Co-Evolution and Reuse of Automation Control and Simulation Software

Identification and Definition of Modification Actions and Strategies

Christoph Legat*, *Member, IEEE*, Frank Steden‡, Stefan Feldmann*, *Student Member, IEEE*,
Michael Weyrich†, Birgit Vogel-Heuser*, *Senior Member, IEEE*
*Institute of Automation and Information Systems, Technische Universität München, Germany
E-mail: {legat | feldmann | vogel-heuser}@ais.mw.tum.de
‡Chair of Automated Manufacturing, University of Siegen, Germany
E-mail: frank.steden@uni-siegen.de
†Institute of Automation and Software Engineering, University of Stuttgart, Germany
E-mail: michael.weyrich@ias.uni-stuttgart.de

***Abstract*—Industrial plants are multi-disciplinary systems that are operated for multiple decades. Changes in these systems are consequently indispensable, making appropriate mechanisms for managing co-evolution of engineering documentation necessary. In this paper, a co-evolution model for control and simulation software is introduced. Typical evolution categories and modification strategies for enabling co-evolution of automation control and simulation software are derived and formally defined. Using description logics, the identification of these complex modification strategies based on atomic modification actions is made possible.**

I. Introduction

As industrial plants are typically operated between 15 and 30 years, industrial companies are forced to evolve their plants due to several conditions, e.g. new technological developments or market requests [1]. Different disciplines, e.g. mechanical, electrical and software engineering, are involved cooperatively in this process; hence, these disciplines need to be considered during engineering. One means to support this process is the application of simulation software: parts of the mechanical and electrical design can be simulated and, thus, aspects such as virtual commissioning of industrial plants can be supported. Nevertheless, changes within industrial plants occur frequently with diverging evolution cycles of involved disciplines [2]. Although first approaches are currently being developed for managing the co-evolution of these disciplines, co-evolutionary aspects of simulation and automation control software have not been sufficiently investigated yet. Hence, in this paper, an abstract co-evolution model for simulation and automation control software is introduced. Based on typical evolution categories, complex modification strategies are defined based on atomic modification actions in order to support co-evolution of automation control and simulation software.

The remainder of the paper is structured as follows: In the next section, the term *evolution* is specified in detail and related work to ease evolution in automation control and simulation software is analyzed. In section III, an abstract model on co-evolution of automation control and simulation software is introduced. The results of a case study on evolution resulting in basic evolution categories are presented in detail in section IV. From these evolution categories, modification strategies are derived and formalized in section V. A conclusion and an outlook on future work are given in section VI.

II. Related work

Evolution of industrial plants involves multiple terms and approaches that are currently being investigated, e.g. *changeability* and *reconfigurability*. A definition of *changeability* is given in [3]. Therein, changeability is defined as "characteristics to accomplish early and foresighted adjustments of the factory's structure and processes". In contrast, *reconfigurability* is related to the "ability of a manufacturing or assembly system to switch with minimal effort and delay to a particular family of work pieces or subassemblies through the addition or removal of functional elements" [3]. Thus, the term *changeability* refers to anticipated changes whereas *reconfigurability* aims at reacting to unanticipated changes and, hence, to the ability of easing a system's evolution [4]. *Dynamic reconfigurability* refers to adjusting (automation control) software during a system's operation [5], [6]. Moreover, *plug and play control* refers to the adjustment of control software during operation in order to address changing hardware [7]. As all these terms refer to evolving aspects of a manufacturing system, they are henceforth considered as *evolution*.

Evolution of automation control and simulation software is a highly addressed research topic. In [8] for example, an approach on adjusting close-loop control in a plug and play manner is proposed. Sünder et al. [9] propose an approach to model and verify the adaptation of automation control software during operation. Nevertheless, one key aspect to enable control software's evolution is its modularity and, enabled by a modular software structure, its reuse [10]. By providing a modular and reusable software structure, evolution may be simplified as systems' parts can easily be managed in central libraries by applying change actions to these parts. In the remainder of this section, research on easing the evolution of automation control and simulation software is discussed.

A. Related work on evolution of automation control software

For easing evolution of automation control software, *modularity and reuse*, *design patterns* and appropriate *software* *architectures* provide suitable means. These aspects are discussed in detail in the following.

1) Modularity and reuse: Despite large research efforts within the area of modularizing industrial plants, a modularization considering all disciplines involved in the engineering process, e.g. mechanical engineering, electrical/ electronic engineering and software engineering, is not yet state of the art. Especially choosing and selecting mechatronic units [11] increases the complexity of the system and the integration of the system's parts to a holistic system [12]. Hence, "copy, paste and modify" is often used for reusing automation control software [13] by applying change actions, e.g. add, delete and modify [14]. The reasons for these challenges are manifold: On the one hand, Jazdi et al. [10] identify conflicts of interest while choosing the correct module granularity, as fine-grained modules increase reuse and flexibility, whereas coarse-grained modules improve efficiency and robustness in module application. To make things worse, choosing an appropriate granularity depends on the engineering phase of the project; hence, hierarchical models are necessary [13]. On the other hand, the modules' size affects reuse capabilities, as numerous small units make choosing necessary modules and their combination difficult, whereas few large units hamper the modules' flexibility [10]. Respective modeling notations for improving module reuse are currently being applied in automation control software engineering, e.g. UML for software systems [15], [16] and SysML for a holistic view on the system [17], [18], [19], and are being applied for generating the control software [20].

2) Design patterns: In order to increase software modularity, reuse and – by that – quality, design patterns are suitable means. Whereas such design patterns are well-established in the computer science domain, e.g. the "Gang of Four" [21], such design patterns are not yet state of the art in automation control software engineering. A first approach for identifying reusable design patterns was proposed in [22]. A design framework for integrating cognitive functions into intelligent technical systems was investigated in [23]. Recently, design patterns are being identified for defining transformations between UML models and IEC 61131-3-compliant software [20]. Moreover, design patterns for distributed automation systems with regard to – among others – timing requirements was introduced in [24]. Nevertheless, if such design patterns were provided within libraries and, hence, were made reusable for diverse engineering projects, reuse would be enhanced and evolution of automation control software would be simplified.

3) Software architectures: For increasing reuse and reconfigurability of existing functionality, diverse software architectures can be applied. *Service-oriented architectures* (SOA) are established for adaptable business integration [25] and are more and more being applied in automation control software engineering enabling to reuse automation processes by encapsulating them as services [26]. Moreover, semantic technologies are applied to semantically describe such services for orchestrating them to a specific software application [27], [28]. By combining these techniques with modeling languages, e.g. BPMN [28] or SysML [29], changes to the software system are simplified. Nevertheless, these SOA techniques are not yet applied in industry and lack in combination with further aspects of the system, e.g. mechanical or electrical parts. *Multi-agent systems* (MAS) were studied intensely within research for intelligent reconfiguration and self-adaptation of industrial plants, cf. [30]. They are applied for reconfiguring IEC 61131-3 [31]-based control systems to handle module breakdowns in inner logistic systems [6], for dynamically reconfiguring logistic systems [32] or for identifying appropriate reconfiguration strategies [33], [34]. Furthermore, a new standard, i.e. *IEC 61499* [35], was introduced in order to address modularity issues regarding deployment of automation functions. Thereby, reconfiguration of manufacturing systems is eased [32]. However, although IEC 61499 runtimes on state of the art controllers exist [36], "IEC 61499 has a long way in order to be seriously considered by the industry" [37].

B. Related work on evolution of simulation software

Simulation models are used for testing and virtual commissioning in industrial plant engineering. With simulation models, the evolution concepts of an industrial plant can be evaluated [38]. It is possible to detect mechanical design errors as well as failures in the automation control software program to avoid collisions of parts of the plant. Furthermore, the control software can be pre-optimized before real commissioning. Hence, the down time of the plant can be reduced significantly.

The simulation model is a functional performance model of the plant often supported with a visualization to ease the analysis of the simulation experiment [39]. The simulation model comprises the functional behavior of the mechanical parts of the plant as well as of actuators and sensors. The level of detail of the functional model (time-based vs. detailed physical model) depends on the simulation's purpose. The modeling effort increases with a higher level of detail: the more necessary details, the more complex and time-consuming the modeling process. A time-based model often meets the requirements with an adequate *cost : benefit* ratio [40]. For the simulation experiment, the simulation model is linked to the control program via the PLC's inputs and outputs (I/Os) in a hardware- or software-in-the-loop simulation.

To evaluate the industrial plant during its evolution, a new simulation model has to be generated or an existing model has to be modified. The manual modeling effort carried out by simulation experts can be time-consuming and may cause large costs. Hence, the challenge is to reduce the modeling effort in simulation projects and to ease and accelerate the creation and modification of a simulation model [39]. To reduce the modeling effort, many approaches show the possibilities for an automated creation of a simulation model. In these approaches, a standardized description language is used for the transformation of the required plant data into a functional simulation model [41]. In case of complete data, the effort for the creation of the necessary simulation model is reduced, but the implementation of these methods for automated model creation is very complex. With incomplete data, the model has to be completed manually by a simulation expert.

The efficiency and economics in simulation projects may also be increased by reusing simulation (sub)models in the plant life cycle [43]. The requirements on engineering systems for processing reusable models have already been analyzed; the VDI standard 3633 recommends the complete or at least partial reuse of simulation models [13], [44]. First approaches demonstrate the economical use of simulation modules [45]. Based on their smaller functional extent, the modules can be reused and modified more easily. In [46] a methodology for

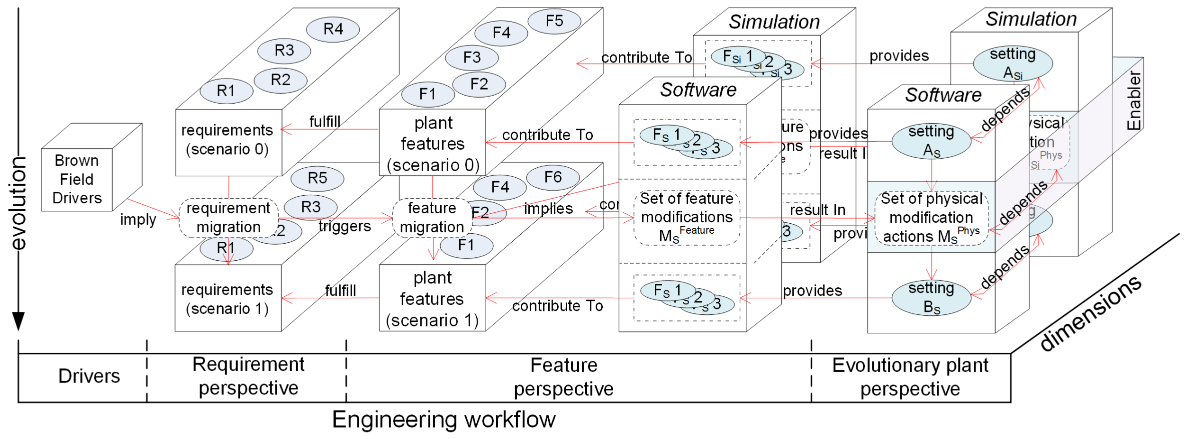


Fig. 1. Co-evolution model of simulation and automation software (based on [42])

the identification and implementation of simulation modules is presented. Depending on the level of detail and the simulation purposes, the simulation can be used in individual engineering steps. In [47] the influencing factors for reuse of simulation modules are identified. In addition, reuse is classified into direct and indirect reuse. The indirectly reusable modules have to be modified. To perform these changes, modification strategies from software engineering are adopted.

C. *Conclusions*

In a nutshell, various approaches address reuse, changeability and reconfigurability of automation control and simulation software. Drivers for evolution are manifold [1], and basically result in changes of requirements on the industrial plant [42]. To make things worse, such changes in requirements not only result in changes in single disciplines, e.g. solely simulation or solely automation control software, but trigger a co-evolution of multiple disciplines. Hence, complex modification strategies, which enable to define modification actions to be applied to the models [14], would provide a suitable means for simplifying automation control and simulation software co-evolution. However, until now, such modification strategies are not applied for automation control and simulation software development. Therefore, an analytic co-evolution model would significantly contribute to the domain of automation control and simulation software engineering.

III. Extended Plant Evolution Model

For investigating simulation and control software co-evolution, a co-evolution model is proposed in detail in this section. Firstly, the plant evolution model proposed in [42] providing the base for the model is described briefly. Secondly, based on an investigation of the interaction between simulation and control software, this model is further developed in two ways: (i) the role of machine simulation within this framework is discussed and (ii) applications of the resulting model for automation control software and simulation are described.

A. *Plant evolution model*

Summarizing the basic idea and terminology of [42], *brown field drivers* (e.g. market dynamics) force *a requirement migration* on the machine or plant (e.g. increased throughput of the machine or plant), cp. Fig. 1, resulting into an adapted set of requirements in a plant. To address the requirement migration, a respective *feature migration* is triggered. Within the context of this paper, we refer to a plant's properties as *features*. Moreover, as engineering is an interdisciplinary task, each discipline of the system refers to specific parts of the system forming the *plant features*. Hence, e.g. *software*, electrical engineering and automation technology (*platform*), as well as mechanical engineering (*context*), form the complete design of a machine or plant. Within this work, we focus on software features representing e.g. the control behavior, and simulation features referring to parts of the system's simulation.

In order to realize the feature migration, respective *modification actions* are necessary, e.g. to add respective software or simulation modules or to correct respective parts by modifying them. Examples for modification actions within the plant's software and simulation are the addition, deletion or modification of pieces of code, functions, function blocks, simulation modules, interfaces, etc. Within these modification actions, each discipline comprises a specific set of possible modification actions, which must in some cases be considered in parallel, i.e. by co-evolving simulation and software. Nevertheless, these modification actions for co-evolving plants have not been investigated sufficiently yet, making the detailed analysis of simulation and control software necessary.

B. *The Role of Simulation within the Evolution Model*

Simulation is used to test control software implementations. In a simulation experiment, the context and parts of the platform of the physical plant form the functional behavior model, which can be connected to a visualization model and to the control software. In Fig. 2 the connection between the control software and the simulation model is depicted.

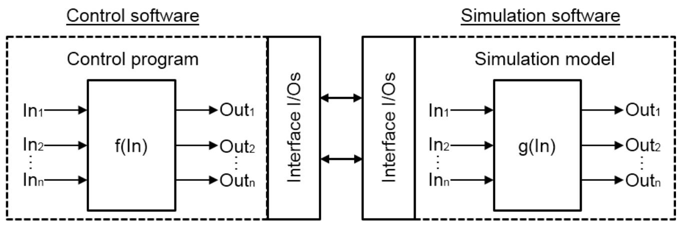


Fig. 2. Interaction of control and simulation software

Both the software and the simulation model are connected via the I/Os of the plant platform. These I/Os are configured in an interface of the control and simulation software. The simulation model is used to generate detailed information in the plant evolution concepts and to validate them. Therefore, new context and platform components have to be integrated into the simulation model and parameters of the model have to be variable in order to validate changes in the plant process.

C. Applications of the model

As can be seen, simulation and software are strongly interconnected. Consequently, simulation and control software evolve in parallel during the plant's evolution. In the following, the application of the evolution model proposed in Fig. 1 for co-evolution of simulation and software is discussed.

In general, there are different applications of this model: (i) a software implementation A_S which was successfully used with the physical plant (i.e. providing the expected software features) can be used to verify whether the simulation A_{Sim} of the settings of platform A_P and context A_C is correct. Furthermore, (ii) before evolving the software setting A_S a correct simulation setting A_{Sim} can be used to derive the contribution of the combination of platform and context settings A_P and A_C features towards the plant features. Finally, (iii) when co-evolving simulation and control software, simulation can be used for virtual commissioning of the control software, i.e. testing the correctness of the implemented software setting A_S. When applying simulation for virtual commissioning, software executed against a simulation must provide the same feature values as expected when evolving it.

When co-evolving both simulation and control software, an eased reconfiguration for software and simulation is required. According to the presented model, increasing the reconfigurability of control software and simulation can be formally defined as minimizing the set of modification actions for software M_S and simulation M_{Sim}. Therefore, in order to minimize these sets of modification actions, a classification of these actions is necessary, which is analyzed in the following.

IV. Classification of change impact for field control software and simulation models

In this section, the dependencies between classes of evolution with respect to their impact on simulation and control software adaptation is presented. The classification into four *Evolution Categories* (ECs) is depicted in Table I.

TABLE I. Identified categories in plant evolution with necessary changes in plant dimensions

Evolution Category	Description of the evolution category	Progress of changes
EC 1	Changes in plant context, simulation model has to be changed in case of possible collisions	C, (Sim, S)
EC 2	Changes in plant context and platform involve changes in software and simulation	C, P, S, Sim
EC 3	Modifications in platform involve changes in Software and Simulation	P, S, Sim
EC 4	Plant process optimizing throughout Software modifications	S

C – Context, P – Platform, S – (Control) Software, Sim – Simulation

Evolution Category EC1 comprises modifications in the context of the plant, e.g. replacing or extending mechanical parts of the plant hence influencing the platform. An example is the increase of the ramp capacity for production parts. Only in case of a possible collision with moving parts, the simulation model has to be modified. To avoid a collision, the context or the software can be modified. The modifications in *EC2* concern the context analogously to *EC1*, but in *EC2* besides the mechanical part of the plant also actuators and sensors are changed or added, e.g. a new conveyor belt for product transportation is installed. Therefore, the platform, the software and the simulation model have to be modified. In *EC3* the context remains unchanged while modifications are performed in the platform. In case of a modification of the platform by replacing functionality with similar functionality using the same interfaces, software and simulation model remain unchained. In case of new interfaces or functionality, the software and simulation model must be changed. The *Evolution Category EC4* is the plant process optimization by modifying the software program. Without changes in context and platform, an existing simulation model can be used for the offline plant optimization by testing the new software.

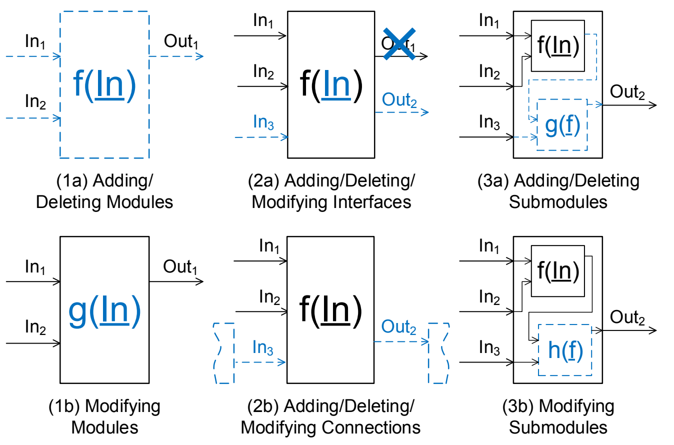
The sequential category order from *EC1* to *EC4* relates to the costs for plant evolution from high to low. Modifications in the context discipline lead to down times even if the plant process is not modified. Changes in *EC4* are limited to changes in the software, but can be managed with a minimum of down time. In *EC4*, the existing simulation model can be used for the software tests with the best *cost : benefit* ratio. The simulation model is only modified in functional parameters, hence, the modeling effort for validating and comparing software evolution concepts is reduced. Consequently, a fast and simple modification of model parameters is demanded. The costs for the necessary changes in *EC1* to *EC3* spread widely and depend on the parts of the simulation model being involved. The interfaces can be modified quickly if the platform changes in *EC3*. Changing the functional behavior of the model is more laborious and the effort increases with a higher level of detail. Therefore, the rapid and simple modification of the model structure is required.

V. Identification of modification strategies for control software and simulation

Former approaches define modification strategies by distinguishing between control software and simulation model changes. These were combined to essential modification strategies, which can be adapted both for control software and simulation. Therefore, the necessary changes in the *Evolution Categories* (section IV) can be managed with a single modification strategy or with a combination of them – e.g. by composing the single-discipline modification strategies to multi-discipline modification strategies as shown in the *Evolution Categories EC1–EC4*. These modification strategies are depicted in Fig. 3 and represent modifications of modules or interfaces within single plant disciplines, e.g. software and simulation.

A. Modification strategies for control and simulation software

Provided that simulation model and control software consist of submodels and subprograms (henceforth referred to as "modules"), the strategies show typical procedures for



In1
In2
f(In)
Out1
(1a) Adding/ Deleting Modules
In1
In2
In3
f(In)
Out1
Out2
(2a) Adding/Deleting/ Modifying Interfaces
In1
In2
In3
f(In)
g(f)
Out2
(3a) Adding/Deleting Submodules
In1
In2
g(In)
Out1
(1b) Modifying Modules
In1
In2
In3
f(In)
Out2
(2b) Adding/Deleting/ Modifying Connections
In1
In2
In3
f(In)
h(f)
Out2
(3b) Modifying Submodules

Fig. 3. Essential modification strategies for control software and simulation model, extended from [47]

modifications. In *Modification Strategy 1a*, an entire module with its functional behavior and the interfaces is implemented or deleted to extend or reduce the functionality of the plant. By that, major context modifications can be realized by adding or deleting software or simulation models as e.g. depicted in *Evolution Categories EC1* and *EC2*.

Modification Strategy 1b involves the modification of a module's behavior, i.e. its specific implementation. Thereby, minor modifications of the module's behavior can be realized as e.g. shown in *EC3*. These changes have to be validated and verified and lead to extra costs for the modification. Depending on the quality of the module documentation, major modifications of the functional behavior, especially of extensive modules, are complex and may lead to failures. In that case, the module should be replaced by an appropriate alternative module (*Modification Strategy 1a*).

By involving the modules' interfaces in modification actions, the interfaces are either modified (*Modification Strategy 2a*) or rewired, i.e. connections between interfaces are changed (*Modification Strategy 2b*). The interface modifications therein are suited for *EC2* and *EC3* as interactions in between simulation and software modules as well as among both are thereby defined through interfaces. An In-Interface added to the module may also represent an internal simulation interface, which is used for the parameterization of the simulation's or software's functional behavior modification in *EC4*. The parameter interface should have a range that must be validated. The modification of the functionality by implementation of submodules through adding or deleting submodules (*Modification Strategy 3a*) or through modifying submodules' behavior (*Modification Strategy 3b*) is a variation of *Modification Strategy 1a* and *1b*. Hence, these strategies are well-suited for major modifications in platform that may also incorporate changes in simulation and software.

B. Abstract automation control and simulation software model

The modification strategies introduced in the previous section represent complex modifications, which are in turn composed of atomic modification actions, e.g. add, delete and modify [14]. For simplifying the parallel and multi-disciplinary evolution of simulation and control software, an automatic classification of sets of atomic modification actions would be significantly helpful. Engineers from different plant disciplines could therefore react more rapidly as these atomic modification actions are provided in a more abstract manner using modification strategies. If e.g. modules are added in the context discipline (*Modification Strategy 1a*), respective modification strategies in parallel plant disciplines, e.g. simulation or software, can be triggered more easily.

In order to reason on the set of changes for both simulation and automation software, an abstract model for defining modification actions is needed, which is introduced in the following. In general, basic modification actions for software (models) exist [14]: model elements can e.g. be added, deleted or modified. These modification actions can be executed on an existing (software) setting to derive a new setting.

An abstract meta model of simulation and control software containing the elements relevant for reasoning on modification actions is shown in Fig. 4, left. Therein, it is distinguished between *Modules*, their *Interfaces* as well as *Connections* between those *Interfaces*. *Modules* represent submodels or subprograms of the software or simulation and are either *Atomic Modules* containing a specific *Implementation*, or *Composite Modules* consisting of further *Modules*. Each module may provide *Interfaces* which are in turn distinguished into *In-Interfaces* and *Out-Interfaces*. As each *Implementation* of an *Atomic Module* may require respective interfaces to be connected, *Connections* point from a *Module*'s *Out-Interface* to another *Module*'s *In-Interface*. Using these meta elements, the simulation and automation software can be described regarding their module and interface structure. Moreover, these main elements inherit from a base *Element*, cp. Fig. 4, right. This base *Element* can further be annotated using respective *modified*, *deleted* or *added* dates defining the time stamp of a modification action. By that, respective modifications of the automation and simulation software can be specified in the model. Hence, all modification actions needed for a specific modification strategy are captured within the model.

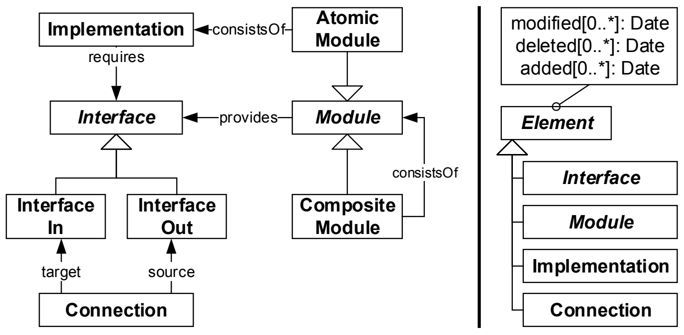


Fig. 4. Abstract software model, excerpt from [48]. Left: Model elements. Right: Modification actions. [48]

C. Reasoning on modification actions

For reasoning on these modification actions defined in the meta model and for identifying respective modification strategies, a formal semantics of the modification strategies is needed. The aim of the classification mechanism is to classify a set of modification actions within an instance of the meta model according to the modification strategies defined in section IV, i.e. whether *an entire module was added, deleted* (*Modification Strategy 1a*) or *modified* (*1b*), whether the *module's interfaces were changed* (*2a*) or *reconnected* (*2b*), or whether its *submodules were changed* (*3a* and *3b*).

In order to capture the formal semantics of these modification strategies, Web Ontology Language (OWL) is used. The meta model elements defined in Fig. 4 are therein represented as OWL concepts and instances of these meta model elements as OWL individuals. Relations between these meta model elements are represented as OWL object properties and annotations as OWL data properties[1]. Therein, as *add*, *delete* and *modify* define the basic change actions, the properties *added*, *deleted* and *modified* are defined as subproperties of *changed*. Hence, besides validating the correctness of the model instances and checking the compatibility of two interfaces or modules [48], two major features are enabled within OWL: (i) by formalizing the modification strategies within additional OWL axioms, modification actions can be classified using the standard reasoning mechanism *Instance Checking* and (ii) by using query mechanisms such as SPARQL Protocol and RDF Query Language (SPARQL), the set of modification strategies applied within a certain time interval can be identified.

Using respective OWL axioms, the modification strategies defined beforehand can be formulated. For a module implementation's modification, it must be distinguished between adding or deleting some containing implementations and modifying implementations of a Module. Hence, respective concepts STRATEGY_1A (axiom 1) and STRATEGY_1B (axiom 2) are defined to be equivalent to the MODULE concept restricted to *contain* some IMPLEMENTATIONS that – in turn – *contain* some *added*, *deleted* or *modified* relations.

STRATEGY_1A **EquivalentTo** MODULE **and** *contains* **some**
(IMPLEMENTATION **and** ((*added* **some** Date) **or**
(*deleted* **some** Date))) (1)

STRATEGY_1B **EquivalentTo** MODULE **and** *contains* **some**
(IMPLEMENTATION **and** (*modified* **some** Date)) (2)

Regarding interfaces, modules can be adapted by changing respective interfaces or by rewiring the interfaces, i.e. changing their connection to other interfaces. Hence, axioms 3 and 4 define the respective modification strategies as concepts STRATEGY_2A and STRATEGY_2B. The latter requires an interface to be a source or target of a respective CONNECTION (indicated by the inverse properties of *source* and *target*), which *contains* some *changed* relations.

STRATEGY_2A **EquivalentTo** MODULE **and** *provides* **some**
(INTERFACE **and** (*changed* **some** Date)) (3)

STRATEGY_2B **EquivalentTo** MODULE **and**
(*provides* **some** (INTERFACE **and** (**inverse**(source) **some**
(CONNECTION **and** *changed* **some** Date))) **or**
provides **some** (INTERFACE **and** (**inverse**(target) **some**
(CONNECTION **and** *changed* **some** Date)))) (4)

Respectively, if submodules are changed, the strategies *3a* and *3b* apply as a variation of strategies *1a* and *1b*. By defining the property *consistsOf* to be transitive, the module hierarchy is flattened within the OWL ontology. Hence, the OWL concepts STRATEGY_3A (axiom 5) and STRATEGY_3B (axiom 6) can be defined as concepts equivalent to a MODULE that *consistsOf* modules being subject to some STRATEGY_1A or STRATEGY_3B, respectively.

STRATEGY_3A **EquivalentTo** MODULE **and**
consistsOf **some** STRATEGY_1A (5)

STRATEGY_3B **EquivalentTo** MODULE **and**
consistsOf **some** STRATEGY_1B (6)

Hence, each OWL individual matching the concept definitions is inferred to be an individual of the respective concept(s). Moreover, these concepts can be formulated within SPARQL queries [49] under OWL entailment regimes [50], e.g. for checking, which modules were modified using a certain modification strategy between two defined time stamps. Hence, using the atomic modification actions *add*, *delete* and *modify*, complex modification strategies can be classified.

By applying these definitions to each discipline of the industrial plant engineering process, e.g. context, platform, simulation and software, the *Evolution Categories EC1–EC4* can be defined formally. Therefore, co-evolution can be supported by defining dependencies between modification strategies in different disciplines and, hence, the user can be supported in identifying modification actions to be undertaken. Another feature provided by the formal model is the ability to formally check model consistency and module compatibility [48].

VI. CONCLUSION AND OUTLOOK

In this paper, an abstract co-evolution model of simulation and automation software is introduced. Based on typical evolution categories, complex modification strategies as composition of atomic modification actions in order to support co-evolution of automation control and simulation software. The modification actions are formally described using description logics axioms. By that, modification strategies can be identified from modification actions applied to the plant model to support their application for different evolution categories. Consequently, using the proposed approach, engineers can be supported in managing the co-evolution of control and simulation software for industrial plant engineering.

Within future work, integrations of the presented classification methodologies into existing tool support is planned. Moreover, the evolution categories and modification strategies will be extended towards more complex application scenarios. Finally, modeling support is aspired, which enables the company-specific modeling of categories and strategies. By that, the evolution categories' and modification strategies' reuse would be increased further.

REFERENCES

[1] E. Westkämper, "Adaptable production structures," in *Manuf. Technol. Mach. Futur.*, A. Dashchenko, Ed. Springer-Verlag, 2003, pp. 87–120.

[2] F. Li, G. Bayrak *et al.*, "Specification of the Requirements to Support Information Technology-Cycles in the Machine and Plant Manufacturing Industry," in *IFAC Symp. Inf. Control Probl. Manuf.*, 2012, pp. 1077–1082.

[3] H.-P. Wiendahl, H. ElMaraghy *et al.*, "Changeable Manufacturing – Classification, Design and Operation," *CIRP Ann. Manuf. Technol.*, vol. 56, no. 2, pp. 783–809, 2007.

[1]As OWL annotations have no semantics and, hence, cannot be processed by a reasoner, data properties are used for capturing the semantics of the modification actions.

[4] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren, "Reconfigurable manufacturing systems: Key to future manufacturing," *J. Intell. Manuf.*, vol. 11, no. 4, pp. 403–419, 2000.

[5] J. D. Walsh, F. Bordeleau, and B. Selic, "Domain analysis of dynamic system reconfiguration," *Softw. Syst. Model.*, vol. 6, no. 4, pp. 355–380, 2007.

[6] P. Vrba and V. Marik, "Capabilities of Dynamic Reconfiguration of Multiagent-Based Industrial Control Systems," *IEEE Trans. Syst. Man, Cybern. – Part A Syst. Humans*, vol. 40, no. 2, pp. 213–223, 2010.

[7] J. Bendtsen, K. Trangbaek, and J. Stoustrup, "Plug-and-Play Control – Modifying Control Systems Online," *IEEE Trans. Control Syst. Technol.*, vol. 21, no. 1, pp. 79–93, 2013.

[8] S. Bodenburg and J. Lunze, "Plug-and-Play control – Theory and implementation," in *IEEE Int. Conf. Ind. Informatics*, 2013, pp. 165–170.

[9] C. Sünder, V. Vyatkin, and A. Zoitl, "Formal Verification of Downtimeless System Evolution in Embedded Automation Controllers," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 1, pp. 1–17, 2013.

[10] N. Jazdi, C. Maga, and P. Goehner, "Reusable Models in Industrial Automation: Experiences in Defining Appropriate Levels of Granularity," in *IFAC World Congr.*, vol. 18, no. 1, 2011, pp. 9145–9150.

[11] A. Lüder, L. H. M. Foehr *et al.*, "Manufacturing system engineering with mechatronical units," in *IEEE Int. Conf. Emerg. Technol. Fact. Autom.*, 2010, pp. 1–8.

[12] M. Weyrich, P. Klein, and F. Steden, "Reuse of modules for mechatronic modeling and evaluation of manufacturing systems in the conceptual design and basic engineering phase," in *IFAC World Congr.*, 2014.

[13] C. R. Maga, N. Jazdi, and P. Gohner, "Requirements on engineering tools for increasing reuse in industrial automation," in *IEEE Int. Conf. Emerg. Technol. Fact. Autom.*, 2011, pp. 1–7.

[14] T. Kehrer, U. Kelter *et al.*, "Understanding model evolution through semantically lifting model differences with SiLift," in *IEEE Int. Conf. Softw. Maint.*, 2012, pp. 638–641.

[15] K. Thramboulidis and G. Frey, "An MDD process for IEC 61131-based industrial automation systems," in *IEEE Int. Conf. Emerg. Technol. Fact. Autom.*, 2011, pp. 1–8.

[16] C. Secchi, M. Bonfe, and C. Fantuzzi, "On the Use of UML for Modeling Mechatronic Systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 4, no. 1, pp. 105–113, 2007.

[17] K. Thramboulidis and G. Frey, "Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation," *J. Softw. Eng. Appl.*, vol. 4, no. 4, pp. 217–226, 2011.

[18] L. Bassi, C. Secchi *et al.*, "A SysML-Based Methodology for Manufacturing Machinery Modeling and Design," *IEEE/ASME Trans. Mechatronics*, vol. 16, no. 6, pp. 1049–1062, 2011.

[19] K. Kernschmidt, P. Klein *et al.*, "Methodology for Identification of adaptive Reusable Modules in automated Production Systems," in *CIRP Des. Conf.*, 2013, pp. 125–135.

[20] C. Fantuzzi, M. Bonfe *et al.*, "A Design Pattern for Translating UML Software Models into IEC 61131-3 Programming Languages," in *IFAC World Congr.*, vol. 18, no. 1, 2011, pp. 9158–9163.

[21] E. Gamma, R. Helm *et al.*, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, US-MA: Addison-Wesley, 1994.

[22] J. Fuchs, S. Feldmann *et al.*, "Identification of Design Patterns for IEC 61131-3 in Machine and Plant Manufacturing," in *IFAC World Congr.*, 2014.

[23] R. Dumitrescu, H. Anacker, and J. Gausemeier, "Design framework for the integration of cognitive functions into intelligent technical systems," *Prod. Eng.*, vol. 7, no. 1, pp. 111–121, 2012.

[24] K. Eckert, A. Fay *et al.*, "Design patterns for distributed automation systems with consideration of non-functional requirements," in *IEEE Int. Conf. Emerg. Technol. Fact. Autom.*, 2012, pp. 1–9.

[25] F. Jammes and H. Smit, "Service-Oriented Paradigms in Industrial Automation," *IEEE Trans. Ind. Informatics*, vol. 1, no. 1, pp. 62–70, 2005.

[26] G. Candido, A. W. Colombo *et al.*, "Service-Oriented Infrastructure to Support the Deployment of Evolvable Production Systems," *IEEE Trans. Ind. Informatics*, vol. 7, no. 4, pp. 759–767, 2011.

[27] J. L. M. Lastra and I. M. Delamer, "Semantic Web Services in Factory Automation: Fundamental Insights and Research Roadmap," *IEEE Trans. Ind. Informatics*, vol. 2, no. 1, pp. 1–11, 2006.

[28] M. Loskyll, J. Schlick *et al.*, "Semantic service discovery and orchestration for manufacturing processes," in *IEEE Conf. Emerg. Technol. Fact. Autom.*, 2011, pp. 1–8.

[29] S. Feldmann, M. Loskyll *et al.*, "Increasing agility in engineering and runtime of automated manufacturing systems," in *IEEE Int. Conf. Ind. Technol.*, 2013, pp. 1303–1308.

[30] R. W. Brennan, P. Vrba *et al.*, "Developments in dynamic and intelligent reconfiguration of industrial automation," *Comput. Ind.*, vol. 59, no. 6, pp. 533–547, 2008.

[31] IEC, "Programmable Logic Controllers – Part 3: Programming Languages. IEC Standard 61131-3," 2009.

[32] W. Lepuschitz, A. Zoitl *et al.*, "Toward Self-Reconfiguration of Manufacturing Systems Using Automation Agents," *IEEE Trans. Syst. Man, Cybern. Part C (Applications Rev.*, vol. 41, no. 1, pp. 52–69, 2011.

[33] P. Leitão and F. Restivo, "ADACOR: A holonic architecture for agile and adaptive manufacturing control," *Comput. Ind.*, vol. 57, no. 2, pp. 121–130, 2006.

[34] Y. Alsafi and V. Vyatkin, "Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing," *Robot. Comput. Integr. Manuf.*, vol. 26, no. 4, pp. 381–391, 2010.

[35] IEC, "Function Blocks. IEC Standard 61499," 2011.

[36] V. Vyatkin, "IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review," *IEEE Trans. Ind. Informatics*, vol. 7, no. 4, pp. 768–781, 2011.

[37] K. Thramboulidis, "IEC 61499 as an Enabler of Distributed and Intelligent Automation: A State-of-the-Art Review – A Different View," *J. Eng.*, pp. 1–9, 2013.

[38] E. Estevez and M. Marcos, "Model-Based Validation of Industrial Control Systems," *IEEE Trans. Ind. Informatics*, vol. 8, no. 2, pp. 302–310, 2012.

[39] R. Wischnewski, J. Rossmann, and O. Stern, "New Methods to Create Variants of 3-D Simulation Models of Manufacturing Systems," in *Int. Conf. Chang. Agil. Reconfigurable Virtual Prod.*, 2012, pp. 74–79.

[40] G. Wünsch and M. F. Zäh, "A new Method for fast Plant Startup," in *Int. Conf. Chang. Agil. Reconfigurable Virtual Prod.*, 2005, pp. 249–255.

[41] M. Barth and A. Fay, "Efficient use of data exchange formats in engineering projects by means of language integrated queries – Engineers LINQ to XML," in *Annu. Conf. IEEE Ind. Electron. Soc.*, 2010, pp. 1335–1340.

[42] C. Legat, J. Folmer, and B. Vogel-Heuser, "Evolution in Industrial Plant Automation: A Case Study," in *Annu. Conf. IEEE Ind. Electron. Soc.*, vol. 68, no. 2020, 2013, pp. 1–10.

[43] G. Mayer and S. Spieckermann, "Life-cycle of simulation models: requirements and case studies in the automotive industry," *J. Simul.*, vol. 4, no. 4, pp. 255–259, 2010.

[44] Verein Deutscher Ingenieure e.V. (VDI), "Simulation of systems in materials handling, logistics and production. VDI Standard 3633," 1996.

[45] U. Schob, "A framework for automating the generation of machine simulation models," in *World Acad. Sci. Eng. Technol.*, 2010, pp. 571–577.

[46] M. Weyrich and F. Steden, "Automated Configuration of a Machine Simulation Based on a Modular Approach," in *CIRP Des. Conf.*, 2013, pp. 603–612.

[47] M. Weyrich and F. Steden, "Simulationsmodule methodisch identifizieren," *atp Ed.*, vol. 55, no. 7-8, pp. 44–52, 2013.

[48] S. Feldmann, K. Kernschmidt, and B. Vogel-Heuser, "Combining a SysML-based modeling approach and semantic technologies for analyzing change influences in manufacturing plant models," in *CIRP Conf. Manuf. Syst.*, 2014.

[49] World Wide Web Consortium (W3C), "SPARQL 1.1 Query Language," 2013. [Online]. Available: http://www.w3.org/TR/sparql11-query/

[50] World Wide Web Consortium (W3C), "SPARQL 1.1 Entailment Regimes," 2013. [Online]. Available: http://www.w3.org/TR/sparql11-entailment/