented in [8], the following steps need to be performed for transferring workpieces from tray 2 to tray 1:

1) call skill *Right (Ri)* (extend X-axis)
2) call skill *Down (D)* (extend Z-axis)
3) call skill *Grip (G)* (close gripper)
4) call skill *Up (U)* (retract Z-axis)
5) call skill *Left (L)* (retract X-axis)
6) call skill *Down (D)* (extend Z-axis)
7) call skill *Release (Re)* (open gripper)
8) call skill *Up (U)* (retract Z-axis)

After this series of steps, the pick & place unit resides again in its initial configuration. Furthermore, the first 5 steps can be grouped to the composed skill called "Pick" and the last 3 steps to the composed skill called "Place". The composition of the 8 steps can be grouped to the composed skill called "Pick&Place".

In this specific setup, two constraints are verbalized to avoid a malfunction of the system:

1) "Do not hit the trays."

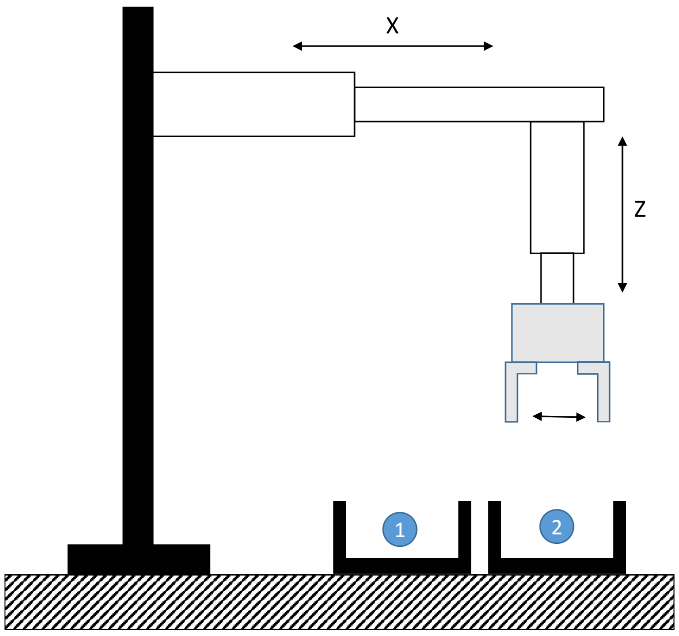

Fig. 1. Application example: pick & place unit with two trays

2) "Do not release a workpiece while transporting."

In order to convert a verbalized constraint to the offered functionalities of the system and its components, the system topology needs to be taken into account. The first constraint is therefore transformed to the physical restriction that no X-movement should be performed while the Z-axis is down. Using the system's skills, the first constraint is restricting any skill-call of the X-axis (skills *Left* or *Right*) after calling the skill *Down* and before having called the skill *Up*. Analog to the first constraint, the second verbalized constraint results in restricting the skill-call *Release* after having called the skill *Up* and before having called the skill *Down*.

IV. Approach

In this work two approaches for modeling behavioral constraints of automation systems are presented. The first approach concentrates on modeling the constraints within the behavior model presented in [4] by removing transitions. The other approach uses regular expressions for describing a valid skill order. A skill order (SO) is a sequence of skill-calls. Both approaches are explained and evaluated by means of the application example.

Skill orders are classified in both approaches in different levels such as arbitrary SO, valid SO, and reasonable SO. The goal of both presented approaches is gaining a model describing a valid SO. This valid SO prevents application- and system-specific malfunctions leading to physical crashes. Both the valid and the reasonable SO are system-specific, as they depend on the system topology.

In order to gain the different levels of SO, the following steps are followed by both approaches:

1) **Modeling an arbitrary skill order** - An arbitrary skill order is a model containing all possible series of skill-calls of the considered automation system. Malfunctions of the system can occur when using this model. Each skill can be called after any other skill and multiple skills can be called simultaneously.
2) **Identifying and modeling constraints** - System-specific constraints must be identified in order to be modeled in the valid SO. The verbalized constraints must be formalized depending on the applied approach. In order to convert a verbalized constraint to the offered functionalities of the system and its components, the system topology needs to be taken into account.
   One approach for the formalization of a constraint consists of the following steps applied on the application example:
   a) Formulate the constraint in a verbalized phrase (e.g. ”Do not hit the trays”)
   b) Formulate the converse (e.g. ”Hit the trays”)
   c) Deduce out of the plant topology the components involved to fulfill the converse (e.g. Z-axis and X-axis)
   d) Formulate the expression to fulfill step b) using the components identified in step c) (e.g. ”Move X-axis while Z-axis is down”)
   e) Formulate the same expression using the skills of the components involved (e.g. ”Call skill *Left* or *Right* after calling skill *Down* and before calling skill *Up*”)
   f) Inverse the statement for getting the constraint from step 1 (e.g. ”Do not call skill *Left* or *Right* after calling skill *Down* and before calling skill *Up*”)
   g) Formalize the expression (e.g. $Down \prec \neg(Left|Right) \prec Up$)

   The critical skill-call is the skill which triggers a constraint and is, in this specific case, the skill *Down*.
   There are several other approaches for formalizing verbal constraints which are out of scope of this paper.
3) **Modeling a valid skill order** - A valid SO is an arbitrary SO including system-specific constraints. The order in which skills are allowed to be called, for creating valid application sequences, is derivable from the valid SO while taking the constraints into account. Crucial malfunctions of the system are prevented by modeling constraints in the valid SO. Therefore, this SO severeness is the intentioned level for industrial use. Preconditions for a valid SO are:
   - The call of a skill can be repeated.
   - By recalling a skill, no malfunction occurs (e.g. recall skill *Grip* with no physical impact; recall composed skill *Pick* and the pick-procedure is performed again).
   - If a skill should not be recalled, this restriction has to be modeled as a constraint.

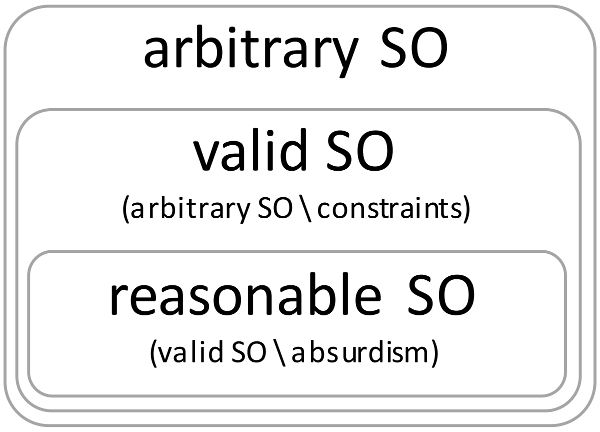

Fig. 2. Relation between different levels of skill orders (SO)

4) **Modeling a reasonable skill order** - A reasonable SO emerges when removing system-specific absurdism of a valid SO. Therefore, parts of the valid SO corresponding to skill orders which are not system-relevant are eliminated. This level of SO is used for optimization or increased efficiency of a monitored automation system. Examples for absurdism are in most cases skill recalls and loops with counter skills. The creation of a reasonable SO is highly system-specific and requires knowledge of the envisioned applications on the automation system.

Figure 2 depicts the relation between the three different SO levels.

A. Approach by modeling constraints in a finite state machine

In [4] a behavior model for automation components and systems using a functional approach was presented. A possible representation of the behavior model is a finite state machine (FSM) consisting of states and transitions. The states represent all possible physical states of an automation component or system. The transitions between the states represent 1 to n (parallel) skill-calls. In this way, simultaneous skill-calls of different automation components are supported and permit, for example, interpolation.

By applying the four steps of the approach presented in Section IV on this behavior model, the different levels of SO are gained.

1) **Modeling arbitrary skill order in FSM** - The arbitrary SO in the behavior model represented as a FSM consists of any (simultaneous) call of a skill offered by one (or more) automation components of the system. This leads to $\prod_{k=1}^{n} s_k$ states, where *n* is the number of automation components in the considered automation system and *s* is the number of skills offered by the automation component *k*.
   As each state can be reached by any other state, each skill can be called after any other skill leading to s^2 transitions where s is the number of states ($s * (s - 1)$

transitions between all the states $+s$ transitions for recalls of the same skill). This modeling step leads to an exponential and therefore unmanageable amount of cross relations. Therefore, Harel statecharts can be used here. This diagram type allows the modeling of superstates which are hierarchically nested states resulting in hierarchical state machines (HSM). In our approach each superstate of an HSM is a component involved in the application example.

2) **Modeling constraints in FSM** - The verbalized constraints of the system must be identified, formalized, and correlated to the corresponding skill-calls. Skill-calls are modeled as transitions in the FSM. Therefore, a constraint prevents the traversing of one or more specific transitions by deleting them. If the obstructed transition is part of a simultaneous skill-call, the transition of the simultaneous skill-call must be deleted.
3) **Modeling valid skill orders in FSM** - Crucial malfunctions of the system are avoided by preventing specific transitions. Therefore, theses specific transitions are eliminated resulting in a behavioral model which supports only valid skill orders.
4) **Modeling reasonable skill orders in FSM** - A reasonable behavior model emerges by removing system-specific absurdism of the valid behavior model. Therefore, further transitions of the valid FSM, such as most of the skill recalls, are deleted. The creation of a reasonable FSM is highly system-specific and requires knowledge of the envisioned applications on the automation system. With this knowledge, further transitions of the valid FSM are eliminated until a system-specific reasonable FSM emerges.

B. Approach by modeling constraints with regular expressions

The second approach for modeling behavioral constraints of automation systems uses regular expressions. A regular expression is a sequence of elements defining a specific search pattern. The search pattern is modeled as a grammar γ, an alphabet Σ contains all considered elements, and a sentence is a series of elements. In this approach a grammar corresponds to all skill sequences which are accepted by the envisioned SO level. All skill orders modeled by constraints are implicitly excluded by the grammar. The alphabet is the sum of all the skills offered by the considered automation system. A sentence is any skill sequence which needs to be checked by the grammar. The goal of this approach is defining an appropriate grammar for the envisioned SO level.

The approach using regular expressions also supports simultaneous skill-calls which are necessary, for example, for interpolation. Simultaneous skill-calls are serialized and then checked by the given grammar. There are $n!$ variants of serialization when n skills are simultaneously called. Each variant needs to be checked by the given grammar.

In addition to the elements of the alphabet, the grammar also contains metacharacters. * symbolizes zero or n occurrences of the preceding element. *?* symbolizes zero or one occurrences of the preceding element.

1) **Modeling arbitrary skill order in regular expressions** - The grammar γ_a for the arbitrary SO is any combination of skills contained in the alphabet Σ.
$$\gamma_a : (Skill_1|Skill_2|...|Skill_n)^*$$
2) **Modeling constraints in regular expressions** - The formalized expression of the constraint resulting from the approach in Section IV must be transformed to a regular expression. The formalized expression will be available in a form such as $Skill_c \prec \neg(Skill_{n1}|Skill_{n2}) \prec Skill_x$. For this purpose, the complete negation of all skills $Skill_n$ is replaced by
$$(\Sigma \setminus Skill_{n1}|...|Skill_{nm})^*.$$
3) **Modeling valid skill orders in regular expressions** - First, the critical skill-call triggering a constraint must be identified. The identified critical skill $Skill_c$ must then be extended in γ_a by the regular expression of the constraint and the leading to the new grammar γ_v for a valid SO:
$$\gamma_v : (Skill_1|Skill_c(\Sigma \setminus Skill_{n1}|...|Skill_{nm})^* Skill_x|...|Skill_n)^*$$
4) **Modeling reasonable skill orders in regular expressions** - The grammar γ_v can be optimized in the last step to a grammar γ_r which supports a reasonable SO. For this purpose, repetitions of undesirable skills and system specific absurdism are excluded from the grammar γ_v. As mentioned before, the creation of a reasonable SO is highly system-specific and requires knowledge of the envisioned applications on the automation system. Therefore, no prescribed strategy for merging γ_v to γ_r can be described here.

In general, it can be observed that the complexity of γ_r is reduced in comparison to γ_v. This is because the system-specific optimization of γ_r leaves out regular expressions which are embedded in γ_v.

V. Evaluation

The approach presented in Section IV is evaluated in this section by means of the application example. Both methods of modeling constraints in a finite state machine or in regular expressions are analyzed.

The correct and intended skill sequence Ri, D, G, U, L, D, Re, U presented in Section III will be accepted whereby the skill sequence L, U, D, G, Ri, Re leading to a crash will be rejected both by the FSM and the regular expression.

A. Evaluation of approach by modeling constraints in a FSM

The application example consists of three automation components with two skills per component. This leads to a FSM with $\prod_{k=1}^{3} 2 = 8$ states and $8^2 = 64$ transitions. The top part of Figure 3 depicts the FSM for the arbitrary SO, whereby bidirectional transitions are grouped for better readability. The

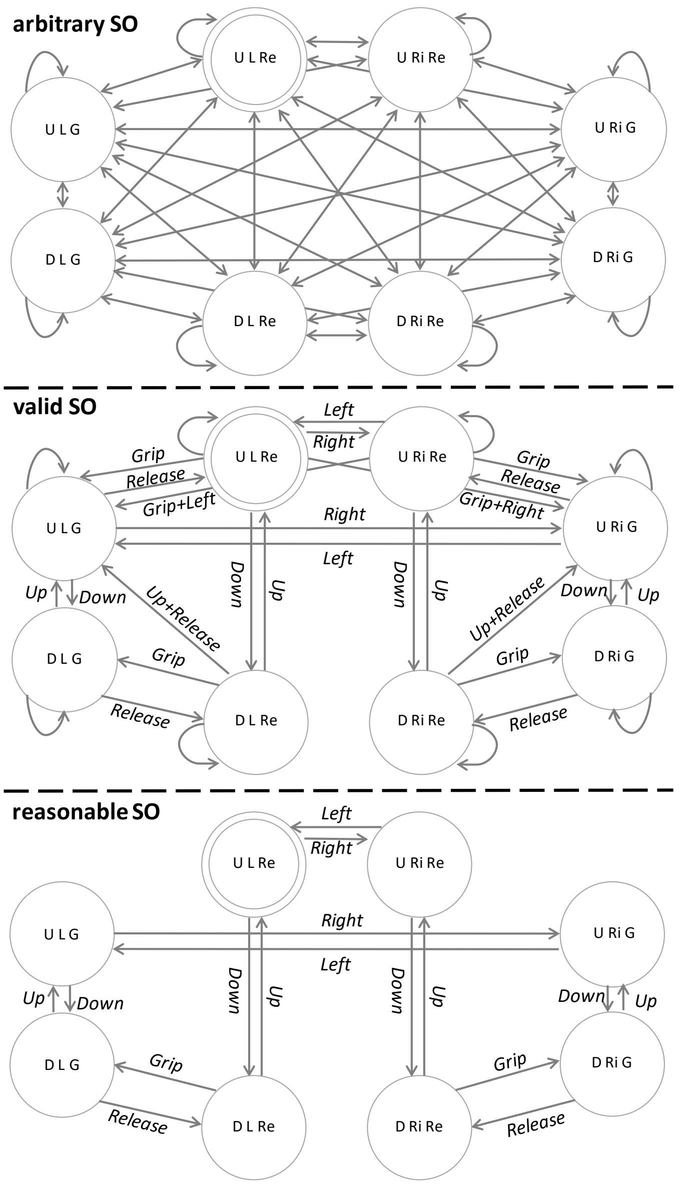

Fig. 3. Evolution of FSM to reasonable SO with two constraints

FSM resides in the same state if one of the skills with no physical impact is recalled. These skill recalls are not explicitly labeled.

Another way of modeling the arbitrary SO of the application example is shown in Figure 4 by using hierarchical state machines (HSM).

The next step consists of modeling the constraints in the FSM. Therefore, specific transitions must be eliminated. In order to identify the specific transitions, first the formalized constraint must be created by applying the presented approach:

1) Formulate the constraint in a verbalized phrase (”Do not release a workpiece while transporting”)
2) Formulate the converse (”Release a workpiece while transporting”)
3) Deduce out of the plant topology the components involved to fulfill the converse (Z-axis and Gripper)
4) Formulate the expression to fulfill step b) using the components identified in step c) (”Open Gripper while Z-axis is up”)
5) Formulate the same expression using the skills of the components involved (”Call skill *Release* after calling skill *Up* and before calling skill *Down*”)
6) Inverse the statement for getting the constraint from step 1 (”Do not call skill *Release* after calling skill *Up* and before calling skill *Down*”)
7) Formalize the expression ($Up \prec \neg Release \prec Down$)

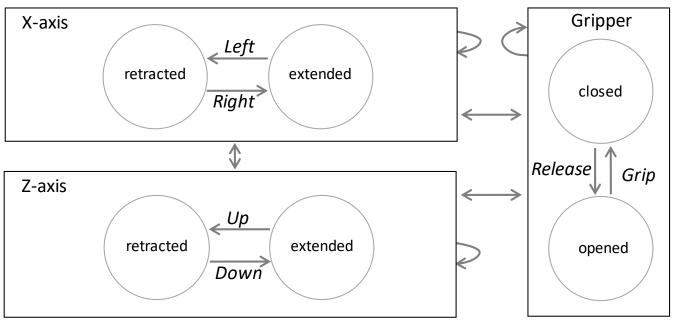

Fig. 4. Arbitrary SO represented as hierarchical state machine (HSM)

The first constraint was already formalized in Section IV: $Down \prec \neg(Left|Right) \prec Up$.

Both formalized constraints need to be mapped to the transitions of the FSM for the arbitrary SO. Thus, in the application example, every transition from *DL** to **R** and every transition from *DR** to **L** must be discarded to fulfill the first constraint. Furthermore, every transition from *U*G* to ***Re* and every transition from *D*G* to *U*Re* must be discarded to fulfill the second constraint. Applying the intended skill sequence Ri, D, G, U, L, D, Re, U on the FSM representing a valid SO is accepted, whereby the skill sequence L, U, D, G, Ri, Re is rejected. The center part of Figure 3 depicts the FSM for the valid SO whereby the transitions are labeled with the corresponding skill-call.

For creating an application- and system-specific reasonable SO, further transitions need to be deleted. Whether a skill order makes sense or not depends highly on the envisioned applications which will be running on the system. Therefore, the reasonable SO can only be created out of the valid SO by an engineer understanding the system and not systematically following a method. The bottom part of Figure 3 depicts the FSM for the reasonable SO.

B. Evaluation of approach by modeling constraints in a regular expression

Evaluating the second approach using regular expressions leads to a set of grammar depending on the SO-level. The grammars are depicted in Figure 5. Prefixes and suffixes depend on the initial position of the system.

The finite alphabet of the application example is:

$$\Sigma = U, D, L, Ri, G, Re$$

Level of severeness	No constraints	Constraint$_1$	Constraint$_2$	Constraint$_1$ + Constraint$_2$
arbitrary SO	$(U\mid D\mid L\mid Ri\mid G\mid Re)^*$	---	---	---
valid SO	$(U\mid D\mid L\mid Ri\mid G\mid Re)^*$	$(U\mid D\underbrace{(D\mid U\mid G\mid Re)^*}_{\Sigma\backslash L\mid Ri}U\mid L\mid Ri\mid G\mid Re)^*$	$(U\underbrace{(U\mid D\mid L\mid Ri)^*}_{\Sigma\backslash G\mid Re}D\mid D\mid L\mid Ri\mid G\mid Re)^*$	$\underbrace{(L\mid U\mid Ri)^*}_{\text{Prefix due to init-pos.}}(D(D\mid G\mid Re)^*U(U\mid L\mid Ri)^*)^*\underbrace{(D(D\mid G\mid Re)^*)^*}_{\text{Suffix}}$
reasonable SO	$(U\mid D\mid L\mid Ri\mid G\mid Re)^*$	$(L\mid Ri\mid D(G\mid Re)U)^*$	$\underbrace{Ri^?}_{\text{Prefix due to init-pos.}}D(U(L\mid Ri)D\mid G\mid Re)^*$	$\underbrace{Ri^?}_{\text{Prefix due to init-pos.}}(D(G\mid Re)U(L\mid Ri))^*(D(G\mid Re)^*)^?$

Fig. 5. Evolution of regular expression grammars depending on SO-level and constraints

The grammar γ_a for the arbitrary SO with no constraints consists of any combination of skills contained in the alphabet Σ:

$$\gamma_f : (U|D|L|Ri|G|Re)^*$$

The next step consists in modeling the formalized constraints in regular expressions. The first formalized constraint $Down \prec \neg(Left|Right) \prec Up$ is transformed to $D(U|D|G|Re)^*U$ with $(U|D|G|Re)^*$ corresponding to $\Sigma\backslash L|Ri$. The second formalized constraint $Up \prec \neg Release \prec Down$ is transformed to $U(U|D|L|Ri)^*U$ with $(U|D|L|Ri)^*$ corresponding to $\Sigma \setminus G|Re$. The critical skill-call for the first constraint is *Down* and for the second constraint the critical skill-call is *Up*. The resulting grammars γ_{v1} and γ_{v2} for the valid SO implicating the first and second constraints are:

$$\gamma_{v1} : (U|D(U|D|G|Re)^*U|L|Ri|G|Re)^*$$
$$\gamma_{v2} : (U(U|D|L|Ri)^*D|D|L|Ri|G|Re)^*$$

Applying the intended skill sequence Ri, D, G, U, L, D, Re, U on γ_{v1} is accepted, whereby the skill sequence L, U, D, G, Ri, Re is rejected.

The grammars γ_r for the reasonable SO are depicted in the last row of Figure 5. The derivation of grammar γ_r from γ_v is system- and application-specific and underlies no recognizable methodology. The concatenation of two grammars emerging from different constraints into one grammar depends on the initial position and can be solved programmatically.

VI. Conclusion

In this paper, we presented an approach to model and verify behavioral constraints for automation systems. The motivation behind this is that manufacturing systems are becoming more complex and foreseeing unintended behavior is becoming difficult. In order to avoid physical crashes of an automation system, the manufacturing system has to be monitored. This is achieved by explicitly modeling constraints in finite state machines or regular expressions which are describing the manufacturing system's intended behavior. The resulting advantages consist of checking at run-time, checking before downloading a new application, coordinating access to common resources, and supporting local and spatially distributed component systems. The approach is applicable in a functional environment and proceeds independently from existing programming sequences. Furthermore, the approach was evaluated on an application example. The results show that the system was able to prevent unintended behavior depending on the level of severeness. In future work, the approach will be extended for finding the strongest level of severeness methodically. Furthermore, the merge of multiple grammars describing different constraints into one grammar needs to be analyzed by means of existing approaches.

References

[1] H. Kühnle, "Post mass production paradigm trajectories," *Journal of Manufacturing Technology Management*, vol. 18, pp. 1022–1037, 2007.

[2] G. Da Silveira, D. Borenstein, and F. S. Fogliatto, "Mass customization: Literature review and research directions," *International journal of production economics*, vol. 72, no. 1, pp. 1–13, 2001.

[3] T. Helbig, S. Henning, and J. Hoos, "Efficient engineering in special purpose machinery through automated control code synthesis based on a functional categorisation," *Machine learning for cyber physical systems*, 2015.

[4] B. Brandenbourger, M. Vathoopan, and A. Zoitl, "Behavior Modeling of Automation Components using cross-domain Interdependencies," *Int. Conf. Emerging Technologies & Factory Automation (ETFA)*, 2016.

[5] H.-M. Hanisch and S. Kowalewski, "Algebraic Synthesis and Verification of Discrete Supervisory Controllers for Forbidden Path Specifications," *4th international Conference on Computer Integrated Manufacturing and Automation Technology (CIMAT)*, 1994.

[6] V. Vyatkin, H.-M. Hanisch, C. Pang, and C.-H. Yang, "Closed-Loop Modeling in Future Automation System Engineering and Validation," *Int. Conf. on Systems, Man, and Cybernetics (SMC)*, vol. 39, no. 1, pp. 17–28, 2008.

[7] S. Preusse and H.-M. Hanisch, "Specification and verification of technical plant behavior with symbolic timing diagrams," *3rd international Design and Test Workshop*, 2008.

[8] B. Brandenbourger, M. Vathoopan, and A. Zoitl, "Engineering of Automation Systems using a Metamodel implemented in AutomationML," *Int. Conf. on Industrial Informatics (INDIN)*, 2016.

[9] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *Society for Industrial and Applied Mathematics*, 1987.

[10] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decr, R. Smits, E. Aertbelin, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *The International Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, 2007.

[11] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning for dynamical systems," *Conference on Decision and Control at the 28th Chinese Control Conference (CDC/CCC)*, 2009.

[12] J. O. Blech and H. Schmidt, "Towards Modeling and Checking the Spatial and Interaction Behavior of Widely Distributed Systems," *Improving Systems and Software Engineering Conference (ISSEC)*, 2013.

[13] D. Witsch and B. Vogel-Heuser, "PLC-Statecharts: An Approach to Integrate UML-Statecharts in Open-Loop Control Engineering Aspects on Behavioral Semantics and Model-Checking," *Int. Federation of Automatec Control (IFAC)*, vol. 44, no. 1, pp. 7866–7872, 2011.