Transdisciplinary Engineering Methods for Social Innovation of Industry 4.0
M. Peruzzini et al. (Eds.)

doi:10.3233/978-1-61499-898-3-93

Digital Manufacturing and Virtual Commissioning of Intelligent Factories and Industry 4.0 Systems Using Graph-Based Design Languages

Nicolai BEISHEIM[a,1], Markus KIESEL[a] and Stephan RUDOLPH[b]
[a]*Albstadt-Sigmaringen University, Jakobstraße 1, 72458 Albstadt-Ebingen, Germany*
[b]*Institute of Aircarft Design, Universität Stuttgart, Pfaffenwaldring 31, 70569 Stuttgart, Germany*

Abstract. The development process for the new kind of Intelligent Factories and Industry 4.0 systems requires a different approach than before, because the complexity and flexibility of all industrial systems increases. The development process has to ensure that every product is manufactured at the end with an efficient production process. Under these conditions, data management and the use of simulations become more important. Furthermore, the couplings and interrelations between product and means of production play a key role in the areas of digital factory and virtual commissioning. Only by using both simulation methods it will become apparent how the product can be manufactured. But to create a virtual commissioning model, a lot of data is needed from the entire development process. This data is usually generated interdisciplinarily by people from various departments. However, today's form of data management as information transfer is only conditionally suitable for efficiently building up a digital factory as a simulation model or even making a virtual commissioning of a production system. Due to the fact that in the development process a lot of IT systems with interfaces inbetween are used, loss of information is quite common experience. This is where graph-based design languages come in. Using this modeling language approach, the product is completely digitally described and then the means of production are derived from the product properties automatically. This creates a consistent digital product life cycle from the initial product requirements to virtual commissioning.

Keywords. Digital Manufacturing; Virtual Commissioning; Simulation; Design Method; Graph-based Design Language

Introduction

Virtual Commissioning is a key technology for improving the reliability, quality and efficiency of production systems by providing the ability to test, evaluate and improve production systems based on their digital mock-up prior to assembly (see [1], [2]). At present however, mainly large companies use this technology because the skills required to create simulation models are quite extensive and do require a great deal of

[1] Corresponding Author, E-mail: beisheim@hs-albsig.de

effort. Innovative technologies such as Industrie 4.0 or Internet of Things (IoT) will increase flexibility of the production system (see [3], [4], [5]). However, this also leads to an enormous increase in complexity of production systems with respect to the already existing technical and organisational complexity.

In order to maintain their current market share, the use of technologies such as virtual commissioning makes also sense for small and medium-sized companies. However, the greatest effort at present is to create and update the simulation models required for virtual commissioning, and with the increasing information management complexity this problem will become even greater.

One possible solution to this problem is the automated creation of simulation models using graph-based design languages. The use of this design language approach results in a continuous digital product life-cycle from the requirements definition phase to the virtual commissioning of new plants and systems. Many variants of the production process of a product can be simulated by the automatic creation of the models. The best process is determined by evaluating the simulation results. This automation distinguishes the graph-based design language approach from today's common programs for virtual commissioning and the digital factory as CAD-CAE software such as Process Simulate and DELMIA.

1. Graph-based design languages and AutomationML

The methodology described in this paper for a simple and automatic creation of simulation models for Digital Manufacturing and Virtual Commissioning includes two main concepts. This chapter provides a general overview of these concepts.

1.1. Graph-based design languages

Graph-based design languages are used to systematize and automate development processes based on abstract knowledge in order to achieve the best possible result (see e. g.[6], [7]). Graph-based design languages possess the following three aspects:

- Vocabulary
  Vocabulary is part of abstract knowledge. It describes the available components within the design language and their correlation to each other.

- Rules
  The rules form the second part of abstract knowledge and they are basically the blueprint for the design process. The rules can be influenced by a variety of parameters, which may lead to completely different models.

- Compilers
  A compiler is used to create instances of the design described in the vocabulary and rules. The model created by the compiler can then be used as a central model for various other applications.

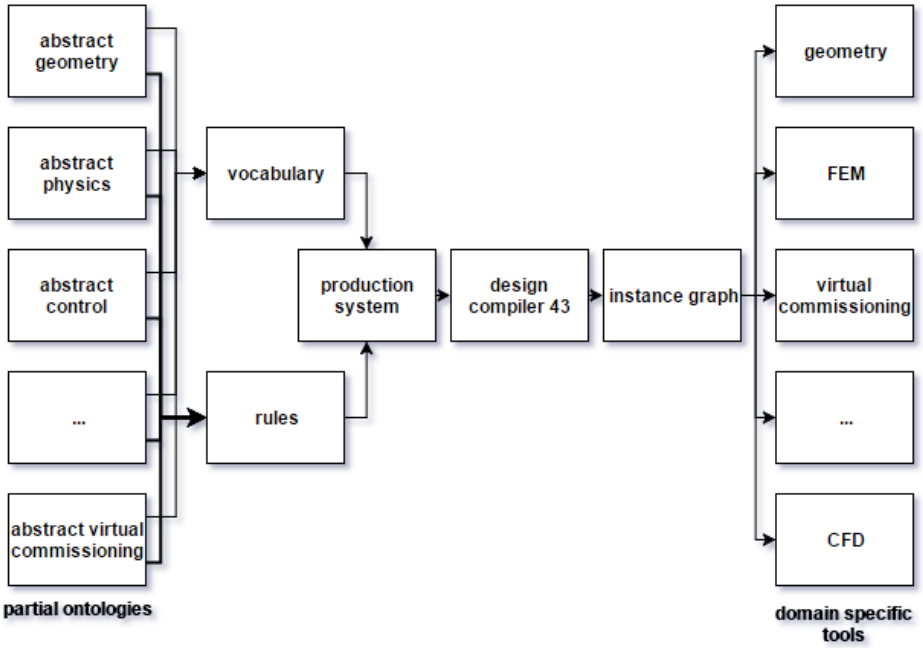

Figure 1. Overview of graph-based design languages (based on [8]).

Graph-based design languages can be used to generate all properties of the target model, e. g. the geometry of a robotic installation and the Robot code required for programming the robotic installation. For virtual commissioning, the generated simulation model is then used together with the controllers for the machines and the robots as interactive simulation. The controllers can be real hardware, e.g. Siemens S7 or software programs such as Beckhoff TwinCAT. Figure 1 provides an overview of the relationships within the graph-based design language process.

The current research project ´Digital Product Lifecycle (DiP)´ uses the Design Compiler 43 of the company IILS mbH [9], a system capable of creating and executing graph-based design languages. This system has already been used in several research and business projects (see e. g. [9], [10]) and has proven its usefulness for the improvement of machine and plant development processes. The aim of the research project is to automatically and optimally implement the development process of an engine hood as a product together with the development of the production line for its manufacture by means of graph-based design languages.

1.2. AutomationML

AutomationML (Automation Markup Language) is a neutral data format for storing and exchanging plant planning data. The data format is based on XML. The aim of using AutomationML is to exchange data in a heterogeneous IT landscape of engineering tools. Areas of application include mechanical design, electrical design, PLC or robot programming. The data exchange format AutomationML is standardized in IEC 62714, whereby both the part "Architecture and general requirements" and the part "Role class library" are already international standards.

AutomationML is expected to become the official standard format for data exchange between IT engineering tools for the development of Intelligent Factories and Industry 4.0 systems, as it offers the possibility to serve a wide range of engineering application areas (see e. g. [11], [12]). It can contain much more information than a CAD exchange format such as JT, STEP or IGES. In order to make AutomationML easily accessible to users, several open standards such as COLLADA and PLCopen XML are used in parallel in their syntax.

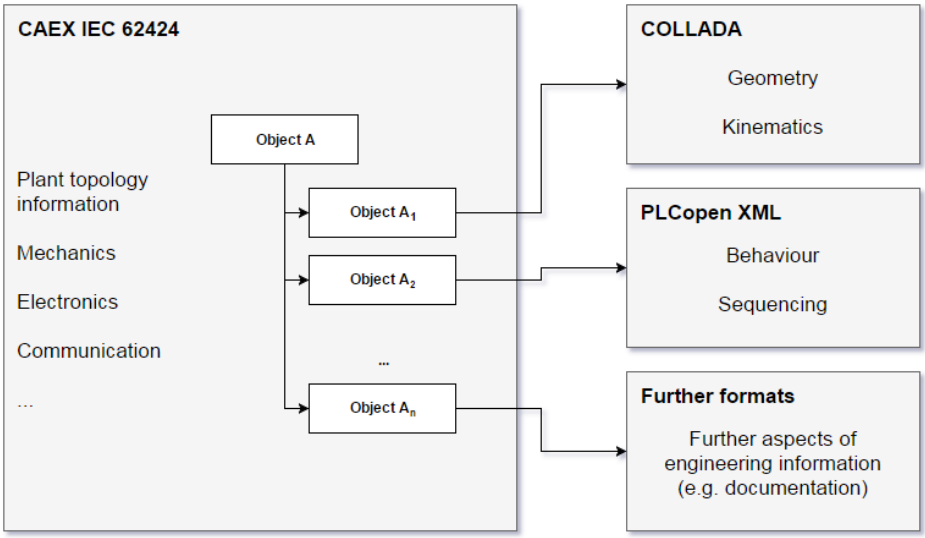

Figure 2. AutomationML Overview (based on [13]).

These open standards used by AutomationML [14] are shown in Figure 2. The AutomationML file itself is based on CAEX format (IEC 6242424) which has been slightly modified to meet the technical requirements. The CAEX format is XML-based and easily extensible by linking to other files.

For the representation of geometry, COLLADA standard is used, which is able to store a geometry as an edge representation, as it is typically used in CAD programs, as well as a triangulated mesh representation. In addition to simple geometry, COLLADA standard can also contain information about kinematics and physics of an object as well as other geometric information.

The use of AutomationML as a data format for virtual commissioning is particularly interesting due to the integration of the PLCopen XML format. These format is based on IEC61131-3, which can be used to store and transfer logic data used by programmable logic controllers for machines or robots.

2. Methodology

This chapter describes the creation process of the AutomationML file for virtual commissioning models with graph-based design languages. This process consists of four steps:

1. (manual) definition of vocabulary
2. (manual) definition of rules

3. (automated) compiling
4. (automated) creation of the AutomationML file

2.1. Definition of the Example

The following subchapters explain the use of graph-based design language using an application example (see [16], [17]). The example is a robot cell consisting of two 6-axis robots. To create a functional model of this cell, the following data and characteristic values of the components must exist:

- CAD data of all components of the cell
- Motion-optimized assembly structure for the robots
- Definition of the robot movement sequence
- Definition of inputs and outputs within the cell

2.2. AutomationML vocabulary

The basis of any graph-based design language is the vocabulary that defines what can be used in the development process. There are different subgroups of the required vocabulary for AutomationML. The most important subsets are:

- CAEX
- PLCopen XML
- COLLADA

In this paper, only the CAEX subset is shown for simplification.

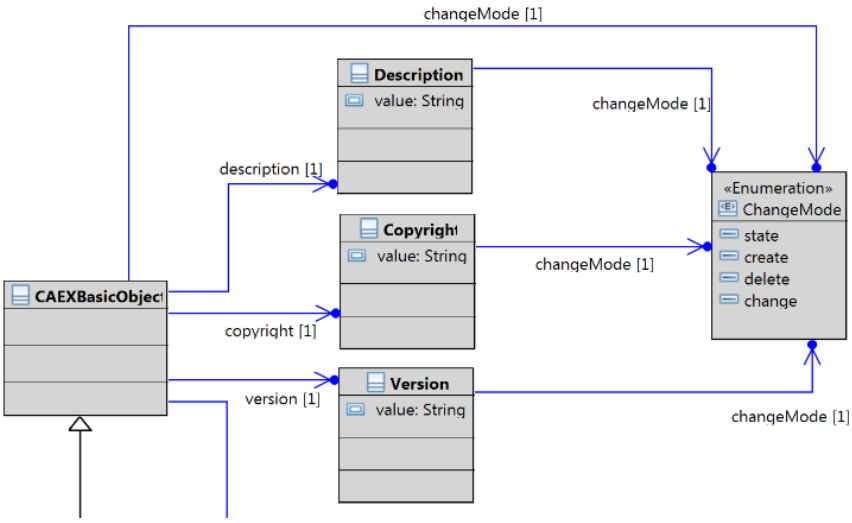

Figure 3. Part of the CAEX Vocabulary.

Figure 3 shows how vocabulary and interdependencies are modeled as UML structure. The mapped CAEXBasicObject is the basis for the other CAEX objects. It has a description, copyright, version, changeMode and (not shown in the picture) a revision. Since all other objects inherit from this object, they also have these attributes. The

entire CAEX structure contains about 30 objects with their dependencies and attributes. The structure shown is based on the CAEX directive IEC 6242424.

The AutomationML standard extends the CAEX structure by four additional objects, which offer the possibility to define a position and rotation in vector space $\mathbb{R}^3$ and to connect external PLCopen XML and COLLADA models with a CAEX object. The AutomationML structure is the basis for creating a technical vocabulary tailored to the specific needs of virtual commissioning.

The robots mentioned in Section 2.1 as an example of application inherit from the InternalElement defined in the CAEX vocabulary on the basis of the UML structure. This allows the robot to be set up as a motion-optimized assembly structure.

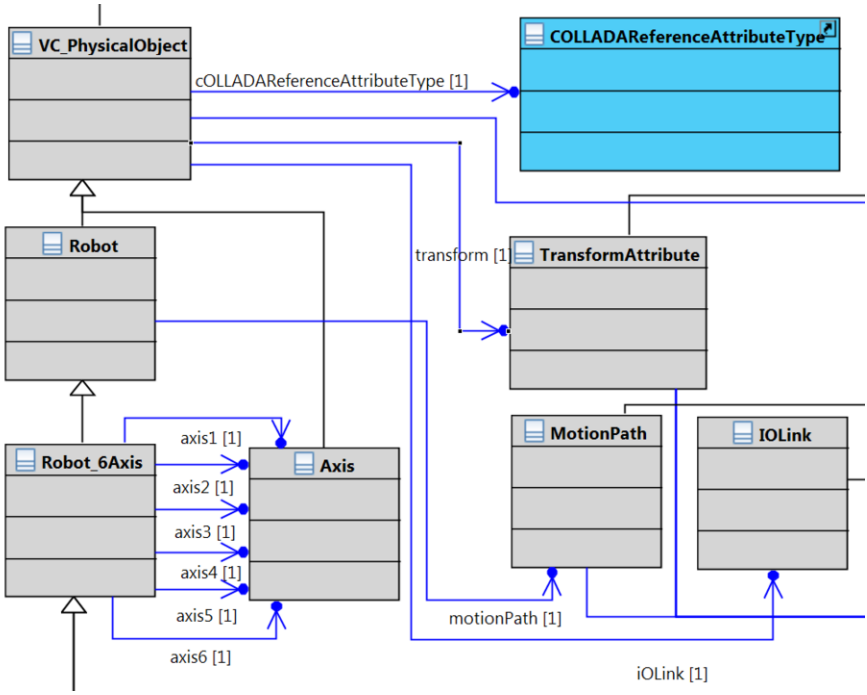

Figure 4. 6 axis robot as simplified UML structure.

A robot is shown as an abstract UML model in Figure 4. To enable the robot model to reference the necessary CAD data, the COLLADAReferenceAttributeType of the AutomationML class model is used.

The motion path of the robot's movements consists of different points with their position and rotation in vector space $\mathbb{R}^3$. As shown in the figure, a new class MotionPath has been created for the motion path to connect the points with the motion path and the robot. This helps a potential downstream engineering algorithm to identify the motion path and points to be used for motion synthesis. With the structure illustrated, the rule set described in the next section is able to create an instance of the robot model.

For the creation of the robot cell, the functionality is still needed to connect the inputs and outputs of the robots in order to exchange any required signals within the logic part of the robot program. This can be done by the class IOLink, which is able to describe the connections between the two robots. For reasons of simplification, no

separate class is created because the functionality is already achieved by the existing class VC_PhysicalObject, which is also shown in Figure 4.

2.3. Defining the rules and compiling

This chapter describes the instance creation process of the robot cell described in sections 2.1 and 2.2. You need a set of rules that uses the defined vocabulary to create the instances correctly. In order to demonstrate the flexibility of graph-based design languages, the motion path and input/output connections of the robots are also generated.

Rules are created according to a schema, i. e. to search for a certain element in the graph and to append newly created instances to the found element. An example of the rule formation within the robot cell example is, to search for the robotic cell as an element and append the two robots to it as new instances. The logic in which elements are connected is defined by the vocabulary. Figure 5 shows the rule formation for the creation of a robot. The rule searches for the VC_PhysicalObject with the name RobotCell and creates a 6-axis robot with all six axes. The connection to the other components is also defined within the rule.

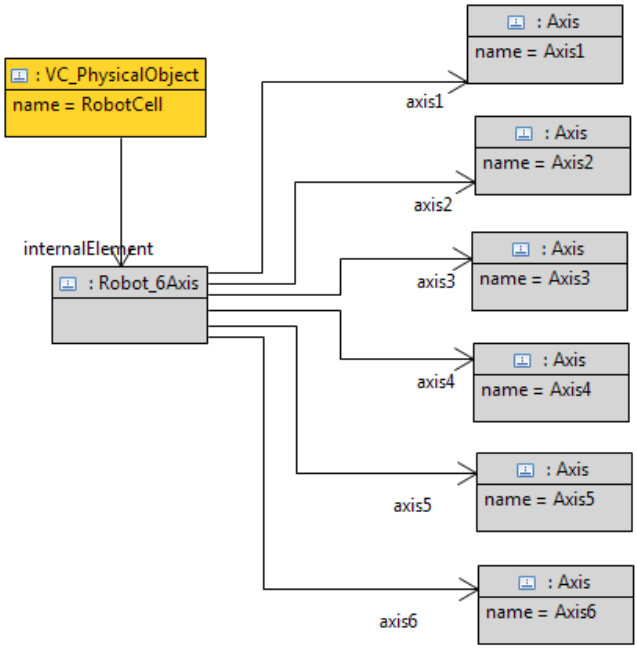

Figure 5. Rule for robot creation.

The other rules for the robotic cell are created according to the same scheme:

- Creation of the first robot cell consisting of a floor and a working device
- Creation of the first robot
- Creation of the second robot
- Attaching a position attribute to each physical object

- Append a CAD reference to each physical object by either creating a new geometry within the compilation process or by linking existing CAD models.
- Calculation and application of the positions of each physical object within the robot cell
- Calculation and creation of the motion path for both robots, based on a point-to-point motion.
- Connection of the robots' inputs and outputs to each other if required

The rule set can then be used to create the missing instances needed for the robot cell specified in the design language. This is done with the help of the Design Compiler 43 [9], which was already mentioned in chapter 1.1. The compiler creates an instance graph that can be used by an engineering algorithm or an export interface. An example of such an instance diagram is shown in Figure 6. Each node in the graph contains the data of the simulation model as a result of the evaluation of the rules during compilation.

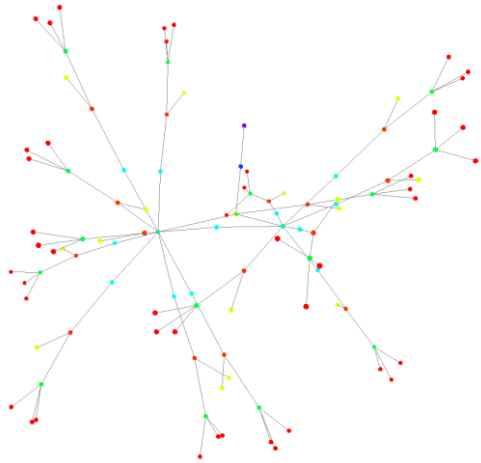

Figure 6. Instance graph.

2.4. The AutomationML file and its use

The instance graph is the starting point for engineering algorithms and is an export interface. The AutomationML interface uses the instance graph to determine which subsections of the AutomationML target file must be filled. A typical AutomationML file consists of four sub-sections, which are composed as follows:

- System Unit Class Library
  All classes used in this AutomationML model are described in this library. They are defined by their attributes, roles and interfaces.
- Role class library
  This library is used to define roles within the model. A large number of roles are already predefined in the AutomationML standard.
- Interface Class Library
  The interaction possibilities between objects are described in this library. An

example of a simple interface would be a mains plug for the power connection of a system.

- Instance hierarchy
  The most important sub-section within the AutomationML file is the instance hierarchy. It declares the structure and relationships between the individual objects and can be used as a starting point for sophisticated engineering algorithms.

For further use of the AutomationML model, it makes sense to fill all four sub-sections, but it is not mandatory. Only the instance hierarchy is required for the robot cell example. Since the UML model of the example continuously uses CAEX and AutomationML class libraries, it is easy to generate the instance hierarchy. The algorithm that generates the instance hierarchy from the instance diagram runs through the diagram and searches for objects and their attributes that need to be serialized.

The AutomationML file thus generated can then be imported into other industry standard applications, such as the RF: Suite program from EKS InTec GmbH [15], which is able to generate a graphical virtual commissioning model from the AutomationML model. There are also interfaces for AutomationML and PLC to the Game Engine Unity 3D [18] to create simulation models. By connecting PLCs to the machines and systems to be simulated as well as the robots, entire production lines can be automatically and virtually put into operation. The connection is made according to the AutomationML guideline using the PLCopen XML standard. Figure 7 shows the graphical simulation model of the application example robotic cell, which was created by using graph-based design language in combination with AutomationML. The model can be used for virtual commissioning including tests of robot movements.

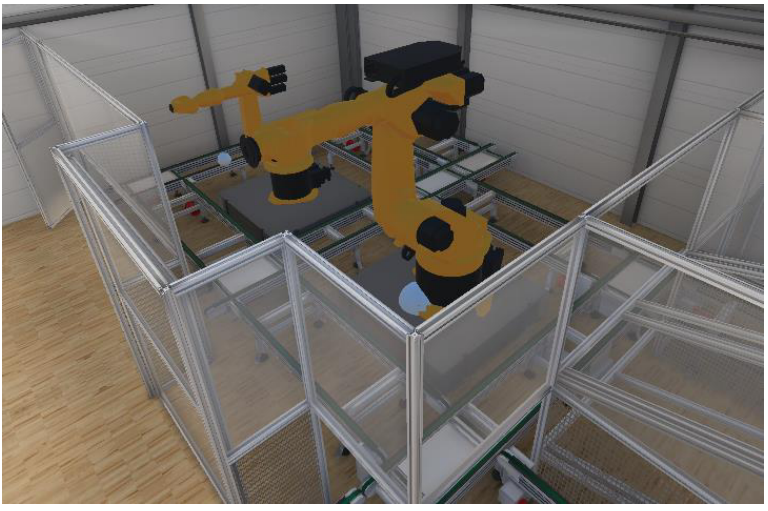

Figure 7. Graphical model for virtual commissioning

3. Conclusion

The use of AutomationML together with graph-based design languages enables a complete digital product lifecycle from requirement definition to virtual

commissioning. This makes it possible to optimally support development processes for the new type of Intelligent Factories and Industry 4.0 systems with IT tools. If the designer changes something about the product, all downstream development processes are generated completely and automatically in order to determine resulting changes, including the means of production. The article shows the application graph-based design languages using a robot cell as an example and the use of AutomationML as a data format for creating the simulation model for virtual commissioning.

Acknowledgement

The German research project ´Digitaler Produktlebenszyklus (DiP)´ (information: http://dip.reutlingen-university.de) is supported by a grant from the European Regional Development Fund and the Ministry of Science, Research and the Arts of Baden-Württemberg, Germany (information: http://www.rwb-efre.baden-wuerttemberg.de).

References

[1] P. Klimant, *Virtual-Reality-gestützte Kinematik- und Materialabtrassimulation für Fräsmaschinen mittels Hardware-in-the-Loop*, Verl. Wiss. Scripten, Chemnitz, 2013.
[2] M. Bergert, J. Kiefer, S. Höme and C. Fedrowitz, *Einsatz der Virtuellen Inbetriebnahme im automobilen Karosserierohbau – Ein Erfahrungsbericht*, Kuka Systems, Augsburg, 2009.
[3] H. Kagermann, W. Wahlster and J. Helbig, *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0*, acatech, Frankfurt, 2013.
[4] O. Genschar, S. Gerlach, M. Hämmerle, T. Krause and S. Schlund, *Produktionsarbeit der Zukunft-Industrie 4.0*, Fraunhofer Verlag, Stuttgart, 2013.
[5] N. Beisheim and M. Kiesel, *Industrie 4.0: Neue Anwendungsfelder für die Simulation*, Forschungsreport für den Maschinenbau in Baden-Württemberg, pp. 39-41, 2014.
[6] S. Rudolph and B. Kröplin, Entwurfsgrammatiken – Ein Paradigmenwechsel?, *Der Prüfingenieur*, 26, 2005, pp. 34-43.
[7] S. Rudolph, Aufbau und Einsatz von Entwurfssprachen für den Ingenieurentwurf, *Forum Knowledge Based Engin.*, CAT-PRO, 2003.
[8] S. Rudolph, On the problem of multi-disciplinary system design - and a solution approach using graph-based design languages, *1st ACCM Workshop on Mechantronic Design*, Linz, November 30, 2012.
[9] "DESIGN COMPILER 43", IILS Ingenieurgesellschaft für Intelligente Lösungen und Systeme mbH, http://www.iils.de/, Accessed: Jan 8 2018.
[10] M. Ramsaier, C. Spindler, R. Stetter, S. Rudolph and M. Till, Digital representation in multicopter design along the product life-cycle, in: *10th CIRP Conference on Intelligent Computation in Manufacturing Engineering - CIRP ICME '16*, Ischia, Italy, 2016.
[11] M. Vogel, *Über Ordnungsmechanismen im wissensbasierten Entwurf von SCR-Systemen - Bericht aus dem Institut*, Universität Stuttgart, Stuttgart, 2016.
[12] R. Drath, A. Luder, J. Peschke and L. Hundt, AutomationML - the glue for seamless automation engineering, *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Hamburg, 2008, 10.1109/ETFA.2008.4638461.
[13] A. Lüder, L. Hundt and A. Keibel, Description of manufacturing processes using AutomationML, *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, Bilbao, 2010, DOI: 10.1109/ETFA.2010.5641346.
[14] AutomationML e.V., 2018, Available: http://automationml.org., Accessed: Jan 8 2018.
[15] EKS InTec GmbH, 2018, http://www.eks-intec.de, Accessed: Jan 8 2018.
[16] M. Kiesel, N. Beisheim and T. Breckle, Generierung und Anreicherung von virtuellen Inbetriebnahmemodellen durch graphenbasierte Entwurfssprachen, In: W. Commerell and T. Pawletta (eds.): *ASIM-Treffen STS/GMMS 2017*, ASIM Mitteilung, 161, ARGESIM Verlag, Wien, 2017.
[17] M. Kiesel, P. Klimant, N. Beisheim, S. Rudolph and M. Putz, Using Graph-based Design Languages to Enhance the Creation of Virtual Commissioning Models, *Procedia CIRP*, Vol. 60, 2017, pp. 279-283.
[18] Unity 3D, 2018, https://unity3d.com/de, Accessed: Jan 8 2018.