[CJW+23] B. Caesar, N. Jansen, M. Weigand, A. Fay, B. Rumpe:
Extracting Hardware Reconfiguration Models Based on Knowledge Synthesis from STEP Files.
In: 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C),
pp. 434-443, IEEE, Oct. 2023.

2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) | 979-8-3503-2498-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/MODELS-C59198.2023.00077

Extracting Hardware Reconfiguration Models based on Knowledge Synthesis from STEP Files

Birte Caesar[1], Nico Jansen[2], Maximilian Weigand[1], Alexander Fay[1], Bernhard Rumpe[2]

***Abstract*—Today's industrial manufacturing challenges force manufacturers to optimize and increase the flexibility of their facilities. In practice, this requires analyzing, preparing for adaptation, and adapting brownfield manufacturing systems. The digital twin, a digital representation of a manufacturing system, is a key enabler for efficiency, flexibility, and sustainability. Unfortunately, the analysis and preparation of brownfield systems for adaptation, as well as the creation of digital twins, are challenging and time-consuming tasks. This paper presents an approach to automatically create digital models of systems based on 3D CAD models. To this end, the CAD data, stored as a STEP file, is analyzed to extract relevant information for a subsequent graph analysis, which is used to identify components, their dependencies, and the resulting functional modules. Finally, the gained knowledge is transformed into feature models that can be used as a digital model for configuration selection to support automatic reconfiguration planning of brownfield manufacturing systems. The developed approach is evaluated based on an industrial use case of a soft gripper system.**

***Index Terms*—CAD data, Reconfiguration, Graph Analysis, Adaptation, Reengineering, Digital Twin, Digital Model**

I. Introduction

The manufacturing industry faces increased competition due to lower market entry barriers as a result of globalization and individualized customer demands. Furthermore, shorter product life cycles and innovation cycles, as well as disrupted manufacturing operations and supply chains, a problem exacerbated by the COVID-19 pandemic, require reconfigurable production systems [1]. To keep companies competitive, entire value chains need to be optimized and automated based on digitally available information.

In practice, this means that existing and operating manufacturing systems must be prepared for reconfiguration to be able to adapt to these changes [2]. In this context, the topic of digital twins (DTs) receives increasing attention, as a study by the market research company Gartner shows that 62 percent of interviewed companies plan to implement DTs and 13 percent already use DTs [3]. To develop a DT, first its desired characteristics and purpose need to be defined. The following implementation is usually carried out specifically for the respective application and is not integrated with the system development [4]. Especially in the case of brownfield systems, where the DT is created independently of the system development, the DT modeling effort is increasingly high, as the virtual representation of real entities must be completely remodeled [5]. Thus, engineering of DTs can be time-consuming and complicated [4].

Today's manufacturing systems were designed to be in use for several decades [6]. Accordingly, many brownfield systems require adaptation and extensions to allow for further years of operation. Therefore, new configurations must be created during the operation phase to adjust to changed requirements. Wiendahl et al. [7] describe that modularization is a key enabler for the adaptation of systems. On the one hand, this is due to the fact that modules provide defined functions and, on the other hand, that modules form units that have few interfaces to other components or modules. To modularize and prepare existing manufacturing systems for adaptation, huge manual effort is required [8].

For manufacturing systems, different engineering documents are available that vary in quality and information content. In most cases, 3D CAD drawings are available, representing the hardware components of the manufacturing system. These components fulfill the mechanical process, whereas software components control these processes. The mechanical process can be divided into several functions, clustering components into modules as a basis for reconfiguration. This results in the following research question. *How can modules of hardware components be automatically identified, based on common engineering artifacts, and transformed into a model with a suitable degree of formalization so that the manual modeling effort for the creation of DTs for reconfiguration planning of existing and in-use systems is reduced?*

Therefore, this paper presents an approach to support the creation of modularized system models based on common engineering artifacts, in particular 3D CAD drawings, as well as the creation of reconfiguration models in the form of feature models. The boundary representation of 3D CAD drawings that represent the minimum information content required to be able to extract mechanical processes is used for this purpose. Components, their contact faces, and the degree of freedom (DoF) that any two components that share a contact face have to each other are extracted from 3D CAD drawings provided as STEP files. Furthermore, knowledge about clustering the components into modules is inferred based on the design principles for modularity described by Salvador [9]. The first principle is that each module must provide at least one function, in this context, the ability of two components to enable relative movement to one another. Finally, we transform the knowledge about the system and its configuration into a feature model. The suitability of feature models to capture the

[1]Institute of Automation Technology, Helmut-Schmidt-University, Hamburg, Germany, email: birte.caesar@hsu-hh.de
[2]Software Engineering, RWTH Aachen University, Aachen, Germany, email: jansen@se-rwth.de

979-8-3503-2498-3/23/$31.00 ©2023 IEEE
DOI 10.1109/MODELS-C59198.2023.00077

configurations of manufacturing systems has been evaluated in our previous work [10]. The feature model is extended step by step with additional configurations the system can take resulting in a suitable model for reconfiguration planning of manufacturing systems. This paper is an extended version of a work-in-progress paper published last year [11].

The remainder of the paper is structured as follows. Sec. II introduces background knowledge about STEP and knowledge graphs as well as definitions of key terms. Sec. III presents an overview of related work, followed by Sec. IV, presenting the concept of extracting knowledge of the hardware components based on STEP files, as well as the concept of transferring the knowledge into a knowledge graph and inferring the functional module and the feature model creation. Sec. V presents details of the implementation of the elaborated methodological concept. The evaluation is presented in Sec. VI including a discussion of the findings. The paper concludes with a summary and outlook in Sec. VII on subsequent work.

II. Background Knowledge

The approach elaborated in this manuscript is based on the **ST**andard for the **E**xchange of **P**roduct (STEP) model data, developed by the International Standards Organization and documented in ISO 10303. It is an exchange format designed to include all product-related information along the product life cycle, widely established in the manufacturing industry. It enables the consistent and error-free usage of 3D CAD data across different (often proprietary) tools. The standard is clustered in several parts specifying different aspects of the product and/or product life cycle information. Amongst others, the standard describes the product in the form of an assembly structure [12] as well as a boundary representation [13]. The former is optional (and often omitted when handing over STEP files for reasons of know-how protection), and additionally is a subjective construct defined by the designing engineer and not created based on objective criteria [14]. Whereas the latter, consisting of the geometric and topological representation of an object, is mandatory in a STEP file. Therefore, we base our approach on the boundary representation. However, some tool-specific information gets lost by using the exchange format STEP. Geometric constraints, which are particularly important for kinematics, are among the non-exported information.

By product, the standard refers to all kinds of 3D objects including assemblies or single parts. Accordingly, a manufacturing system can also be referred to as a product. In the following, we use the term system instead of product and component instead of part to avoid confusion, as product in the manufacturing domain is understood as the product that is created by the manufacturing system.

Knowledge formalized in knowledge graphs can be utilized to infer new knowledge automatically by using so-called reasoners. Using standardized query languages (e.g., SPARQL), query engines can be used to retrieve knowledge systematically. These advantages are the reason why knowledge graphs are commonly used for knowledge formalization in engineering. A prominent means for modeling knowledge graphs in a machine-interpretable fashion is the Web Ontology Language (OWL). Ontologies are conceptual representations of knowledge in a specific domain and are commonly used in research and manufacturing engineering [15]. Terminological knowledge about concepts of the domain, their relations, and attributes are formalized in the terminological box (TBox). Similarly, assertional knowledge about individuals, i.e., entities within the domain, is formalized in the assertional box (ABox), following the rules specified in the TBox. Knowledge graphs are graph data structures implementing the concept of an ontology by defining TBox and ABox data as nodes and edges of a directed graph.

III. Related Work

In the following we will compare related approaches and present in Table I an overview of the models used to extract the knowledge, the input files required to use the approach and the resulting output of each approach.

Barbau et al. [16] created an ontology TBox to represent the concepts of the STEP standard as well as an algorithm to automatically transform the STEP data into ABox individuals. The goal of the approach is to link the transferred data with other models that obtain information which are not provided by the STEP file data, e.g., function and behavior description. Therefore, the authors created an alignment ontology for the core product model (CPM) [17] and the open assembly model (OAM) [18]. However, the additional CPM and OAM knowledge must be created manually and is required as input, see Table I. Gong et al. [19] extend this work aiming to infer the assembly structure from STEP files not including the assembly structure description of ISO 10303-44. Perzylo et al. [20] follow a similar approach and as well only considers the boundary representation of objects, described in ISO 10303-42, see Table I. This enables the automatic transformation of STEP and Initial Graphics Exchange Specification (IGES) files into ABox individuals of the so-called OntoBrep ontology as both file types follow the same concept of boundary representation. The OntoBrep ontology comprises a TBox specification for geometric constraints that can be used for robot assembly task description. Whereby it is not explicitly described whether the geometric constraints are automatically inferred or must be created manually. Köcher et al. [21] transfer the assembly structure (ISO 10303-42) from a STEP file as ABox individuals into a system structure defined according to VDI 2206 (TBox). During the transfer, the parts are created as components and the assemblies are created as modules.

Other solutions deal with the semantic annotation of 3D CAD models for better reuse during engineering design. Lupinetti et al. [22], [23] extract from STEP files the assembly structure and analyze the contact type of each two components being in contact. The number of contacts between two components and the resulting degree of freedom is calculated. The shape of the contact surface strongly influences the resulting degree of freedom. This additional information allows more precise, keyword-independent searches of 3D CAD model databases. The approach of Han et al. [24] is based on that of

Lupinetti et al. [22] and extends it by inferring more details of the functional description of the components. First, the components are classified into connecting and functional components. Connecting components are components that connect or attach two components, all others are functional components. Second, based on the classification, the components are grouped. A functional description is assigned to a group of components using a similarity analysis. The similarity is calculated based on a comparison with existing graph patterns in a database that are associated with functions. Vilmart et al. [14] present a similar approach to Lupinetti et al. [22] that allows for the additional analysis of cellular models where single components are merged and information about contact faces is missing. Furthermore, analysis of symmetric patterns and repetition of these within one system are used to define modules. In this context, the term "module" is used to refer to assemblies that do not necessarily represent a function as defined in the Introduction. In Table I it is referred to as design module.

Another set of approaches extracts structural, kinematic, or functional information of systems for further use in, for example, virtual commissioning. The difference is that proprietary file formats are used for the extraction as more information such as geometric mating constraints. Thongnuch et al. [25] present a method for the automatic generation of kinematic simulation models from mechanical CAD assembly models for virtual commissioning of mechatronic systems. Components of the assembly model and assembly constraints, i.e., kinematic relations between the components, are extracted and mapped to kinematic joints. Based on the joints and constraints, motion vectors of the machine components are derived and combined based on a fixed root component, resulting in a kinematic motion chain of the system. This generated range of motion and geometrical information is formulated as COLLADA model for standardized geometry and kinematics information exchange. Further, Hildebrandt et al. [26] present a method to extract kinematic skills, i.e., abilities to implement a production-related process, from 3D CAD assembly data of a robotic gripping unit to assist engineers in checking its functionality. The method utilizes movement restrictions and materials to infer kinematics, reachable positions and a maximum payload. The information is mapped into a target ontology by means of a rule set. The ontology assigns reachable positions to components and combines movement vectors of components to movement descriptions of the system. Kinematic skills are inferred from these descriptions in combination with production system information.

The presented approaches show potential methods for extracting information from 3D CAD models and the value of reusing existing 3D CAD data across applications. Furthermore, the related work shows that the information relevant to our approach can be extracted from the STEP files. As described in the introduction, a function is the freedom of relative movement of components to each other. The DoF are extracted explicitly by Lupinetti et al. and Han et al. and indirectly by Thongnuch et al. and Hildebrandt et al. within the kinematic description, see Table I. However, Han et al., Thongnuch et al., and Hildebrandt et al. use proprietary data formats that contain kinematic information that is not accessible when using STEP files. The work of Lupinetti et al. provides an important scientific basis for our approach. Furthermore, the approaches that take into account the assembly structure either use information that is not available in all STEP files (ISO 10303-44), such as Lupinetti et al. and Köcher et al., or rebuild the assembly structure (ISO 10303-42), such as Gong et al., without taking into account that the assembly structure is a subjective construct defined by the designing engineer and not created based on objective modularization criteria [14]. Vilmart et al. creates the assembly structure based on the contact classification and the identified design modules. However, the division of the modules does not correspond to the principles of modularization as already described above. The approach of Köcher et al. uses the same target model as our approach (VDI 2206), see Sec. IV-B, but adopts the assembly structure from the STEP file without any modification, thus neglecting that the assembly structure does not follow any modularization principles. In general, the limitations apply to the usage of information not available in all STEP files, i.e., ISO 10303-44 information or proprietary CAD file information, or considering a assembly structure not based on objective modularization criteria.

To the best of our knowledge, there are no approaches that use 3D CAD files to create feature models to represent the hardware configurations of mechatronic systems. However, in the context of mechatronic systems, there are approaches that build feature models based on the existing control code [27]–[29] or process descriptions [30]. Furthermore, there exist several approaches that deal with merging feature models [31], [32]. These approaches assume that feature models exist and that merging is necessary due to their evolution, for example, due to multiple teams working on the same models.

In summary, there is no approach that takes into account the hardware structure of the mechatronic system when creating reconfiguration models (i.e., feature models). Either 3D CAD data are analyzed or feature models are created. Therefore, the aim is to develop a system structure variability mining approach that considers 3D CAD data as a basis for the feature model creation while taking into account objective modularization principles.

IV. Method for Reconfiguration Model Extraction

In this section (a) the method to identify the functions of the examined systems, (b) the classification of the system components into modules, and (c) the reconfiguration model creation are described. Following assumptions have to be considered to conduct the analysis. All components are considered as rigid parts and the contacts of these parts do not change over time. Further, models with volumetric interferences can be processed but no meaningful results are created. The method, therefore, considers only surfaces in contact, and if a clearance is detected, the surfaces are processed as not in contact. Only STEP models following the application protocol AP214 can

TABLE I: Related Work Overview

Approach	Model	Input	Output
Barbau et al.	ISO 10303-42 ISO 10303-44 CPM OAM	STEP file CPM file OAM file	Integrated Graph
Gong et al.	ISO 10303-42	STEP file	Assembly Structure
Perzylo et al.	ISO 10303-42	STEP file IGES file	Geometric Constraints
Köcher et al.	ISO 10303-44 VDI 2206	STEP file	Assembly Structure
Lupinetti et al.	ISO 10303-4 ISO 10303-44	STEP file	Assembly Structure Contact Classification Degree of Freedom
Han et al.	not specified	Proprietary CAD file	Assembly Structure Degree of Freedom Geometric Constraints Component Classification Assembly Function
Vilmart et al.	ISO 10303-42	STEP file	Assembly Structure Contact Classification Symmetric Patterns Design Modules
Thongnuch et al.	not specified	Proprietary CAD file	Kinematics
Hildebrandt et al.	not specified	Proprietary CAD file	Kinematics
This approach	ISO 10303-42 VDI 2206	STEP file	System Structure Degree of Freedom Contact Classification Reconfiguration Model

be processed, as there are syntactical differences for each STEP application protocol. The procedure can be clustered into three main tasks, which will be subsequently explained in Section IV-A, IV-B and IV-C. First, extracting the relevant data from the STEP file and analysing the contact faces of the components regarding their DoF, see Fig. 1, ①. Second, converting the gathered information into a knowledge graph to perform the knowledge graph analysis to identify the modules (Fig. 1, ②). Third, creating feature models based on the created knowledge (Fig. 1, ③).

A. Component & Degree of Freedom Extraction

Generally, STEP provides a topological and geometrical export of the 3D object. The topology describes adjacency relationships and the position and arrangement of geometric objects, while the geometry is the mathematical description of the object itself. However, knowledge of the functional hierarchy is not explicitly provided. To extract mechanical interfaces between components, the possible interactions between the different solids concerning their mutually relative position and shape need to be analyzed. Therefore, the geometry of atomic elements of the solids, i.e., their faces, edges, and vertices, are pairwise analyzed for contact events to derive their mutual translation and rotation properties to each other.

Translational events result from at least two faces of different components being in contact. The result of this contact is a restriction of the translational movement to a specific direction. The direction, in which they interlock, is given by the normal vector of the corresponding faces. Since 3D

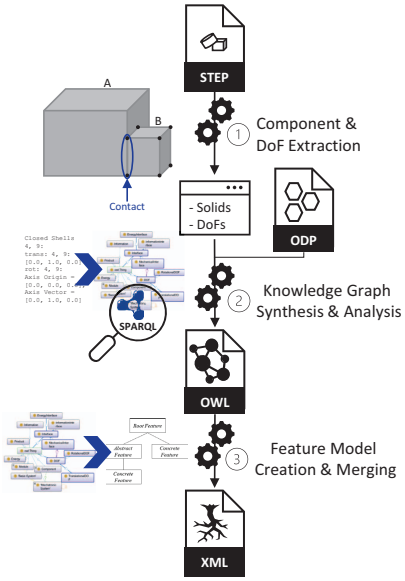

Fig. 1: Proposed method for STEP data extraction to reconfiguration model

modeling of mechanical systems allows complex geometric shapes, performing this translational detection analysis thoroughly is essential. Even relatively simple components have concave surfaces (e.g., gears). Determining the exact collision of such solids is a complex and computationally expensive task known from computer graphics [33], which can result in a long computation times. However, since the detection described does not have to be performed in real-time, it is possible to perform the analysis pairwise for all faces without any simplifications. Whenever a contact is detected between two solids, we store a translational event for the associated assemblies in the normal vector direction. Since various faces with different orientations can exhibit contact points, multiple translational events between two components are possible. Rotational events result from two curved (e.g., cylindrical) surfaces of different components forming a kind of joint. This occurs when these faces have the same radius, are bent around the same axis, and cover the same space. In contrast to the constraining nature of a translational event, a rotational event actually allows for a mutual rotation of the respective components around the shared axis. For components without interlocking faces, no constraints are derived.

Since an event between two components yields a single motion constraint only, the next step involves analyzing all events in a shared context to derive their mechanical interface. For instance, a rotational event not only allows a rotation around a corresponding axis but implicitly prohibits translation in all directions orthogonal to that axis. Thus, after collecting the events between all components of a STEP file, we use them to extract their DoFs. The DoF is defined by the number of independent variables of a system, which mechanically characterize the independent motions of a body. It is expressed by its translational and rotational possibilities of motion.

TABLE II: Exemplary events and their corresponding translational and rotational DoFs in x, y, and z direction

	Events	T_x	T_y	T_z	R_x	R_y	R_z
1	T(1,0,0)	0	✓	✓	-	-	-
2	T(1,0,0), T(0,1,0)	0	0	✓	-	-	-
3	R^Y(0,0,0)	0	✓	0	0	✓	0
4	T(0,1,0), R^Y(0,0,0)	0	0	0	0	✓	0
5	R^Y(0,5,0), R^Y(0,9,0)	0	✓	0	0	✓	0
6	R^Y(0,5,0), R^Y(3,0,4)	0	✓	0	0	0	0
7	R^X(0,0,0), R^Y(1,1,0)	0	0	0	0	0	0

Generally, translation and rotation with respect to each other are possible in any direction of a global coordinate system. For simplicity reasons, but without loss of generality, we explain the DoF derivation based on the standard axis directions x, y, and z. Table II depicts exemplary event occurrences between two components and their corresponding DoFs concerning translation and rotation in, respective around, these three coordinate directions. A single translational event (l. 1) in x direction results mutual translational DoFs towards y and z. Similarly, two such events in different directions block the movement in the direction the event occurred, so that translation is only possible in the direction of an axis orthogonal to the event constraints (l. 2). In contrast, a rotational event allows rotation along the specified axis as well as translation along this axis (l. 3) as long as it is not blocked by another translational event in the same direction (l. 4). Multiple rotational events between two assemblies can have a very different impact on the DoFs depending on their characteristics. For instance, two rotational events around the same axis may affirm their DoFs (l. 5) if and only if their origin lies on their rotation axis. This is the case if two joints are placed on top of each other, as in a door hinge. On the other hand, if their origins are displaced, rotation is blocked entirely (l. 6) while translation along the direction axis is still possible. In the case of two rotational events with different axis (l. 7), no movement is possible anymore, independent of their origins. After evaluating all events between the two components, we receive their remaining local DoFs.

B. Knowledge Graph Synthesis & Analysis

Once the components and their contact interfaces including their DoFs have been extracted and computed from the STEP file, the information is transformed into a knowledge graph. Fig. 2 shows the TBox the knowledge graph synthesis and analysis is based on. It is based on an ontology design pattern (ODP) representing the guideline VDI 2206. The general concept of creating reusable, modular TBoxes to adapt and extend them according to project specific requirements was defined in [34]. In this paper, the ODP of the VDI 2206 established in [15] is reused. To describe the DoFs of two components, the TBox has to be extended (cf. Fig. 2). While grey elements represent the existing ODP, green boxes and green highlighted relations depict new elements.

The extracted solids are created as individuals of the class *Component*. Individuals of classes at a higher hierarchical level (i.e., *Modules* and *MechatronicSystems*) are created later on,

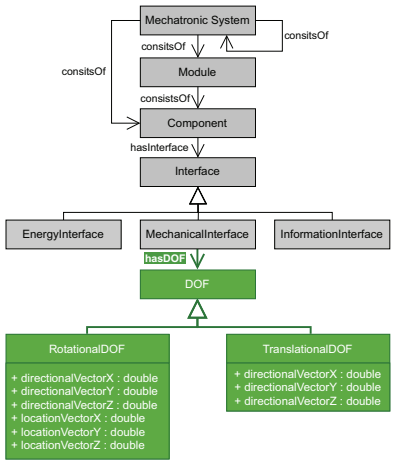

Fig. 2: Excerpt of the Extended TBox ODP VDI 2206

during the knowledge graph analysis. Mechanical interfaces resulting from detected events (cf. Sec. IV-A) are created as individuals of the class *MechanicalInterface* along with object properties *hasInterface* connecting the corresponding components and the mechanical interface. The mechanical interface is created for each pair of components, regardless of whether the calculation resulted in no DoF (i.e., the components are connected rigidly) or DoFs have been calculated. If DoFs have been calculated, they are created as instances of the new class *DoF*, or more specifically, the appropriate subclass *TranslationalDoF* or *RotationalDoF*. *DoF*, *TranslationalDoF* and *RotationalDoF* are extensions to the VDI 2206 ODP. The created mechanical interface is connected with these DoFs using the object property *hasDoF*, which is also an extension to the VDI 2206 ODP. Each *DoF* is defined by a directional vector and, in case of a rotational DoF, a location vector. Those vectors are defined by the data properties *directionalVectorX, -Y and -Z* and *locationVectorX, -Y and -Z*. This data also results form the calculation of DoFs, according to the logic described in Sec. IV-A and exemplified in Table II.

The information modeled so far represents the explicitly defined geometric objects in the STEP file as well as resultant geometric constraints. The goal of the knowledge graph analysis is to group the system components into modules by using implicit knowledge. Our approach is based on the design principle for modularity that are defined by Salvador [9] based on a systematic literature review including publications from 1963 to 2003. First, each module includes at least the components that contribute to fulfill one function, titled functional binding. As stated in the introduction, we define a function as the ability of two components to enable a relative movement to one another. This definition can be considered analogous to the definition of capabilities in [35]. Relative movement can be enabled in rotational and translational direction. A complex movement, involving multiple components, can therefore be divided into multiple functions. The second principle, loose coupling, is that few relationships are implemented between modules, but strong relationships exist between components within modules.

In a first step, components are grouped together whenever they are adjacent and able to move relatively to each other (i.e., they are connected with mechanical interfaces that have assigned DoFs). Groups also consist of components that are related to group members by sharing a mechanical interface. For each group of components, an individual of the class *Module* is created along with the object property *consistsOf* between the new module and each component in the group.

Static components are components that are not assigned to any module in the first step, i.e. they are either not adjacent to any other component or adjacent and fixed to another component. In terms of the defined TBox, static components do not have an assigned mechanical interface, or have one or more mechanical interfaces without any assigned DoFs.

The second analysis phase starts considering all static components. Whenever a static component has a mechanical interface with exactly one other component belonging to a module, it is assigned to the same module. If it has a mechanical interface with multiple components belonging to a module, the static component is assigned to the same module, if all of those components belong to the same module. In any other case, the static component is not assigned to any module.

In the last step, a single individual of the class *Mechatronic-System* is created, and the object property *consistsOf* is created between the mechatronic system and each previously defined module. Also, all components which have not been assigned to any module are assigned to the mechatronic system. This results in a system hierarchy strictly following the design principles for modularization.

C. Feature Model Creation & Merging

Once the system structure is defined in the knowledge graph, it can be transformed into a feature model. Each created knowledge graph represents exactly one configuration of the system. To create a reconfiguration model representing the configurations the system can take, it would be necessary to introduce variability means into the knowledge graph. This would require a profound extension of the ontology TBox and a rule set to maintain model consistency. Therefore, feature models are preferred to ontologies as reconfiguration models due to their usability and performance. This transformation offers a further advantage, as only the configuration-relevant aspects are transferred to the feature model and the holistic system knowledge is retained in the form of the knowledge graph. A clear separation between system knowledge and configuration logic is achieved in this way. To avoid losing the semantic meaning contained in the knowledge graph and to establish an unambiguous mapping of features to individuals, the individual's IRI is used to name the feature.

To transform the system structure into a feature model, the instance of class *MechatronicSystem* is created as *Root Feature*. As the VDI 2206 ODP allows a mechatronic system itself to consist of several mechatronic systems, only the top hierarchy mechatronic system is created as root feature. Each other mechatronic system is created as *Abstract Feature* below the root feature. The instances of class *Module* are created as *Abstract Feature* below the root or respectively below the abstract feature created for the mechatronic system, of which they are part. Finally, the instances of class *Component* are created as *Concrete Feature* either below the abstract feature of the module they are part of or below the abstract feature of the mechatronic system they are part of. The later only applies if the component is not part of any module. Fig. 3 shows the transformation mapping between the VDI 2206 ODP and a feature model, here only visualizing the transformation with exactly one mechatronic system. Additionally, it is assumed that each component in a single configuration of the system is mandatory. Therefore, each concrete feature, representing a component, is created as a mandatory feature, as at this time no knowledge about optional features is present. This results in a feature model from which only one possible configuration can be derived. Only a restriction of such a strong nature can guarantee that no configurations will be derived that cannot be implemented in the real world. By merging several feature models, which each represent one configuration, this restrictive definition is relaxed as knowledge is gained about optional features with each additional analyzed configuration.

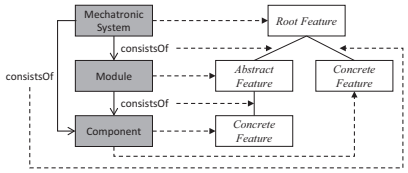

Fig. 3: Mapping from Ontology VDI 2206 into Feature Model

The reconfiguration model is created based on a pairwise comparison of two feature models, such as $\mathcal{FM}_\mathcal{A}$ and $\mathcal{FM}_\mathcal{B}$. There are three types of differences that can occur during comparison. First, the comparison identifies differences in the existence of features, e.g., feature $\mathcal{B}$ is not a feature of the features $\mathcal{F}_\mathcal{B}$ of $\mathcal{FM}_\mathcal{B}$ ($\mathcal{B} \notin \mathcal{F}_\mathcal{B}$). Second, the selection rules differ, for example, different group types are used, or a feature is mandatory in one feature model and optional in the other. Third, there are differences in the location of the feature in the feature tree; for example, the feature $\mathcal{B}$ has a different parent feature in $\mathcal{FM}_\mathcal{A}$ and $\mathcal{FM}_\mathcal{B}$. Each identified difference between the two compared feature models must be considered, and a merging rule is applied to integrate both feature models. These rules are rather strict, as it is necessary to not allow the selection of configurations that have not been specified by the system manufacturer to guarantee safe operation of the system. This is of great importance as we consider explicitly the hardware configuration. Untested, not validated, and approved configurations can lead to complete system failure, as well as serious damage to the system environment, such as personal injury caused by the expulsion of improperly mounted components. Therefore, when merging feature models, the goal is to design the selection rules so that the number of valid configurations is the same as the number of system variants. Due to limited pages the merging rules are not included in this paper, but can be accessed on GitHub[1].

[1] github.com/birtecaesar/FMCreaM

V. Implementation

This section presents insights into our implementation of the presented method. We have created two tools, the first one step-modularization covering step one and two of our presented method (Sec. V-A) and the second, FMCreaM implementing the feature model creation. For details beyond our following explanation we refer to our GitHub repositories[1,2], where the code is publically avialable.

A. Tool Implementation: step-modularization

In STEP, the different parts are stored in a data series-like topology, where each line starts with a unique identifier and further contains a classification label with a list of parameters and references to other identifiers. The geometry of a component is hidden in the structure of the format, making it eligible for serialization but requiring further transformation for an in-depth analysis. Our implementation is realized via MontiCore [36], a language workbench for efficiently developing domain-specific modeling languages. Besides a parser for reading the textual model (i.e., the STEP file), MontiCore automatically generates infrastructure, supporting analyzing, traversing, and cross-referencing the processed model. We process the textual STEP file and, using the visitor infrastructure MontiCore provides [37], automatically transform it into a data structure containing the actual elements, such as shells, their faces, and corresponding edges and vertices.

After preprocessing and deriving the data structure representing the geometry of the model, the tool collects all components for further analysis. Here, we use so-called *closed shells* (representing single solids) as simplification, ignoring further augmented information, such as local displacements or distinguishing metric from imperial units. Including additional geometric transformation steps, which are stored in the STEP artifact, is generally possible but out of the scope of this proof of concept. We pairwise compare the closed shells for translational and rotational contact events (cf. Sec. IV-A) to further derive their mechanical interfaces and mutual DoFs required for the knowledge graph synthesis.

To create a knowledge graph representation of the extracted components as well as the calculated mechanical interfaces with their associated DoFs, the Apache Jena[3] framework is used. Apache Jena is an open-source Semantic Web framework with different features, e.g., an API for creating knowledge graph data programmatically, which is used to implement the steps mentioned in Sec. IV-B. The Apache Jena framework also offers a query interface, which allows to analyze existing knowledge graphs using SPARQL. Furthermore, SPARQL INSERT operations can be used to systematically extend the data in the knowledge graph. The steps described in Sec. IV-B are implemented in this way.

All SPARQL queries found in this work (cf. Listings 1 to 3) have been shortened and are not functional as they are printed. The complete queries can be found in our GitHub repository[3].

[2] github.com/hsu-aut/step-modularization
[3] jena.apache.org

First, components are grouped together whenever they are connected with mechanical interfaces that have assigned DoFs. Listing 1 shows a SPARQL INSERT operation that evaluates this kind of relations in the WHERE clause. In the INSERT clause, an auxiliary object property (*temp:hasCommonInterfaceWith*), is created.

```
INSERT {
  ?comp1 temp:hasCommonInterfaceWith ?comp2 .
}
WHERE {
  ?comp1 vdi2206:hasMechanicalInterface ?interface .
  ?comp2 vdi2206:hasMechanicalInterface ?interface .
  ?interface vdi2206:hasDOF/a vdi2206:DoF .
  [...]
}
```

Listing 1: SPARQL query to create auxiliary object properties

Listing 2 shows how this auxiliary object property is used to find groups of components that are connected with mechanical interfaces that have assigned DoFs using SPARQL property paths. Property paths are means to specify a path consisting of a sequence of object properties between two entities in a knowledge graph without specifying in-between individuals. Different operators exist to specify the desired sequence of object properties, as well as the number of object properties in the sequence. Using the operator + in SPARQL denotes a sequence of one or more object properties of one kind in sequence. As explained in Sec. IV-B, two components shall also be grouped, if there is no direct mechanical interface between them, but mechanical interfaces exist across one or more other components. Therefore, property paths are used to implement this behavior in an efficient way, which is the reason for creating the auxiliary object properties beforehand. The INSERT clause creates a new module as well as the object properties connecting the module and components.

```
INSERT {
  ?module a vdi2206:Module .
  ?module vdi2206:consistsOf ?comp .
}
WHERE {
  ?comp temp:hasCommonInterfaceWith+ ?comp2 .
  [...]
}
```

Listing 2: SPARQL operation to create modules

All components that have not yet been assigned to modules are considered as static components. In the next step, several static components are added to the created modules. Listing 3 shows the corresponding SPARQL operation. It evaluates the knowledge graph for static components having mechanical interfaces with other components that have been assigned to modules previously. For such components, the number of distinct corresponding modules is deduced using the COUNT operation. If exactly one module is found, the component is added to this module, by creating an *consistsOf* object property, which is realized by the INSERT clause.

Finally, the main system is created as a *MechatronicSystem* instance. The property *consistsOf* is created between the main

system and each module, as well as each component that could not be assigned to any modules in the previous steps.

```
INSERT {
  ?module vdi2206:consistsOf ?static_comp .
}
WHERE {
  SELECT
    ?static_comp
    (COUNT(DISTINCT ?module) AS ?m_count)
    [...]
  WHERE {
    ?static_comp
      vdi2206:hasMechanicalInterface ?int .
    ?comp2 vdi2206:hasMechanicalInterface ?int .
    ?module vdi2206:consistsOf ?comp2 .
    FILTER ( ?comp2 != ?static_comp )
    [...]
  }
  GROUP BY ?static_component
  HAVING (?m_count = 1)
}
```

Listing 3: SPARQL query to add static components to modules

B. Tool Implementation: FMCreaM

To create a feature model based on the knowledge graph, the data is accessed via SPARQL. The tool to create and merge feature models is created using Python. To query the data a the SPARQL Endpoint wrapper for Python, SPARQL Wrapper[4] is used. To extract the present hierarchy levels of the knowledge graph, four SPARQL SELECT operations are executed. The first query selecting all systems the system consists of. The second query selecting all modules a system consists of. The third query selecting all components a module consist of. The fourth query selecting all components a system consists of, which are not part of a module.

Feature models can be encoded and exchanged as eXtensible Markup Language (XML) documents, a flexible, machine- and human-readable, textual data format. XML documents are hierarchical structured, whereas tags define the elements which can be nested by enclosing them within parents elements. Additionally, each element can be enriched by attributes. To transform the retrieved data into a feature model, the tag “feature” is assigned to leaf features, cf. components. The tags “and”, “alt”, or “or” are used to specify the group type of child features of no leaf features, cf. systems and modules. Furthermore, the attributes “mandatory” and “abstract” are used to indicate the type of features and finally the attribute “name” is used to give each feature a name. As stated above the IRI of each individual is used as name. To create and parse the XML documents the element tree XML API[5] is used.

To pairwise merge two feature models, the XML documents are parsed and converted into Python dictionaries. The dictionaries are analyzed, and the respective rules are applied. The resulting merged feature model is encoded as an XML document using the same tags and attributes as previously described to create a feature model. Additional tags are used to create constraints, such as “impl” to create a requires constraint and “eq” to create an excludes constraint.

[4]sparqlwrapper.readthedocs.io/en/latest/main.html

[5]docs.python.org/3/library/xml.etree.elementtree.html

VI. Evaluation & Discussion

With the evaluation of our approach, we want to demonstrate the consistency of our method. Consistency means that the method should be able to generate a consistent feature model from a STEP file. In addition, the method should be able to produce comparable results when applied repeatedly to the same input data, thus demonstrating reliability and reproducibility. The evaluation is conducted in two stages. For the first stage, one of the authors created 3D CAD files of simple geometric objects, e.g., in the form of wooden toy blocks. The created files are used as follows. Two such objects were placed in space so that they shared no contact face, exactly one contact face, exactly two contact faces, and finally exactly three contact faces. For each of these different numbers of contact occurrences, two test files, i.e., STEP files, were created, so that both planar and cylindrical contact faces were considered, resulting in translational and rotational DoFs. In total, eight test files were analyzed. For each of these test files, the correct results were extracted. In detail, exactly two components were identified, each with no event, one translational or rotational event, two translational or one translational and one rotational event, and finally three events combining two translational and one rotational event and vice versa. In the knowledge graph for each test file exactly two individuals of the type *component* were instantiated, one mechanical interface, and respectively both components are related to that mechanical interface by the object property *hasInterface*. Additionally, the test files with events resulting in a DoF, a DoF is instantiated and related to the respective mechanical interface. Correctly, a DoF was instantiated for five test files and no DoF was instantiated for three test files. The test files for which no DoF was instantiated had no contact (two of the test files) or the movement was restricted by the combination of events (one test file). The correct modularization is not assessable by such a small number of components in each of the test files and will be addressed during a effectiveness evaluation of future work. A feature model was created for each test file, with exactly two mandatory leaf features representing the components, and one root feature representing a system. The merged feature model contains ten optional leaf features because the one contact and no contact test files used the same components, and the two contact test files also reused one component from each of the previous files. The components were selected so that the consistency could be demonstrated even for feature models that not have common leaf features.

As second stage of the evaluation, we execute a case study based on files kindly provided by the Chair of Logistics Systems Technology of the Helmut-Schmidt-University (HSU). This is a soft-gripping system[6] usable, for example, as the end effector of a robotic arm. In total 15 variants are analyzed, nine variants with four gripping fingers and six variants with three gripping fingers. The variants consist of 29 to 39 components. To reduce the error sources and still be able to evaluate the consistency of our method, we have decided to access only the

[6]soft-gripping.com/

modularization and the feature model creation and merging, since these are the method steps that go beyond the state of the art. To do so, we need the components, their contacts, and their DoFs as input. This information was provided as Excel spreadsheets, which are transferred into a knowledge graph based on a rule set. To measure the effectiveness, we checked the number of possible configurations after each additional merged variant. The number of merged variants must equal the number of possible configurations that can be derived from the feature model. In addition, the created modules are examined to see if the same relationships always lead to the same modules and if the modules are suitable for reuse. The latter is evaluated by a researcher of the Chair of Logistics Systems Technology of HSU.

Additionally, we repeated the test for both cases three times each and always received the same results. We conclude our method provides consistency as the reproducibility and the creation of feature models from STEP was demonstrated. How far this result is generalizable is discussed in the following.

A. Discussion

With small and understandable examples, we have proven the consistency of our method. However, during testing the robustness by using different input files from industry partners, to rise the complexity and vary the input data, we encountered difficulties extracting the components. In detail, difficulties arose in extracting the correct number of components if the components are complex geometric objects consisting of multiple solids. During extraction, components are derived directly from solids, neglecting that components can consist of multiple solids. This deficiency can be overcome by extracting the "*advance_brep_shape_representation*", which groups the solids that represent a component. The grouping does not change the overall method. However, it requires an additional step in which the identified contacts of the solids representing a component are deleted. As the surfaces of the individual solids represent the total surface of the complex component, contacts with solids that are not part of the component remain relevant. This problem was also reported by [19], providing a solution for encountering the correct number of components but not solving the challenge of recognizing the correct contact surfaces, since they are repositioned in space by the transformation. To avoid these errors and still be able to evaluate the graph analysis and the feature model creation and merging, we created a rule set for the case study to map the required information from Excel tables into a knowledge graph. The scalability of our approach has been evaluated only to a limited extent, as the size of the case study variants does not reflect the size of industrial systems, where the number of components for small systems is around 200 components, e.g., autonomous logistic towing train. The number of variants compared is in a realistic range. However, the number of variants can reach several hundred.

For the DT, a comprehensive digital representation of the system is critical. Therefore, our approach has the advantage of automatically creating a reconfiguration model for use with the DT, even with the current limitations. As presented in the Related Work section, to the authors' knowledge, there is no approach that allows the creation of a reconfiguration model, i.e., a feature model, based on 3D CAD data. Our approach closes the gap between the creation of the knowledge graph and the resulting feature model for dynamic reconfiguration, thus eliminating the need to reconstruct the data. It also reduces the manual effort required to create the DT model. By using the VDI 2206 as a basis for the module classification and the system description itself, our method can be applied to a varity of systems in the manufacturing domain. Each of the individual method steps can be executed separately from each other and the results used as needed. The merging of the feature models can also be performed separately from the creation, if existing feature models are provided. The code is publicly available, see the respective GitHub repositories[1,2].

VII. Summary & Outlook

This paper presents a method to extract relevant information to infer knowledge to cluster system components into modules that represent mechanical functions. The implementation to extract and identify the DoFs, the transformation into a knowledge graph, the modularization, as well as the feature model creation and merging has been described and can be accessed online. The implementation and evaluation show that stepwise extraction and analysis is an effective way to generate further knowledge about a system's hierarchy. As a first step, we have implemented the mathematical analysis of DoFs for which a knowledge graph is not suited. The strengths of the knowledge graph in generating knowledge based on relationships and interconnections can be built on these calculations. In addition, the use of knowledge graph-based analysis enabled efficient querying based on the SPARQL query language. A particular strength of our approach is the use of the community information resource VDI 2206 as the domain consensus of a system hierarchy. The system description according to VDI 2206 is automatically generated using the objective modularization principles and the STEP file data. Furthermore, we take into account the manufacturing domain specific ensuring that only existing variants can be derived from the merged feature model, by applying rather strict merging rules. Several different STEP files have been used to verify the results and a real world case study was used to evaluate the consistency, but a more in-depth evaluation with industry examples to assess the scalability and effectiveness must performed. Additionally, we will investigate if we can integrate our approach with existing approaches, e.g., [23], to overcome the identified difficulties in extracting the components and DoFs.

In general, our approach shows the potential of reducing the manual work effort for DT model generation. Furthermore, the potential usage of the generated model in the context of reconfiguration of manufacturing system can lead to a reconfiguration time reduction up to 58% [38].

ACKNOWLEDGMENTS

(From SE RWTH:) Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612.

REFERENCES

[1] J. Morgan, M. Halton, Y. Qiao, and J. G. Breslin, "Industry 4.0 smart reconfigurable manufacturing machines," *Journal of Manufacturing Systems*, vol. 59, pp. 481–506, 2021.

[2] X.-L. Hoang, C. Hildebrandt, and A. Fay, "Product-oriented description of manufacturing resource skills," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 90–95, 2018.

[3] Costello, Katie; Omale, Gloria, "Gartner survey reveals digital twins are entering mainstream use," February 2019.

[4] P. Bibow, M. Dalibor, C. Hopmann, B. Mainz, B. Rumpe, D. Schmalzing, M. Schmitz, and A. Wortmann, "Model-driven development of a digital twin for injection molding," in *Advanced information systems engineering*, vol. 12127 of *LNCS Sublibrary: SL 3, Information Systems and Applications, incl. Internet/Web, and HCI*, pp. 85–100, 2020.

[5] D. Braun, M. Riedhammer, N. Jazdi, W. Schloegl, and M. Weyrich, "A methodology for the detection of functional relations of mechatronic components and assemblies in brownfield systems," *Procedia CIRP*, vol. 107, pp. 119–124, 2022.

[6] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, "Evolution of software in automated production systems: Challenges and research directions," *Journal of Systems and Software*, vol. 110, pp. 54–84, 2015.

[7] H.-P. Wiendahl, H. A. ElMaraghy, P. Nyhuis, M. F. Zäh, H.-H. Wiendahl, N. Duffie, and M. Brieke, "Changeable manufacturing - classification, design and operation," *CIRP Annals*, vol. 56, no. 2, pp. 783–809, 2007.

[8] P. Marks, X. L. Hoang, M. Weyrich, and A. Fay, "A systematic approach for supporting the adaptation process of discrete manufacturing machines," *Research in Engineering Design*, vol. 24, no. 3, pp. 1–21, 2018.

[9] F. Salvador, "Toward a product system modularity construct: Literature review and reconceptualization," *IEEE Transactions on Engineering Management*, vol. 54, no. 2, pp. 219–240, 2007.

[10] X.-L. Hoang, B. Caesar, and A. Fay, "Adaptation of manufacturing machines by the use of multiple-domain-matrices and variability models," *IFAC-PapersOnLine*, vol. 52, no. 13, pp. 1361–1366, 2019.

[11] B. Caesar, N. Jansen, M. Weigand, M. Ramonat, C. S. Gundlach, A. Fay, and B. Rumpe, "Extracting functional machine knowledge from step files for digital twins," in *2022 27th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–4, 2022.

[12] ISO 10303-44, "Industrial automation systems and integration - product data representation and exchange - part 44: Integrated generic resource: Product structure configuration," 2019.

[13] ISO 10303-42, "Industrial automation systems and integration - product data representation and exchange - part 42: Integrated generic resource: Geometric and topological representation," 2019.

[14] H. Vilmart, J.-C. Léon, and F. Ulliana, "From cad assemblies toward knowledge-based assemblies using an intrinsic knowledge-based assembly model," *Computer-Aided Design and Applications*, vol. 15, no. 3, pp. 300–317, 2018.

[15] C. Hildebrandt, A. Köcher, C. Küstner, C.-M. López-Enríquez, A. W. Müller, B. Caesar, C. S. Gundlach, and A. Fay, "Ontology building for cyber–physical systems: Application in the manufacturing domain," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 3, pp. 1266–1282, 2020.

[16] R. Barbau, S. Krima, S. Rachuri, A. Narayanan, X. Fiorentini, S. Foufou, and R. D. Sriram, "Ontostep: Enriching product model data using ontologies," *Computer-Aided Design*, vol. 44, no. 6, pp. 575–590, 2012.

[17] S. Fenves, S. Foufou, C. Bock, R. Bouillon, and R. Sriram, "Cpm 2: A revised core product model for representing design information," 2005. NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD.

[18] X. Fiorentini, I. Gambino, V. Liang, S. Foufou, S. Rachuri, M. Mani, and C. Bock, "An ontology for assembly representation," 2007. NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD.

[19] H. Gong, L. Shi, D. Liu, J. Qian, and Z. Zhang, "Construction and implementation of extraction rules for assembly hierarchy information of a product based on ontostep," *Procedia CIRP*, vol. 97, pp. 514–519, 2021.

[20] A. Perzylo, N. Somani, M. Rickert, and A. Knoll, "An ontology for cad data and geometric constraints as a link between product models and semantic robot task descriptions," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2015)*, pp. 4197–4203, 2015.

[21] A. Köcher, C. Hildebrandt, B. Caesar, J. Bakakeu, J. Peschke, A. Scholz, and A. Fay, "Automating the development of machine skills and their semantic description," in *2020 IEEE 25th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1013–1018, 2020.

[22] K. Lupinetti, F. Giannini, M. Monti, and J.-P. Pernot, "Automatic extraction of assembly component relationships for assembly model retrieval," *Procedia CIRP*, vol. 50, pp. 472–477, 2016.

[23] K. Lupinetti, F. Giannini, M. Monti, and J. Pernot, "Cad assembly descriptors for knowledge capitalization and model retrieval," in *Tools and methods of competitive engineering*, 2016.

[24] Z. Han, R. Mo, H. Yang, and L. Hao, "Cad assembly model retrieval based on multi-source semantics information and weighted bipartite graph," *Computers in Industry*, vol. 96, pp. 54–65, 2018.

[25] S. Thongnuch and A. Fay, "Mcad2sim: Towards automatic kinematic joints recognition," *Computer-Aided Design and Applications*, 2020.

[26] C. Hildebrandt, M. Glawe, A. W. Müller, and A. Fay, "Reasoning on engineering knowledge: Applications and desired features," in *European Semantic Web Conference*, vol. 10250 of *Lecture notes in computer science*, pp. 65–78, 2017.

[27] D. Wille, *Custom-Tailored Product Line Extraction*. Dissertation, Technische Universität Braunschweig, Braunschweig, 31.08.2018.

[28] A. Schlie, K. Rosiak, O. Urbaniak, I. Schaefer, and B. Vogel-Heuser, "Analyzing variability in automation software with the variability analysis toolkit," in *Proceedings of the 23rd International Systems and Software Product Line Conference volume B - SPLC '19*, pp. 1–8, 2019.

[29] A. Schlie, A. Knüppel, C. Seidl, and I. Schaefer, "Incremental feature model synthesis for clone-and-own software systems in matlab/simulink," in *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A* (R. E. Lopez-Herrejon, ed.), pp. 1–12, 2020.

[30] K. Meixner, K. Feichtinger, R. Rabiser, and S. Biffl, "Efficient production process variability exploration," in *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems*, ACM Digital Library, pp. 1–9, 2022.

[31] S. Segura, D. Benavides, A. Ruiz-Cortés, and P. Trinidad, "Automated merging of feature models using graph transformations," in *Generative and transformational techniques in software engineering II*, vol. 5235 of *Tutorial*, pp. 489–505, 2008.

[32] V. Bischoff, K. Farias, L. J. Gonçales, and J. L. V. Barbosa, "Towards a semiautomatic tool to support the integration of feature models," in *Proceedings of the XV Brazilian Symposium on Information Systems*, ACM Digital Library, pp. 1–8, 2019.

[33] J. A. Carretero and R. Uppuluri, "A novel optimization-based pruning strategy for concave minimum distance problems," in *Proceedings of the 2004 CSME Forum, London, Ontario*, pp. 1–7, 2004.

[34] J. R. Reich, "Ontological design patterns: Metadata of molecular biological ontologies, information and knowledge," in *Database and Expert Systems Applications*, pp. 698–709, 2000.

[35] E. Järvenpää, P. Luostarinen, M. Lanz, and R. Tuokko, "Presenting capabilities of resources and resource combinations to support production system adaptation," in *2011 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, pp. 1–6, 2011.

[36] K. Hölldobler, O. Kautz, and B. Rumpe, *MontiCore Language Workbench and Library Handbook: Edition 2021*. Aachener Informatik-Berichte, Software Engineering, Band 48, May 2021.

[37] F. Drux, N. Jansen, and B. Rumpe, "A Catalog of Design Patterns for Compositional Language Engineering," *Journal of Object Technology (JOT)*, vol. 21, pp. 4:1–13, October 2022.

[38] B. A. Talkhestani, D. Braun, W. Schloegl, and M. Weyrich, "Qualitative and quantitative evaluation of reconfiguring an automation system using digital twin," *Procedia CIRP*, vol. 93, pp. 268–273, 2020.