

Semantic rules for capability matchmaking in the context of manufacturing system design and reconfiguration

Eeva Järvenpää^a, Niko Siltala^a, Otto Hylli^b, Hasse Nylund^a and Minna Lanz^a

^aFaculty of Engineering and Natural Sciences, Automation Technology and Mechanical Engineering, Tampere University, Tampere, Finland;

^bFaculty of Information Technology and Communication Sciences, Computing Sciences, Tampere University, Tampere, Finland

ABSTRACT

To survive in dynamic markets and meet the changing requirements, manufacturing companies must rapidly design new production systems and reconfigure existing ones. The current designer-centric search of feasible resources from various catalogues is a time-consuming and laborious process, which limits the consideration of many different alternative resource solutions. This article presents the implementation of an automatic capability matchmaking approach and software, which searches through resource catalogues to find feasible resources and resource combinations for the processing requirements of the product. The approach is based on formal ontology-based descriptions of both products and resources and the semantic rules used to find the matches. The article focuses on these rules implemented with SPIN rule language. They relate to 1) inferring and asserting parameters of combined capabilities of combined resources and 2) comparison of the product characteristics against the capability parameters of the resource (combination). The presented case study proves that the matchmaking system can find feasible matches. However, a human designer must validate the result when making the final resource selection. The approach should speed up the system design and reconfiguration planning and allow more alternative solutions be considered, compared with traditional manual design approaches.

ARTICLE HISTORY

Received 24 June 2021

Accepted 18 May 2022

KEYWORDS



Manufacturing ontology; capability matchmaking; semantic rules; automatic inference; smart manufacturing; SPIN rules

1. Introduction

Smart manufacturing calls for responsive production systems as well as design and planning approaches that allow companies to operate efficiently in a highly dynamic environment. This dynamism arises from the ever-increasing requirements of the highly flexible production of individualized products in small batches (Bortolini, Gabriele Galizia, and Mora 2018; Lu and Xu 2018). Presently, system design and reconfiguration planning are still human-driven laborious activities. They require the designer to browse various paper and online catalogues, searching for suitable resources. The designers must complete several tasks during this search and resource selection process. First, they compare the product characteristics against the technical properties of the available resources. Second, they analyse the emerging combined capabilities of the resources that will be connected. Third, they make sure that these resources have compatible interfaces. This search and filtering process is called here as ‘capability matchmaking’.

Since there is no standard, vendor neutral way to describe the resources in these catalogues, the comparison of alternatives from various resource providers may be difficult. Depending on the system complexity, the number of required resources might number in the thousands. The cumbersome and slow search and filtering activity limits the number of resource alternatives that may be considered. This limit means that better solutions might be unintentionally neglected as the designer favours their former solutions, which might be sub-optimal – if even that – for the given task. Thus, the system design and reconfiguration planning should be supported by new intelligent decision support tools that reduce the time and effort needed for these design and planning activities.

A key enabler of smart manufacturing is the virtualization of physical assets of the manufacturing, namely, resources and products (Lu and Xu 2018; Thoben, Wiesner, and Wuest 2017). In the context of capability matchmaking, such information models

CONTACT Eeva Järvenpää  eeva.jarvenpaa@tuni.fi  Faculty of Engineering and Natural Sciences, Automation Technology and Mechanical Engineering, Tampere University, Korkeakoulunkatu 6, Tampere 33720, Finland

© 2022 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.

This is an Open Access article distributed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives License (<http://creativecommons.org/licenses/by-nc-nd/4.0/>), which permits non-commercial re-use, distribution, and reproduction in any medium, provided the original work is properly cited, and is not altered, transformed, or built upon in any way.

should allow resource vendors to describe the functionality of their offerings comparably and system designers to make a match between product requirements and resource capabilities. Formal ontologies and other Semantic Web technologies have become popular solutions for addressing resource virtualization and representing other heterogeneous production-related information (Jardim-Goncalves, Grilo, and Popplewell 2016; Leitão, Walter Colombo, and Karnouskos 2016). For instance, the recent review by Yahya, Breslin, and Intizar Ali (2021) showed that several ontologies have been developed recently to support smart manufacturing and the Industry 4.0 paradigm. In the context of distributed intelligent systems, such as agent-based or service-oriented systems, ontologies play a key role, as they provide a shared, machine-understandable vocabulary for communication of domain knowledge among dispersed actors (Leitão, Colombo, and Karnouskos 2016).

An inherent activity during the resource search and selection is the generation of resource combinations that, together, match the processing requirements of the product. The capabilities of the production systems originate from the tool and device level. Especially in the case of modular and reconfigurable plug-and-produce production systems, these devices can be organized into various configurations. Thus, the formal resource models and the capability matchmaking system should automatically infer the combined capability information based on the capability description of these individual devices. Pure OWL (Ontology Web Language) does not provide solutions for making such inferences and assertions of new instances and their property values (The OWL Working Group 2004; Meditskos et al. 2013). Therefore, the OWL-based ontology needs to be enriched with semantic rules and supported with external software for enabling the automatic inference of capability and resource combination information.

In their earlier works, the authors have presented a Manufacturing Resource Capability Ontology (MaRCO), which is a unique OWL-based information model for describing the capabilities of manufacturing resources (Järvenpää et al. 2019a). This article will describe the concept and implementation of the capability matchmaking approach and software. It aims to support production system design and reconfiguration planning by providing automatic means for

finding alternative system configuration suggestions to product requirements from large search spaces. While earlier publications concentrated on describing the information models or the matchmaking approach in general, the specific contribution of this article is the detailed explanation of the usage of SPIN (SPARQL Inferencing Notation) rules during matchmaking. These rules 1) automatically infer and assert the parameters of combined capabilities based on the parameters of the lower-level capabilities originating from individual resources, and 2) compare the product requirements with this inferred information to find resources matching with the product requirements.

The article is organized as follows. First, [Section 2](#) will discuss the background and limitations of the related works to highlight the contributions of this work. [Section 3](#) will introduce the overall capability matchmaking concept and related OWL-based information models. In [Section 4](#), the matchmaking process will be introduced and continued by detailed examples of the rules used during matchmaking. Then [Section 5](#) describes the implementation of the capability matchmaking software. The approach will be validated by illustrating the matchmaking steps and results from a case study with real industrial data in [Section 6](#). [Section 7](#) analyses and discusses the results and impact of the presented approach, and finally, [Section 8](#) concludes the article.

2. Background and limitations of existing approaches

Extensive literature reviews on capability and resource models appeared in the authors' earlier research works (Järvenpää et al. 2019a) and will not be repeated here. Köcher et al. (2020) summarized that the earlier research approaches on capability or skill descriptions fell into two categories: first, to contributions focusing on formal models, such as ontologies to create descriptions that can be used as a shared vocabulary or for reasoning purposes; and second, to works, which focus on the plug-and-produce production environment, where skills are used to encapsulate machine functionalities and are directly used to control the execution of the processes. The work presented in this article falls into the first category. The execution of the processes is out of its scope.

In the context of Cloud Manufacturing, researchers have proposed many semantic ontology-based descriptions for the service description (Luo et al. 2013; Yuan, Deng, and Chaovalitwongse 2017; Lu, Wang, and Xu 2016). Lu and Xu (2017) also presented a service composition and mapping approach, as well as Jena rules which compared the service request with the offering. However, although several techniques for service description and resource virtualization have appeared in the literature, the current research has shown more conceptual solutions than practical implementations of the capability or service matchmaking. Furthermore, based on the earlier review by the authors (Järvenpää et al. 2019a), the existing resource and capability models do not consider the combined capabilities of multiple co-operating resources. Consequently, they do not include mechanisms for automatic reasoning of the combined capability parameters.

In order to perform capability matchmaking, also the product requirements need a formal description. The skill-based approach has been utilized in many studies to define both the processing requirements of a product and functionalities provided by the resources. These works, for instance (Backhaus and Reinhart 2017; Pfrommer et al. 2014; Bengel 2009) have often attempted to enable autonomous setup and execution of production tasks. The same skill model has been used for both product and resource representation. This allows simple matchmaking between the product requirements and resource offerings but forces the designer to represent the products in terms of specific skills it requires, rather than describing the product characteristics and processes in a resource-independent way. In this work the goal is to avoid such resource-centric representation of product requirements and separate the product description from the resource description.

The literature discusses several works utilizing semantic rules to extend the reasoning abilities of pure OWL-ontologies. Ameri and McArthur (2014) presented an idea comparable to capability matchmaking. In their work, SWRL (Semantic Web Rule Language) was utilized for intelligent supplier discovery based on the services they offer. With SWRL, they were able to infer new capabilities that were not explicitly stated in the original service description and to classify concepts based on the given property values. However, SWRL cannot assert new instances

or property values to the ontology (Horrocks et al. 2004; Meditskos et al. 2013), which is a key functionality needed by the capability matchmaking pursued in this article. For instance, in the work of Ameri and McArthur (2014), inferring new capabilities meant that the capabilities were predefined instances in the ontology, and the rule inference established new relations between the supplier and these capability instances but did not create new named individuals.

SWRL has also been widely applied in other research works. For instance, Li et al. (2018) presented an ontology-based product design framework for manufacturability verification and knowledge reuse. They used SWRL to create inference and constraint rules between the design and manufacturing knowledge to provide design recommendations for the product designers during the design process. Sun, Ma, and Gao (2009) applied SWRL rules to store experts' design experiences, product configuration and variant rules as well as constraints, in order to provide routine design assistance for product configuration. Cao et al. (2019) used SWRL rules to identify defects in parts to support machine condition monitoring.

Efthymiou et al. (2015) presented an approach that utilizes inference rules written in Jena Rule language. Also, it uses similarity measurements that facilitate the manufacturing system design by automatically identifying past similar projects, which can then be the basis for the design of the new production line. Pintzos, Matsas, and Chrysosolouris (2012) presented an ontology representation for manufacturing performance indicators, which included calculation formulas for the value of specific performance indicators based on the value of other indicators. However, they didn't specify the implementation method for the rules, and it remains unclear if the calculation takes place through automatic inference or by specific software. Maleki et al. (2018) presented an ontology-based framework that integrates sensing systems and machine components to allow machine health monitoring and notifications when maintenance actions are needed. Their approach does not infer new knowledge to the ontology by rules, but the services are based on external software querying the ontology with SPARQL.

SPIN (SPARQL Inferencing Notation) is a semantic rule language that, among other features, provides the functionality of asserting new named individuals. Meditskos et al. (2013) used SPIN to perform temporal reasoning with context information and to assert new

named individuals in an application related to recognition of human activity. Aarnio, Vyatkin, and Hastbacka (2016) targeted industrial maintenance support. They utilized SPIN for situation rules in context modelling. Doulaverakis et al. (2017) presented an approach to model and execute clinical practice guidelines (CPG) using OWL-ontologies and SPIN rules. They demonstrated it through CPGs for arterial hypertension management. Other research publications utilizing SPIN are rare, and the authors have not found any application of SPIN comparable to the work presented in this article.

As a summary, the work presented in this article aims to overcome the limitations of existing approaches and will provide the following novel contributions: 1) formal information model and semantic rules, which allow automatic inference of combined capability information; 2) the ability to present the product requirements independently from the resource capabilities, but to find matches between these two; and 3) a detailed illustration on how to utilize SPIN in the context of capability matchmaking, to automatically infer and assert new knowledge from the manually asserted information.

3. Capability matchmaking and involved information models

The following subsections will introduce the aims and scope of capability matchmaking in general. Next is the introduction to the underlying OWL-based

information models used during matchmaking. Ontology Engineering Methodology (Sure, Staab, and Studer 2009) has been applied to construct these ontologies and the development process, including a detailed requirements definition, has been described in (Järvenpää et al. 2019a). This section will summarize the main points about the concept and models to provide a solid background for the reader to understand the remaining parts of this article.

3.1. Aims and scope of capability matchmaking

This work builds on the integrated product-process-system framework, which has been widely used in production- and manufacturing-related research (Rampersad 1994; Cutting-Decelle et al. 2007; Tolio et al. 2010), and extends it with the concept of capability (see Figure 1). Capability matchmaking aims to support the production system designers and reconfiguration planners by automating the search for feasible resources and resource combinations to particular product requirements from large resource spaces. The approach is implemented as a software component, which external design and planning systems can utilize as a service. The primary goal of matchmaking is to provide these systems with information about resources or resource combinations that can perform a specific process step. The detailed definition of the requirements for matchmaking from

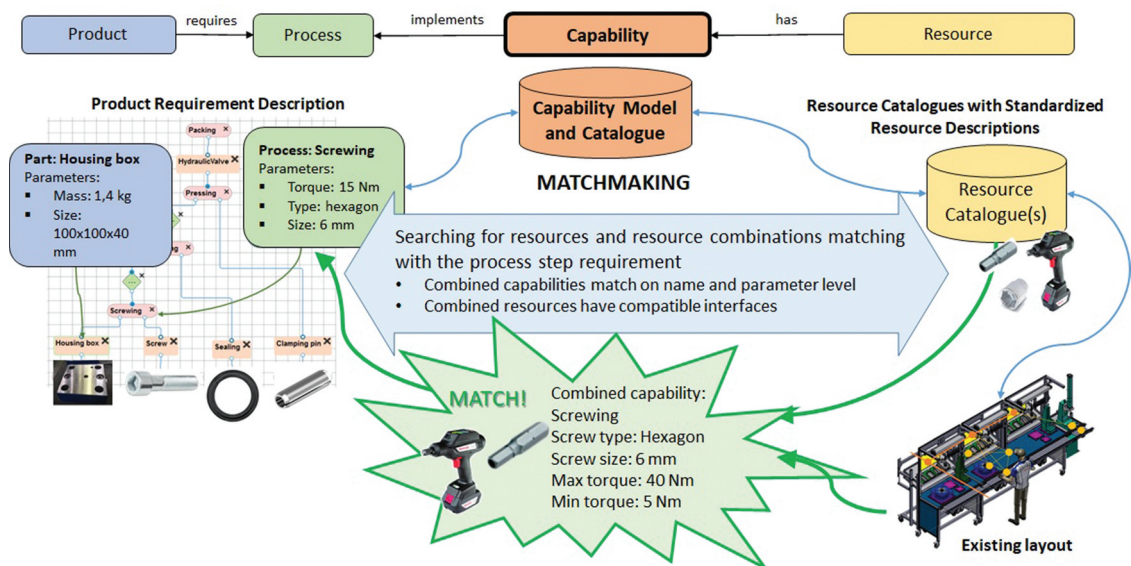


Figure 1. Basic idea of capability matchmaking.

the information model perspective and the ontology development process appeared in (Järvenpää et al. 2019a).

As opposed to other related works, this work aims to avoid resource-centric representation of product requirements and clearly separate the product description from the resource description. The required processes are organized in a taxonomic hierarchy, implying that the product requirements may be defined in different levels of detail. If the definition of a specific processing method is not critical from the product perspective, the process designer does not have to specify it exactly. For example, the designer can simply define that the joining of two parts requires a riveting process. The selection of the actual method of riveting can be left until after the potential results have been found by the matchmaking process. The method could be e.g. impact, radial or orbital riveting. Several existing process descriptions and standards were used as a basis for the development of the process taxonomy and associated capabilities, including the German standard DIN 8580, the EUPASS processes (Lohse, Maraldo, and Barata 2008) and the production taxonomy used in the CO2PEI-initiative (CO2PEI 2010).

The matchmaking system supports both greenfield (new system design) and brownfield (reconfiguration) scenarios (Järvenpää et al. 2019b). Figure 1 represents the conceptual idea of capability matchmaking. On the left-hand side, there is the product requirement description, containing the relevant product characteristics and processing requirements and on the right-hand side, there are the resource catalogues, containing the capability and interface descriptions of resources included in the matchmaking search space. Both the product requirements and the resource capabilities are presented as formal OWL-based ontologies, which will be introduced in the following section.

The matchmaking procedure and software have been developed to partly automate the design process traditionally done by human designer. Thus, it should mimic the manual process. In the greenfield scenario, the matchmaking system searches for suitable resources through the given resource catalogue(s) and creates resource combinations that can perform the process step requested by the product. Each resource has its own capabilities, and when the resources are combined with other resources,

combined capabilities emerge. These combined capabilities and their parameters need to be inferred based on the capability descriptions of the single resources involved in the combination. While creating these resource combinations, the matchmaking system checks the compatibility of the resource interfaces. Finally, it checks that the parameters of the capabilities match the parametric requirements of the requested process steps. In the brownfield scenario, the capabilities existing on the current layout are matched against the product requirements in a similar fashion.

3.2. *Involved information models*

As stated earlier, capability matchmaking relies on formal ontological descriptions of product requirements and capabilities of manufacturing resources. The developed OWL-based (Web Ontology Language) information models have been introduced in detail in (Järvenpää et al. 2019a, 2018b, 2017, 2018a; Siltala, Järvenpää, and Lanz 2018). The import structure of these models appears in Figure 2, and the models are available to download from (Järvenpää, Siltala, and Hylli 2019).

The Matchmaking Ontology Model, importing the Product and Resource Models (see Figure 2), is used to perform the capability matchmaking. Figure 3 presents a limited view of the Matchmaking Ontology. It shows its main classes and relations, including those inherited from the imported models. The boxes in the figure represent the classes of the ontology, while the inheritance arrows indicate the subclass hierarchies. The association arrows specify object properties, which model the relations between the instances belonging to those classes. In addition, datatype properties characterize the classes, but these are not in the figure. Similar colours indicate the model (Figure 2) to which the classes (boxes in Figure 3) belong. For instance, 'Device' is a class in the Resource Model. Also, the prefix in the class name indicates the namespace (i.e. the model) from where it originates (e.g. 'rm' refers to the Resource Model, as shown in Figure 2). The same prefixes are used when classes are referred to in the following text and rule examples.

The Resource Model ontology (Järvenpää et al. 2019a) describes the available manufacturing resources and their characteristics, such as interfaces

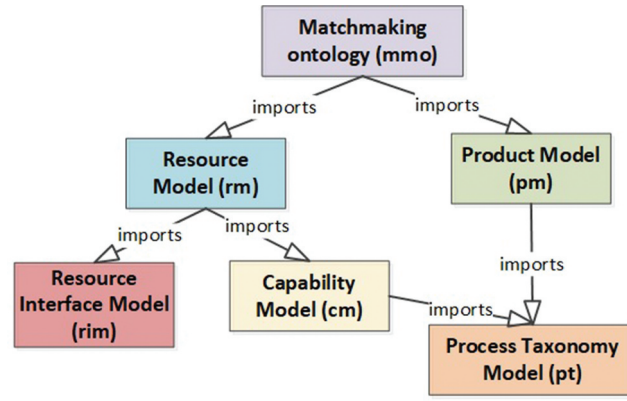


Figure 2. Information models used for capability matchmaking. (Modified from Järvenpää et al. (2019b) by adding the ontology namespace acronym to the brackets.).

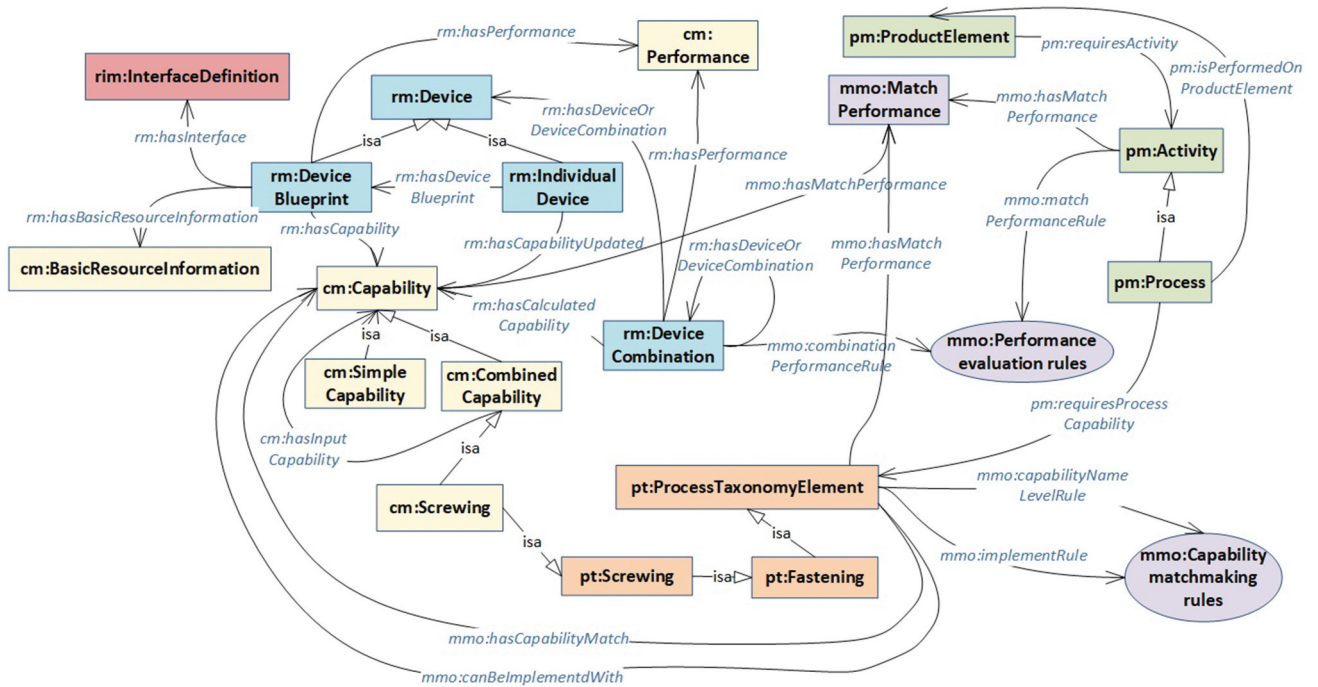


Figure 3. Simplified view of the matchmaking ontology. (Modified from Järvenpää et al. (2019b) by adding namespace definition in front of the class and property names and reorganizing the elements to improve readability).

and capabilities. It can also describe systems, that are aggregated from multiple resources. Class `rm:Device` (Figure 3) is used for modelling the machine and tooling resources. Human workers belong to another class that is not in the figure. The focus here is on the class descriptions for classes `rm:DeviceBlueprint`, `rm:IndividualDevice`, and `rm:DeviceCombination`. These classes link the capabilities to both catalogue devices and actual devices existing on the factory floor. Therefore, they form the very core of the developed model. They enable the emergence of the combined capabilities by modelling the combinations of the

devices. The Resource Model imports two other ontologies, namely the Resource Interface Model and Capability Model. The Resource Interface Model (Siltala, Järvenpää, and Lanz 2019b) is used to give a formal description of the resource interfaces. With this information it is possible to identify whether the interfaces of multiple resources are compatible and if the resources can be connected from their interface perspective.

The resource functionalities and their related parameters are formalised by the Capability Model (Järvenpää et al. 2019a). This model defines simple

and combined capabilities and formalises their relationships through the *cm:hasInputCapability* object property. As an example, a robot can have the simple capability 'Moving', and similarly, a gripper can have simple capabilities 'Grasping' and 'Releasing'. If a robot and gripper are combined, they can have combined capabilities 'Pick and Place' and 'Transporting'. The instances of *cm:Capability* are linked to the instances of *rm:Device* through the *rm:hasCapability* object property. The formalised relations between the simple and combined capabilities allow computer programs to form potential resource combinations having specific combined capabilities by utilizing information queried from the ontology by SPARQL. SPARQL is a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework (RDF) format and OWL ontologies (W3C SPARQL Working Group 2013). The following section will discuss the rules that allow automatic inference of the parameters of the combined capabilities. The *rm:hasCalculatedCapability* object property links this combined capability information to the specific device combination.

The Capability Model imports another ontology, called the Process Taxonomy Model. This model categorizes various manufacturing and assembly processes in a hierarchical structure. It is a pure taxonomy, without any properties. In the Capability Model, the different sub-classes of *cm:Capability* are linked to the sub-classes of *pt:ProcessTaxonomyElement* according to what kind of process they can provide. This linkage is implemented as a direct sub-class (is-a) relationship between the classes of *cm:Capability* and *pt:ProcessTaxonomyElement*. For instance, *cm:Screwing* is a sub-class of the *pt:Screwing*. Due to the class inheritance, instances of capability *cm:Screwing*, will also become instances of process *pt:Fastening*. Thus, the process taxonomy links the capabilities to the upper levels in the process hierarchy, e.g. the 'Milling' capability will be automatically classified as a 'Material Removing' process in the taxonomy.

The primary purpose of the Product Model ontology is to represent the processing requirements of the product in a manner by which these requirements can be matched against the resource capabilities. The Product Model was introduced in detail in (Järvenpää et al. 2018b). The Product Model describes the parts and their basic characteristics, subassemblies and

their contained parts, the processes of the parts and subassemblies, the capability requirements related to the processes, and the sequence of the processes. The Product Model imports the same Process Taxonomy as the Capability Model, and it models the product's processing requirements as instances of the *pt:ProcessTaxonomyElement* subclasses. For example, if the product requires a screwing process, this requirement is modelled as an instance of the taxonomy class *pt:Screwing*. This link is established through the *pm:requiresProcessCapability* object property between the instances of the *pm:Process* class and the instances of the *pt:ProcessTaxonomyElement* class. In the Product Model, the parametric requirements related to the processes are modelled as property restrictions of the *pt:ProcessTaxonomyElement* subclasses. An example of such a parameter may be the minimum torque required for screwing.

Since the publication of (Järvenpää et al. 2018b), the Product Model has added a few new properties to support the matchmaking software implementation. For instance, the model includes properties to indicate whether automatic matchmaking should be performed. The target is to allow full description of the product to be used, without forcing the matchmaking for each process step, thus lightening the matchmaking process and the associated matchmaking result. For instance, if it is predefined that the process will be performed manually, it can be omitted from the automatic matchmaking. There are two ways to limit matchmaking. First, the *pt:ProcessTaxonomyElement* class has a property *pm:matchmakingRequired*, which gets Boolean values. This property will indicate whether automatic matchmaking should be performed. Second, if a *pm:Activity* instance does not refer anywhere by the *pm:requiresProcessCapability* relation, then that instance will be naturally left out from the matchmaking. The first option provides a more dynamic process to change the input data for various matchmaking scenarios by just changing the Boolean value, while the second option requires less modelling effort, but is less flexible for different matchmaking scenarios.

In addition to importing the two ontologies, Product Model and Resource Model, the Matchmaking Ontology includes a few additional object properties, as illustrated in Figure 3. These properties are *mmo:hasCapabilityMatch* and *mmo:canBeImplementedWith*, which link the capability requirements (instances of

pt:ProcessTaxonomyElement subclasses) with the cm: Capability instances. This linkage establishes dynamically when the capability matchmaking rules infer new knowledge to find the matches. The *mmo:hasCapabilityMatch* indicates that the capability matches the requirement on the capability concept name level, while the *mmo:canBeImplementedWith* indicates a detailed match, i.e. that the capability parameters also match with the requirement.

4. Capability matchmaking procedure and rules

In addition to the information models, matchmaking requires several rules. The development of the rule base began by defining what kind of rules are needed in various phases of the system design process. Manufacturing engineering domain knowledge, including manufacturing engineering handbooks, resource provider datasheets, and discussions with manufacturing engineers were used to define the content of the rules. Even though a case-based approach was used in the development and testing of these rules, the rules were defined to be generally applicable to all cases and not just for the specific use cases in testing and validation. The following subsections will introduce the overall matchmaking procedure and the semantic rules used during matchmaking.

4.1. Matchmaking procedure

The overall capability matchmaking procedure consists of multiple steps and viewpoints, which require specific rules to perform the needed reasoning process. Figure 4 illustrates these viewpoints and rules. First, matchmaking must generate new resource

combinations for the different capability requirements. Second, matchmaking needs to check whether the capability of the resource matches with the requirements of the product. When new resource combinations are generated, the matchmaking must consider the capability of the combined resources and whether the resources can be combined from their interface perspective. Combined capability rules calculate the parameters of the combined capabilities based on the capabilities of the resources involved in the combination. The interface matchmaking rules check the compatibility of the interfaces between the intended combined resources. Finally, when the resource combinations have been created and their combined capabilities have been calculated, these combined capabilities need to be compared to the characteristics and requirements of the product. For this purpose, capability matchmaking rules have been defined. Furthermore, as the design systems consuming the matchmaking results benefit from getting some information about the performance of the found resource combinations, i.e. a roughly estimated duration of the process step with the suggested match, simple rules for performance evaluation have also been defined. This evaluation is not at the core of the capability matchmaking approach, and therefore, it is marked with dashed lines in Figure 4. The authors consider the performance evaluation to be the task of the design and planning system. It is dependent on, e.g. the layout of the system and spatial relations and limitations between the resources, which cannot be analysed in detail with the presented ontologies.

Figure 5 gives an overview of the matchmaking procedure, including both the brownfield (reconfiguration) and greenfield (new system design)

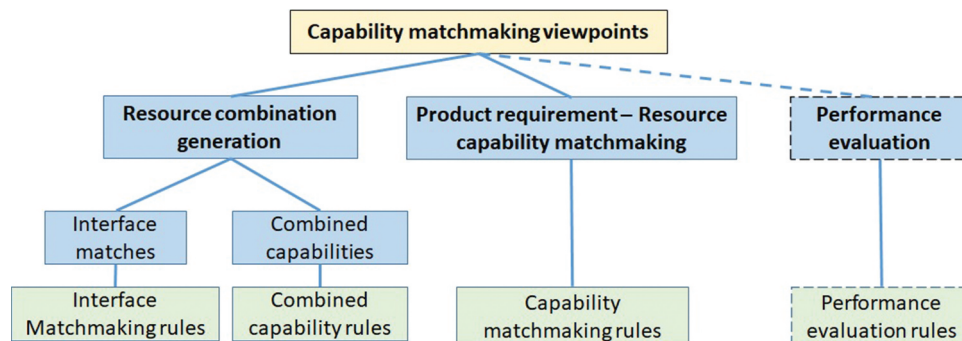


Figure 4. Matchmaking viewpoints and rules.

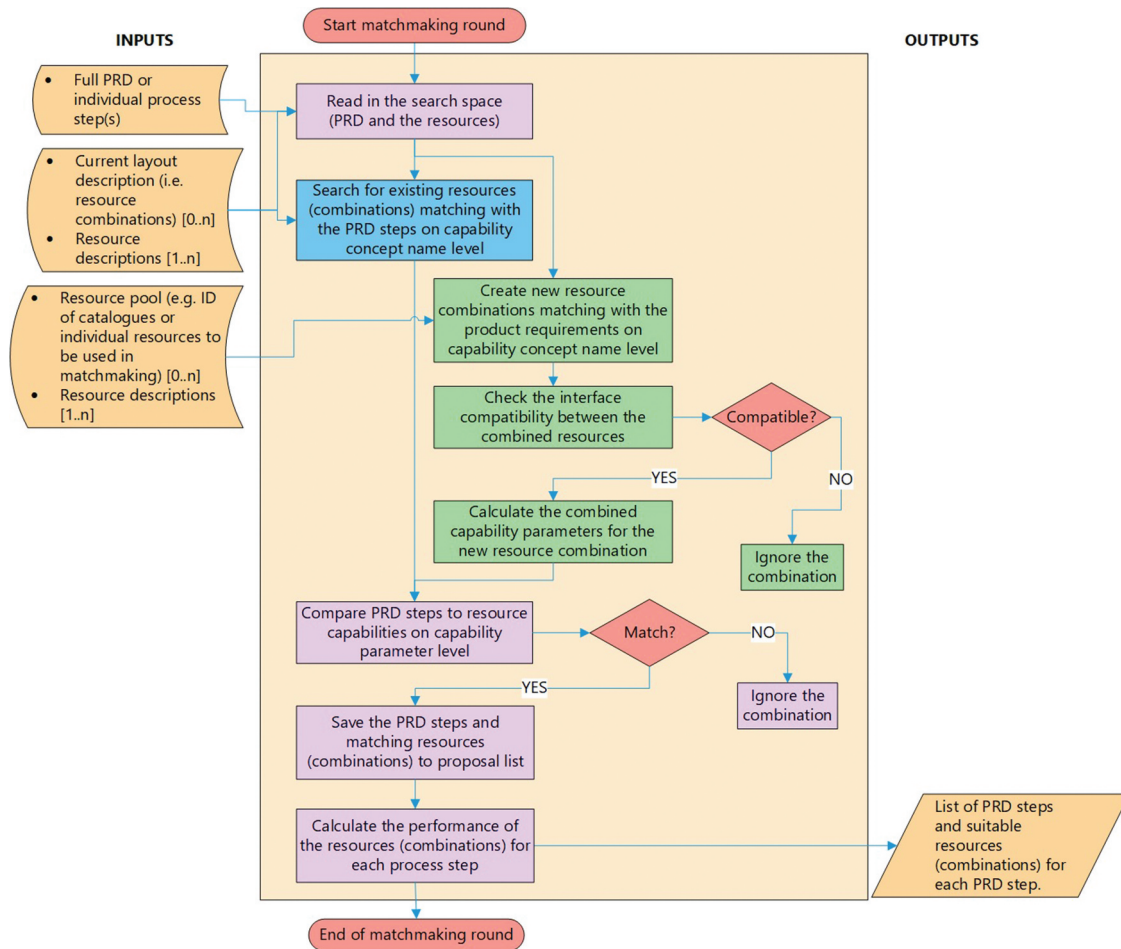


Figure 5. Capability matchmaking procedure on a high level.

viewpoints. As an input, the matchmaking needs to receive the search space used for the matchmaking. It includes the Product Requirement Description (PRD) and the Resource Pool. The Resource Pool consists of the selected resources from the resource catalogue(s) that the system designer wants to include in the matchmaking search space. The input differs, depending on the scenario. In the brownfield scenario, the existing layout description is also an input. The System Layout consists of the description of the existing resource combinations of the current system, modelled in MaRCO format. External design and planning tools, which want to utilize the matchmaking service, will provide these inputs for matchmaking.

The capability matching algorithm takes the capability requirements (i.e. pm:Activity instances) and matches them with the existing capabilities on the current layout or creates new resource combinations that match with the requirements. The matchmaking result is provided as an output. It includes the IDs of

the resources and resource combinations matching with each process step defined in the PRD. The output is independent of the scenario. This output is provided to the external design tools, where the designer will utilize it to make decisions about the resource selection and system configuration based on their valued criteria. The matchmaking procedure can be run in an iterative manner. Thus, several matchmaking rounds can be run with different inputs, depending on the design strategy.

4.2. Rule implementation

The combined capabilities and their parameters should be automatically defined and saved to the Resource Model ontology without the need to manually fill in the parameters. Similarly, the link between requirements and matching capabilities should be automatically inferred and asserted within the Matchmaking ontology. These tasks require semantic

rules. As pure OWL does not provide solutions for making required inference and assertions of new instances and their property values, nor to perform complex arithmetic operations (Meditskos et al. 2013), SPIN (SPARQL Inferencing Notation) is used for rule implementation. SPIN is a W3C member submission that was, at the time of the rule implementation, the de-facto industry standard to represent SPARQL rules and constraints on Semantic Web models (SPIN Working Group 2017). In research the more commonly used SWRL (Semantic Web Rule Language) was not considered appropriate, as it does not support the assertion of new named individuals. Furthermore, as discussed by Bassiliades (2018), despite its longevity, SWRL had never achieved a W3C recommendation status.

SPIN allows to link SPARQL queries directly to the class definitions in the ontology. These queries capture constraints and rules which formalise the expected behaviour of those classes. The extra information created by these rules can be inferred by a SPIN compatible reasoner tool, such as SPIN API, and then be used in further SPARQL query execution (Knublauch 2016). Consequently, these SPARQL queries can be organized in an object-oriented manner, which makes the rules accessible and easy to maintain, extend, and share. With SPIN the rules can be represented and stored as SPARQL queries as a natural part of ontology knowledge in the same knowledge base (SPIN Working Group 2017). This is a clear advantage over SWRL, which stores rules as a flat list. In this work the combined capability rules are stored directly into the Resource Model ontology, while the matchmaking rules are stored in the Matchmaking Ontology, which make them commonly shared across the users.

SPIN offers the ability to calculate property values based on other property values. It can also be used to isolate a set of rules to be executed under certain conditions, e.g. to support incremental reasoning, to initialize certain values when a resource is first created, or to drive interactive applications. Furthermore, SPIN features a useful metamodeling capability, which lets users specify their own reusable SPARQL query templates and functions. The templates are parameterized SPARQL queries that can be customized by

instantiating them with the argument values of a new context. The SPIN function is a special kind of template query that returns only one result value, and it can be used as a part of another SPARQL query. Functions can be chained, meaning that functions may utilize other functions and so on (SPIN Working Group 2017).

The rules needed during the matchmaking procedure relate to the creation of the new resource combinations and comparing the product requirements to the resource capabilities. These rules were first defined in an informal textual format based on domain expert knowledge. After that, they were implemented in SPIN with the help of the TopBraid Composer Semantic Web editor. Many rules share similarities, and thus the metamodeling features of SPIN, i.e. functions and templates, were utilised. The following subsections discuss these rules in detail, including examples of the usage of templates and functions.

4.2.1. Resource combination generation

As visualised by Figure 4, when two or more resources are to be combined, two aspects need to be considered: 1) the combined capability that they can produce and 2) the compatibility of the resource interfaces. This section will only discuss the rules used for inferring the combined capability parameters, because the interface compatibility check is described in detail in other publications (Siltala, Järvenpää, and Lanz 2019b; 2019a; 2021).

There are different ways how the combined capability parameters are formed, for example (Järvenpää et al. 2018a):

- Directly inheriting a parameter value from one of the involved lower-level capabilities. For instance, in the case of 'Transporting' combined capability, formed by 'Moving' and 'FingerGrasping' capabilities, the value of the *dof* (degrees of freedom) property is the same as the value of *dof* of the 'Moving' capability.
- Calculating the parameter value by arithmetic operations from two or more involved lower-level capabilities. For instance, the value of the

payload property of ‘Transporting’ capability is the value of the *payload* property of the ‘Moving’ capability subtracted by the value of the *mass* property of the gripper or the value of the *payload* property of the ‘FingerGrasping’ capability. The smaller value dominates.

- Defining the parameter value by comparing the values of the involved lower-level capabilities and selecting the maximum or minimum value, depending on the specific capability parameter. For example, the *itemSize_max* property, which is used for multiple capabilities, gets its value from the capability that has the lowest value.

The ‘lower-level capability’ refers to capabilities that are on a lower level in the *part_of* hierarchy. The term ‘simple capability’ is not appropriate because the combined capabilities can be composed of other combined capabilities, not just simple ones.

The combined capability rules are linked to the related `cm:CombinedCapability` subclasses in the Resource Model ontology. For each parameter of a combined capability, there is a SPIN rule that links to that specific capability class in the ontology. Common for all these rules is that they must retrieve at least one of the lower-level capabilities that produced the combined capability. In addition, they must retrieve one of the parameters of that lower-level capability. For retrieving the specific lower-level capability instance of interest, a SPIN function, *getPartCapability*, was created (see Rule example 1). The function can be used in another SPARQL query by giving specific values to its two arguments (*?arg1*, which refers to the instance of `cm:CombinedCapability` and *?arg2*, which refers to the capability class, i.e. any subclass of `cm:Capability`) from outside, as will be later shown in the Rule example 2. Line #3 retrieves the *?deviceCombination*, which is linked to the given combined capability instance (*?arg1*) through the *rm:hasCalculatedCapability* object property. Line #4 retrieves a device (*?device*), which is linked to the *?deviceCombination*. Line #5 retrieves the *?blueprint* of the specific device (*?device*), and line #6 retrieves the capability instance (*?capability*) associated with that *blueprint*. Finally, the SELECT clause (row #1) returns

the found capability instance (*?capability*) if it is of a type defined by the *?arg2* (row #7). For more information about SPARQL and SPIN syntax, please refer to Knublauch (2013) and the W3C SPARQL Working Group (2013).

SPIN Rule example 1: Function for retrieving the capability instance of interest – *getPartCapability* (arguments *arg1* and *arg2* highlighted)

```
#1 SELECT ?capability
#2 WHERE {
#3   ?deviceCombination rm:hasCalculatedCapability ?arg1.
#4   ?deviceCombination rm:
#5     hasIndividualDeviceOrDeviceCombination)* ?device .
#6   ?device rm:hasDeviceBlueprint ?blueprint .
#7   ?blueprint rm:hasCapability ?capability .
#8   ?capability a ?arg2 .
}
```

As described earlier, the first and easiest way to infer and assert a combined capability parameter is to inherit it from a lower-level capability instance. For example, the *dof* of ‘Transporting’ is the same as the *dof* of ‘Moving’, or the *torque_max* of ‘Screwing’ is the same as the *torque_max* of the ‘SpinningTool’ capability. In both cases, the logic of the rule is the same, but just the type of the capability and the parameter are different. For this kind of situation, a SPIN template called *inheritCapabilityParameter* (see Rule example 2) has been created. A template has a body consisting of a SPARQL query and arguments (*?capability*, *?parameter*), whose values will be replaced in the query. The argument *?capability* refers to the capability class (subclass of `cm:Capability`) from which the value of the parameter, referred by the argument *?parameter*, will be inherited.

CONSTRUCT and WHERE clauses are used for those rules which infer and assert new parameter values to the ontology. The CONSTRUCT defines a set of triple patterns to add to the underlying resource model upon the successful pattern matching of the triple patterns in the WHERE clause. The CONSTRUCT part (lines #1–3) defines how the query results are utilized to modify the ontology: give the parameter value to the combined capability instance. On line #5, we use the *getPartCapability* function, defined in Rule example 1, to retrieve the instance of the given lower-level capability (*?capability*) related to the combined capability instance (*?this*). Line #6 checks whether the function returned a result. If not, nothing happens.

On line #7, the result is assigned to a different variable (*?resourceCapability*). Using the *?result* variable directly would lead to issues because, if it is unbound after the function call, it can bind to other capability instances. Finally, line #8 gets the value (*?value*) of the desired parameter (*?parameter*) from the lower-level capability instance, which is then saved by the CONSTRUCT clause to the higher-level capability instance (*?this*) (lines #1–3).

SPIN Rule example 2: Template for inheriting the capability parameter – *inheritCapabilityParameter* (arguments *?parameter* and *?capability* highlighted)

```
#1      CONSTRUCT {
#2      ?this ?parameter ?value .
#3      }
#4      WHERE {
#5      BIND (:getPartCapability(?this, ?capability) AS ?result) .
#6      FILTER bound(?result) .
#7      BIND (?result AS ?resourceCapability) .
#8      ?resourceCapability ?parameter ?value .
#9      }
```

The actual combined capability rule is then an instance of this template, in which the arguments are given concrete values. For example, when the *inheritCapabilityParameter* is used to create the combined capability parameter rule for defining the *torque_max* of ‘Screwing’, the argument *?capability* is replaced with ‘cm:SpinningTool’ and *?parameter* with ‘cm:torque_max’. The SPIN rule engine, which executes the rules, will use the template and create the rule with given concrete values for the arguments.

Some rules are identical for different capabilities. For example, the payload rules of ‘Transporting’ and ‘PickAndPlace’ are identical, and therefore, the same SPIN rule (Rule example 3) can be copied and applied in each case. The CONSTRUCT part (lines #1–3) creates and assigns the payload value (*?payload*), determined by the rule, to the combined capability instance (*?this*). Line #5 utilizes the *getPartCapability* function to retrieve the instance of the ‘cm:FingerGrasping’ capability involved with the combined capability instance (*?this*). Line #7 assigns the result to the variable *?fingerGrasping*. Line #8 retrieves the device instance, which is associated to the given capability *?fingerGrasping*, and line #9 retrieves its basic resource information. Lines #10 and #11 retrieve the mass and payload of the gripper. Lines #12–15 similarly retrieve the payload of the ‘Moving’ capability. Line #16 subtracts the gripper’s mass from the payload of the

‘Moving’ capability and binds it to the variable *?alternative*. Finally, line #17 compares the result from the previous line to the gripper payload and defines the lesser value as the *?payload* of the combined capability *?this*.

SPIN Rule example 3: Calculating the payload of ‘Transporting’ and ‘PickAndPlace’ capabilities

```
#1      CONSTRUCT {
#2      ?this cm:payload ?payload .
#3      }
#4      WHERE {
#5      BIND (:getPartCapability(?this, cm:FingerGrasping) AS ?result) .
#6      FILTER bound(?result) .
#7      BIND (?result AS ?fingerGrasping) .
#8      ?gripperDevice rm:hasCapability ?fingerGrasping .
#9      ?gripperDevice rm:hasBasicResourceInformation ?info .
#10     ?info cm:mass ?mass .
#11     ?fingerGrasping cm:payload ?graspingPayload .
#12     BIND (:getPartCapability(?this, cm:Moving) AS ?result2).
#13     FILTER bound(?result2) .
#14     BIND (?result2 AS ?moving) .
#15     ?moving cm:payload ?movingPayload .
#16     BIND ((?movingPayload - ?mass) AS ?alternative) .
#17     BIND (IF((?alternative > ?graspingPayload), ?graspingPayload, ?alternative) AS ?payload) .
#18     }
```

The reasoner saves the inferred parameter values to the same ontology file with the initial instances. The value is linked to the *rm:DeviceCombination* instance by the *cm:hasCalculatedCapability* object property. This property is a subproperty of *cm:hasCapability*. Thus, when the matchmaking tries to search for devices with specific capability, the device combinations also linked to that capability by the *cm:hasCalculatedCapability* property are taken into consideration.

4.2.2. Product requirement – resource capability matchmaking

The Matchmaking Ontology contains SPIN rules for matchmaking, and properties and classes used by the rules. The capability matchmaking rules are attached to the subclasses of the *pt:ProcessTaxonomyElement*. These rules are executed in the matchmaking process after the combined capabilities and their parameters have been calculated for involved device combinations with the rules discussed in the previous section.

During the product requirement – resource capability matchmaking, there are rules for the following purposes:

- (1) Find capability name-level matches
- (2) Find parameter-level matches

These rules and their associated properties and classes are described in more detail below. Since these rules must be executed in the order listed above, they are associated with the ontology classes by different sub-properties of *spin:rule*, defined in the Matchmaking Ontology. The execution priority of these rules is then determined with the *spin:nextRuleproperty* property.

4.2.2.1. Find capability name-level matches. The first part of the matchmaking process is to find the resources that, on the capability name-level, could be used to perform the product manufacturing activity. The *pm:Activity* instances in the product requirement description refer to specific instances of the *pt:ProcessTaxonomyElement* subclasses. The capability instances defined in the Resource Model are also instances of specific subclasses of the *pt:ProcessTaxonomyElement* (see Figure 3). This relation creates a direct conceptual match between the product requirements and the resource capabilities. Thus, this first step of capability matchmaking is performed with one SPIN rule (Rule example 4), which is associated with the *mmo:capabilityNameLevelRule* property to the *pt:ProcessTaxonomyElement* class. For the given instance of *pt:ProcessTaxonomyElement* or, in this case, one of its subclasses (e.g. *pt:Screwing*), the rule will find all capabilities that implement the process i.e. its subclasses (lines #4–9). Then all instances of that *cm:Capability* are connected to the *pt:ProcessTaxonomyElement* instance with the *mmo:hasCapabilityMatch* property (lines #1–3). However, the Matchmaking software runs this rule only to the *pt:ProcessTaxonomyElement* instances which have the value ‘true’ for the *pm:matchmakingRequired* property.

SPIN Rule example 4: Finding match with capability name level

```

#1      CONSTRUCT {
#2          ?this:hasCapabilityMatch ?instance .
#3      }
#4      WHERE {
#5          ?this a ?class .
#6          ?capability (rdfs:subClassOf)+ ?class .
#7          ?capability (rdfs:subClassOf)* cm:Capability .
#8          ?instance a ?capability .
#9      }

```

The subsumption-based reasoning, related to various alternative processing methods, is handled by the Process Taxonomy. The reasoning ability of OWL allows a direct inference to be made that each instance saved to a specific *cm:Capability* class is also an instance of the *pt:ProcessTaxonomyElement* class, which was defined as the parent class of that specific capability class. Thus, if the product requires a riveting process, any resource providing riveting capabilities (e.g. orbital or radial) can be suggested as a match.

4.2.2.2. Find parameter-level matches. After the capability concept name-level matches are found, the actual matches can be determined by comparing the parameters of the manufacturing activity requirements and the parameters of the capability. This process uses various matchmaking rules attached to the *pm:ProcessTaxonomyElement* subclasses with the *mmo:implementRule* property. Each rule gets the *cm:Capability* instance connected to the *pt:ProcessTaxonomyElement* instance with the *mmo:hasCapabilityMatch* property and makes process- and capability-specific parameter comparisons. If they match, the *cm:Capability* instance is further connected to the *pt:ProcessTaxonomyElement* instance with the *mmo:canBeImplementedWith* property.

Since there are many similarities between the rules, many common comparisons have been implemented as SPIN functions that can be reused in multiple rules. These include, for example, comparisons of shape and size or torque. An example of a function for checking whether the torque range of the provided capability matches with the torque required by the product is presented in Rule example 5. It takes two arguments as inputs: *?capability*, which refers to an instance of *cm:Capability*, and *?taxonomy*, which refers to an instance of *pt:ProcessTaxonomyElement*. This function is utilized e.g. by the matchmaking rule for Screwing, illustrated in Rule example 6 (on row #12). Some of the capabilities may have only *cm:torque_max* defined, and some of the product requirements may define either exact, maximum, or minimum values for required torque. Therefore, the rule also needs to include these as optional parameters, as shown in lines #3–14. If the parameter does not exist, it is ignored by the ‘not

bound” part of the filter (#15–18), and the filter passes. If the parameter exists, it must comply with the rest of the filter conditions.

SPIN Rule example 5: Function for matching with torque parameters – *torqueMatch* (arguments *?capability* and *?taxonomy* highlighted)

```
#1  ASK WHERE {
#2    ?capability cm:torque_max ?max .
#3    OPTIONAL {
#4      ?capability cm:torque_min ?min .
#5    } .
#6    OPTIONAL {
#7      ?taxonomy pm:requiredTorque_max ?requiredMax .
#8    } .
#9    OPTIONAL {
#10     ?taxonomy pm:requiredTorque_exact ?requiredExact .
#11    } .
#12   OPTIONAL {
#13     ?taxonomy pm:requiredTorque_min ?requiredMin .
#14   } .
#15   FILTER ((!bound(?requiredExact)) || (?requiredExact ≤ ?max)) .
#16   FILTER (((!bound(?requiredExact)) || (!bound(?min))) || (?
#17     requiredExact ≥ ?min)) .
#17   FILTER (((!bound(?requiredMax)) || (!bound(?min))) || (?
#18     requiredMax ≥ ?min)) .
#18   FILTER ((!bound(?requiredMin)) || (?requiredMin ≤ ?max)) .
#19 }
```

Rule example 6 shows an example rule that finds capabilities that match with the required screwing process. The rule compares the type and size of the screw and the torque requirements with the parameters of the available capabilities (lines #10–12) and connects the requirement instance (*?this*) with the capability instance (*?instance*) by the *mmo:canBeImplementedWith* property (lines #1–3).

SPIN Rule example 6: Matchmaking for Screwing process parameters.

```
#1  CONSTRUCT {
#2    ?this mmo:canBeImplementedWith ?instance .
#3  }
#4  WHERE {
#5    ?this pm:screwType ?requiredType .
#6    ?this pm:screwDiameter ?diameter .
#7    ?this mmo:hasCapabilityMatch ?instance .
#8    ?instance cm:screwType ?instanceType .
#9    ?instance cm:screwSize ?size .
#10   FILTER (?requiredType = ?instanceType) .
#11   FILTER (?diameter = ?size) .
#12   FILTER mmo:torqueMatch(?instance, ?this) .
#13 }
```

The rules attached to the classes follow similar inheritance like other properties. Thus, the rules attached to a class apply also to the subclasses of this class. For instance, the rules defined for pt:

Feeding are also used on the subclasses of pt: Feeding. In addition, it is possible to add specific rules to the lower-level classes.

5. Software architecture for the matchmaking

The implementation of the Capability Matchmaking Software follows the principles of client-server architecture and the common layered model of service-oriented architectures. The layered architecture is illustrated in Figure 6 and presented earlier in (Järvenpää et al. 2019b). The main layers are the data model layer, data layer, business layer, and web service layer. The purpose here is not to present a novel software architecture, but to introduce the full architecture that makes available the novel functionalities provided by the modules on the business and data model layer. Thus, this section focuses on the activities taking place on the business layer, where the actual matchmaking activities occur, i.e. the internal functionality of the matchmaking system.

The data model layer contains the ontology models discussed earlier, while the data layer represents the actual data, i.e. instances used during matchmaking. The web service layer provides a back-end application, Matchmaking Service, which can be called by other design and planning software through the RESTful Web Service interface. As RESTful API supports both XML and JSON, the interaction format for inputs and outputs can be selected based on the requirements of the client software. The data model for inputs and outputs has been optimized for XML-structure. The web service and associated software modules are deployed and hosted on an Apache Tomcat server.

As discussed in (Järvenpää et al. 2019b), from the viewpoint of the matchmaking reasoning activities, the most important software packages in the architecture are the Capability Query Library (CQL) and the Matchmaker. The Matchmaker takes care of the management, sequencing, and execution of the matchmaking process for the incoming requests. Its constructor receives Product Model and Resource Model instance ontologies, between which the matchmaking should be performed. It also receives the matchmaking request, which determines the process steps that should be matched for and, in case

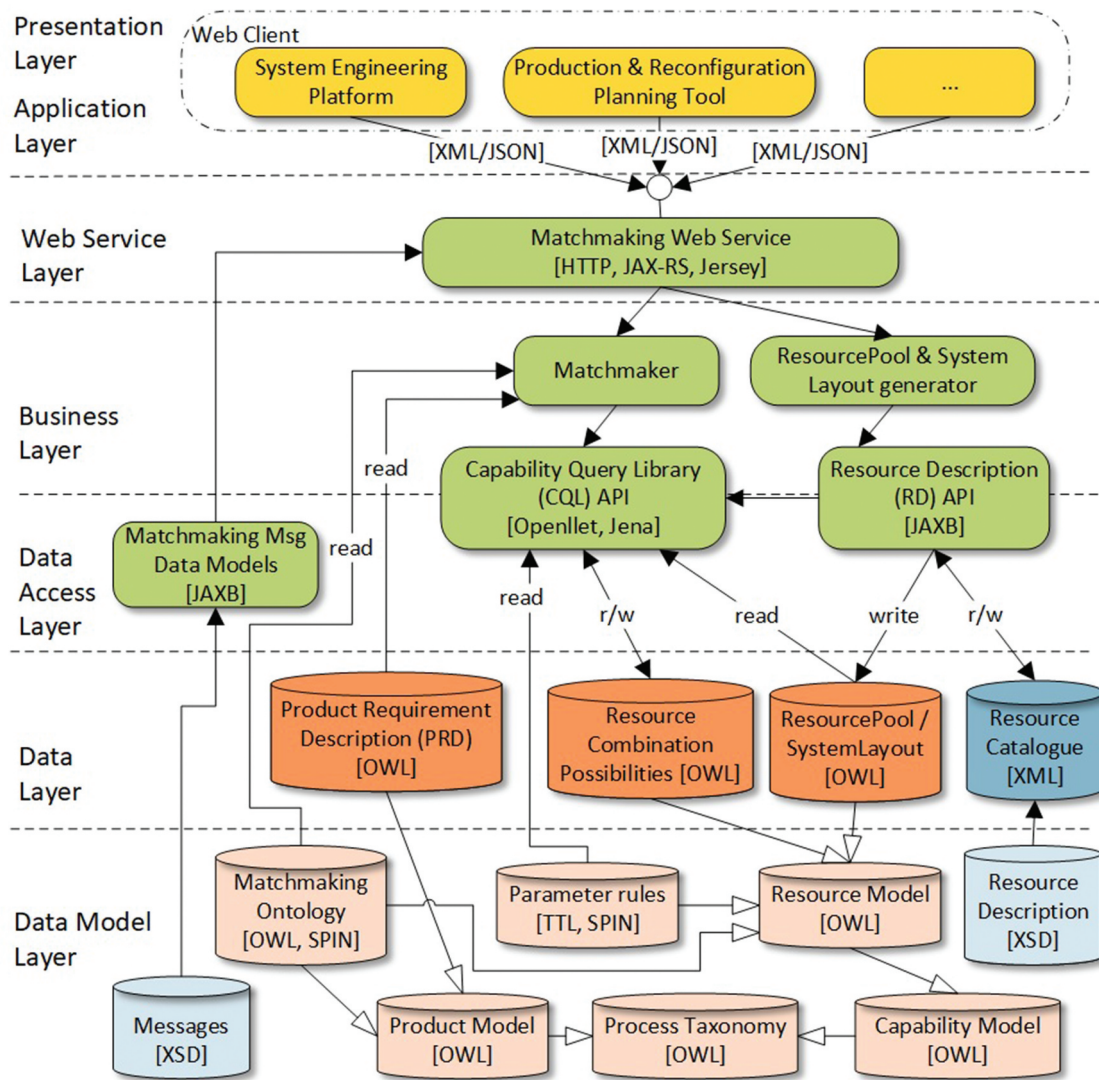


Figure 6. Overall software architecture of the capability matchmaking system, modified from Järvenpää et al. (2019b).

there is an existing system layout, whether it should be kept fixed (i.e. if new combination possibilities can be created from the resources in the layout or not). It creates resource combination possibilities, calculates combined capabilities and parameters for the resource combinations, checks the interface compatibility of the resources, executes matchmaking rules, and constructs the matchmaking result from the rule inferences. The Matchmaker calls the execution of the SPIN rules through the Capability Query Library (CQL).

The Capability Query Library (CQL) is a Java-based application that offers a Java API for other applications to work with the developed MaRCO model. In capability matchmaking, it is responsible

for running the rules needed for the resource combination generation and combined capability calculation. The CQL uses the open source Jena semantic web framework (Apache Software Foundation 2017) and Pellet reasoner (Sirin et al. 2007) for working with the ontology models. For the execution of the SPIN rules, the SPIN API is used, since Jena or Pellet themselves do not support SPIN. SPIN API is an open-source library that builds on top of Jena (Knublauch 2016).

Figure 7 illustrates the functionality of the CQL and Matchmaker during a matchmaking scenario. The defined combined capability SPIN rules, including the templates and functions used by the rules, are included

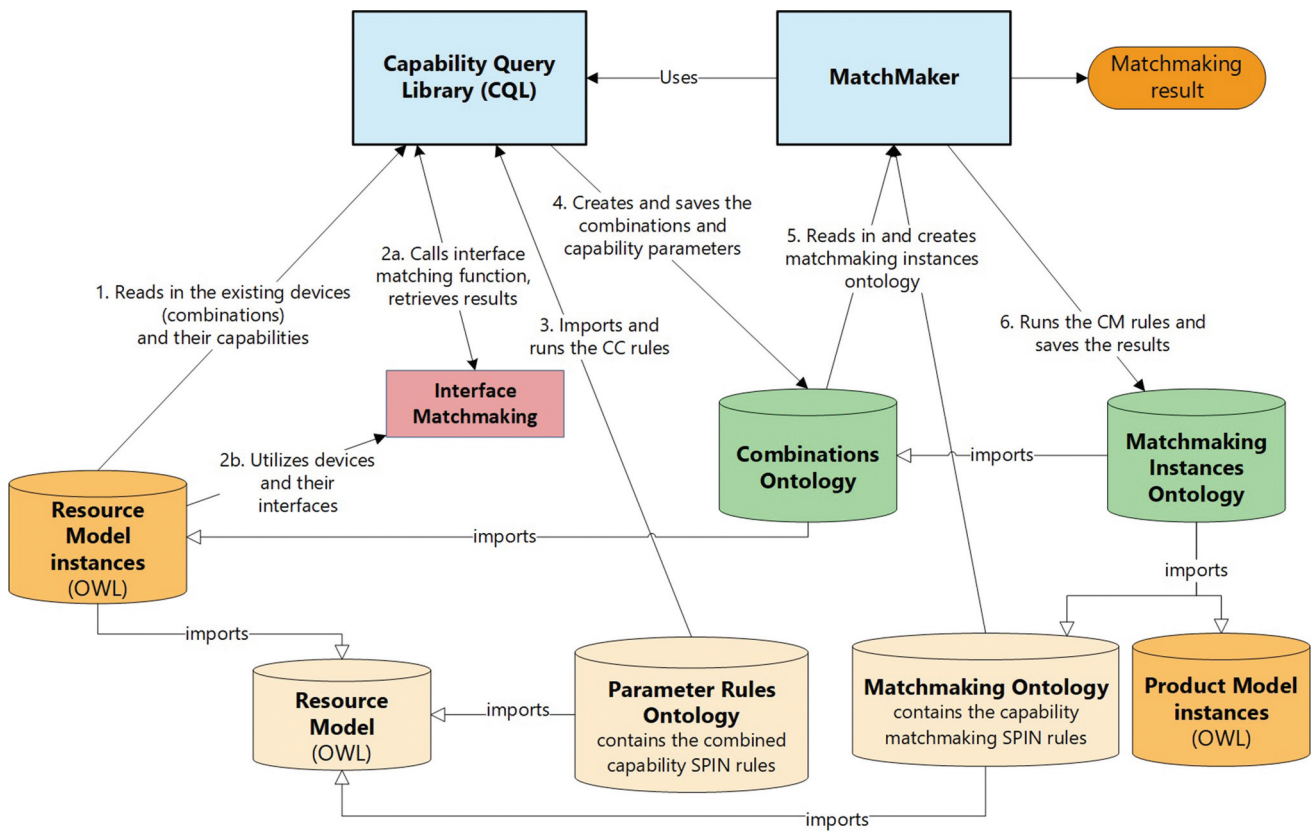


Figure 7. Functionality of the CQL and matchmaking process module during capability matchmaking.

in a separate Parameter Rules ontology used by the CQL. This ontology imports the Resource Model ontology. When defining the combined capabilities for resources, CQL reads in the Resource Model instances ontology and calculates the combined capabilities of the device combinations on a concept name level. This combined capability information is saved to a temporary ontology inside CQL (Combinations Ontology in Figure 7), which imports the original Resource Model instances ontology and the Parameter Rules ontology. The combined capability parameters are then inferred by SPIN API for the combinations, and this information is saved to the Combinations Ontology. If new device combination possibilities are required, CQL will generate combination possibilities for the required capabilities. The process can be given an external function for filtering the created combinations based on their interface compatibility. This same process is then used to calculate the combined capabilities and their parameters for these combination possibilities, and this information is then saved again to the temporary ontol-

ogy inside CQL. This generated ontology file will then be used by the Matchmaker for capability matchmaking. The Matchmaker executes the matchmaking rules and builds the matchmaking result by making various SPARQL queries to the ontology containing the information inferred by the rules and related product and resource information.

6. Use case and validation

This section illustrates an example use case and walks through the matchmaking process. The use case initially appeared in (Järvenpää et al. 2021). Secondly, the validation of the matchmaking results is discussed, followed by a discussion of the performance of the matchmaking system.

6.1. Example scenario of capability matchmaking

In the example scenario, the production system designer designs a new production system for a switch valve assembly process and utilizes the

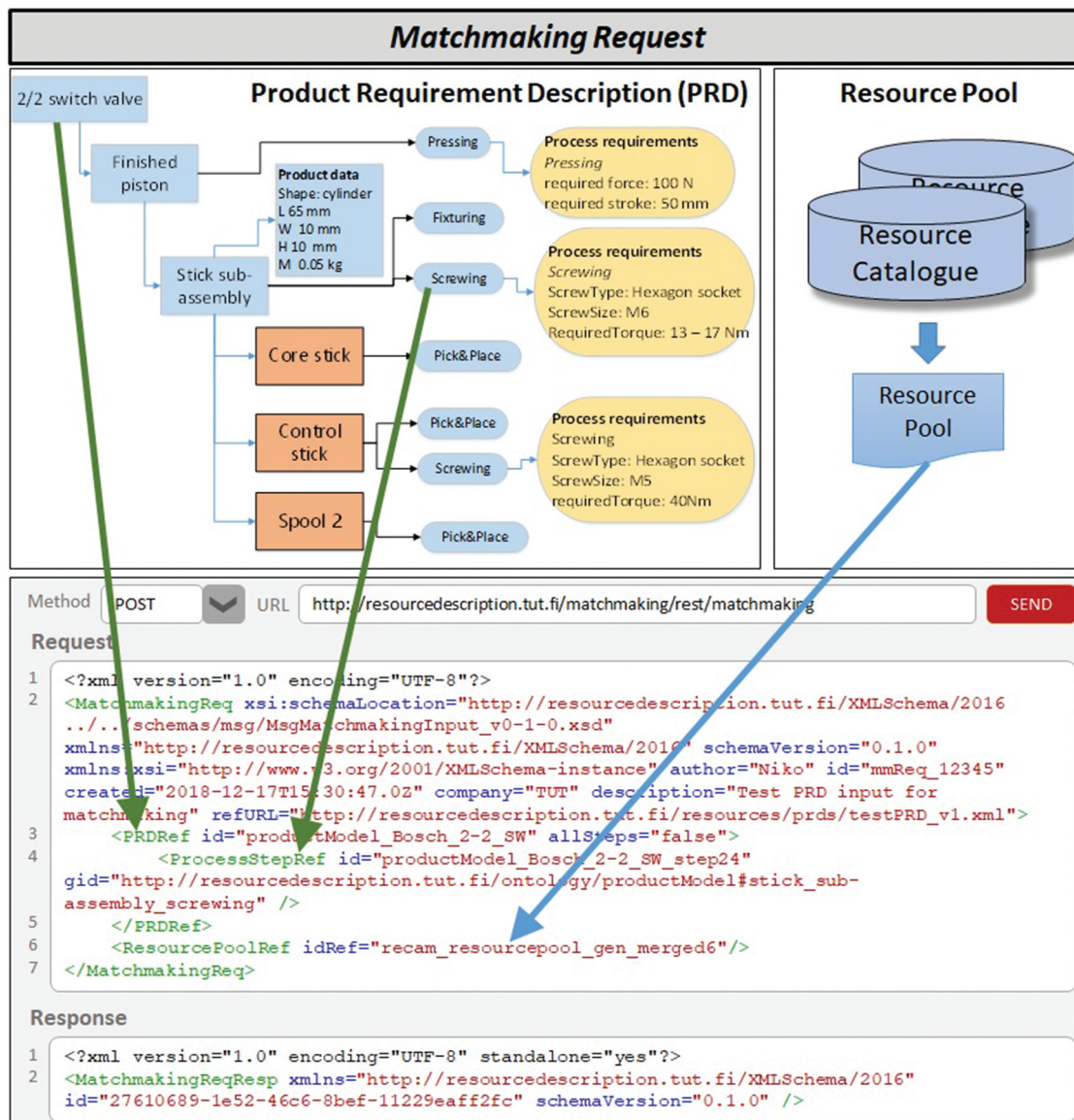


Figure 8. Example of matchmaking request.

capability matchmaking system to find feasible resources or resource combinations. Figure 8 demonstrates the inputs provided by the client system to the matchmaking system, i.e. the matchmaking request. Figure 9 demonstrates the outputs received by the client system from the matchmaking system, i.e. the matchmaking result. The figures show real test data from running the matchmaking system through its web service interface.

First, the request contains a reference to the product requirement description (PRD) of the switch valve product. This PRD contains all the process steps and their requirements included in the valve assembly process, but for illustration, the focus here

is on one step, 'stick subassembly screwing' (green arrow). It requires the screwing of an M6 hexagon socket head screw to the end torque between 13 and 17 Nm. The PRD is graphically illustrated in the top left corner of Figure 8, while the lower part illustrates the actual request message sent by the client through the web service interface in XML format. Figure 10 illustrates the requirement data modelled with the Protégé ontology editor in the Product Model format. It contains the description of the parts and assemblies, the processes, and their related process capability requirements, including the

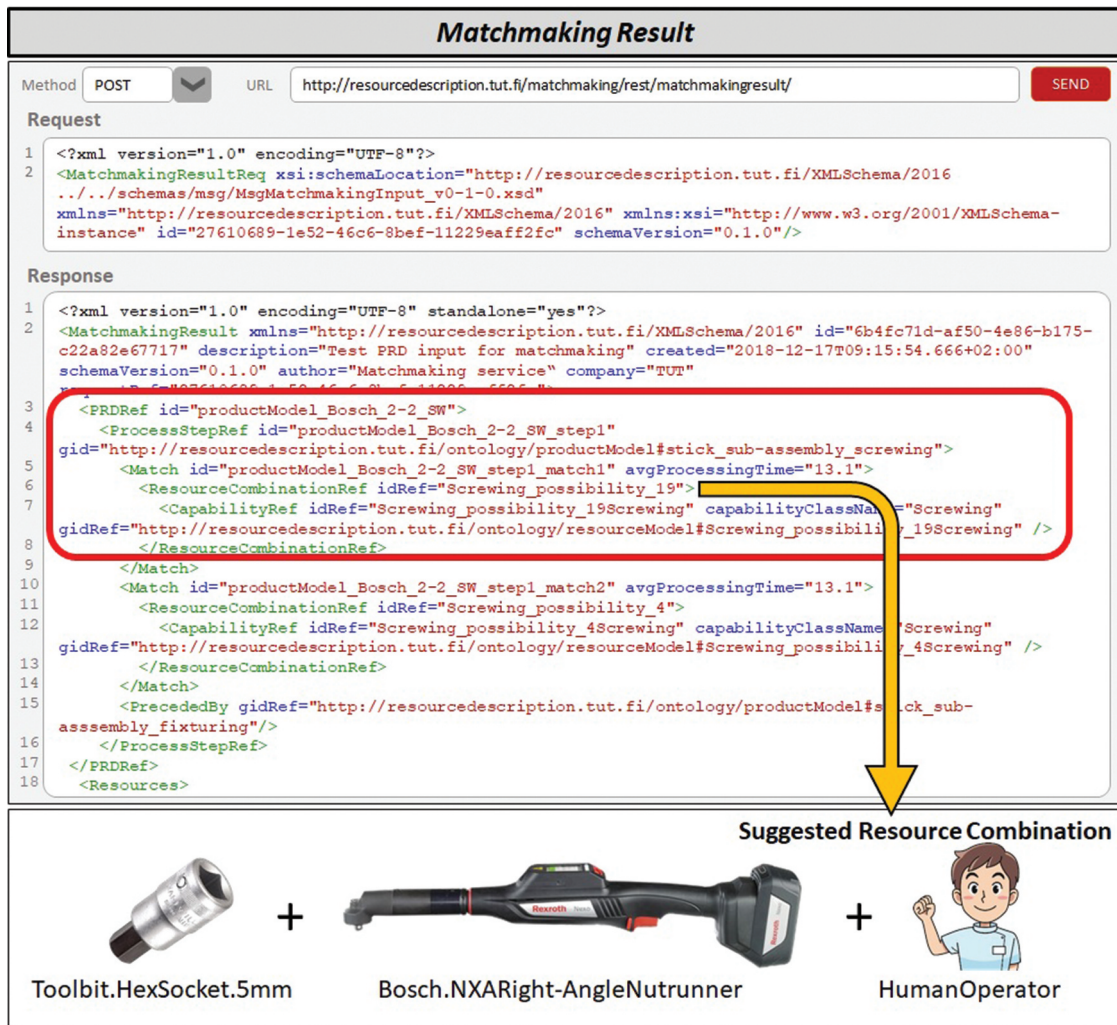


Figure 9. Example of matchmaking result.

parameters. The figure highlights the requirement data related to the specific process step 'stick sub-assembly screwing'.

Another input needed by the matchmaking system is the description of the resources included in the matchmaking search space (i.e. the resource pool). This is illustrated with the resource pool and blue arrow in Figure 8. Figure 11 represents the example capability and interface information of a few sample resources included in the sample resource pool. The interface information contains the interface standard, interface gender, and interface parameters with associated values. The interfaces must be compatible to allow for a connection. For example, IF2 of UR10 and IF1 of FingerGripper are connectable: they follow the same standard (Schunk_SWS), and their parameters are the same. Interface gender is the opposite (M)ale

and (F)emale, so these interfaces are connectable. Thus, these resources are connectable. Interface concept and matchmaking are discussed in more detail in (Siltala, Järvenpää, and Lanz 2019b; Järvenpää et al. 2019a; Siltala, Järvenpää, and Lanz 2021).

After the request has been sent through the designer's client system, matchmaking is performed for each process step. Figure 12 represents the internal reasoning process of the matchmaking system for dealing with the specific process step requiring the 'Screwing' capability. Only a subset of the complete test data is shown. First, matchmaking creates new resource combinations from the available resource pool for the required capability at the concept name level, i.e. resource combinations that could provide the 'Screwing' capability. The available resources in the resource pool are

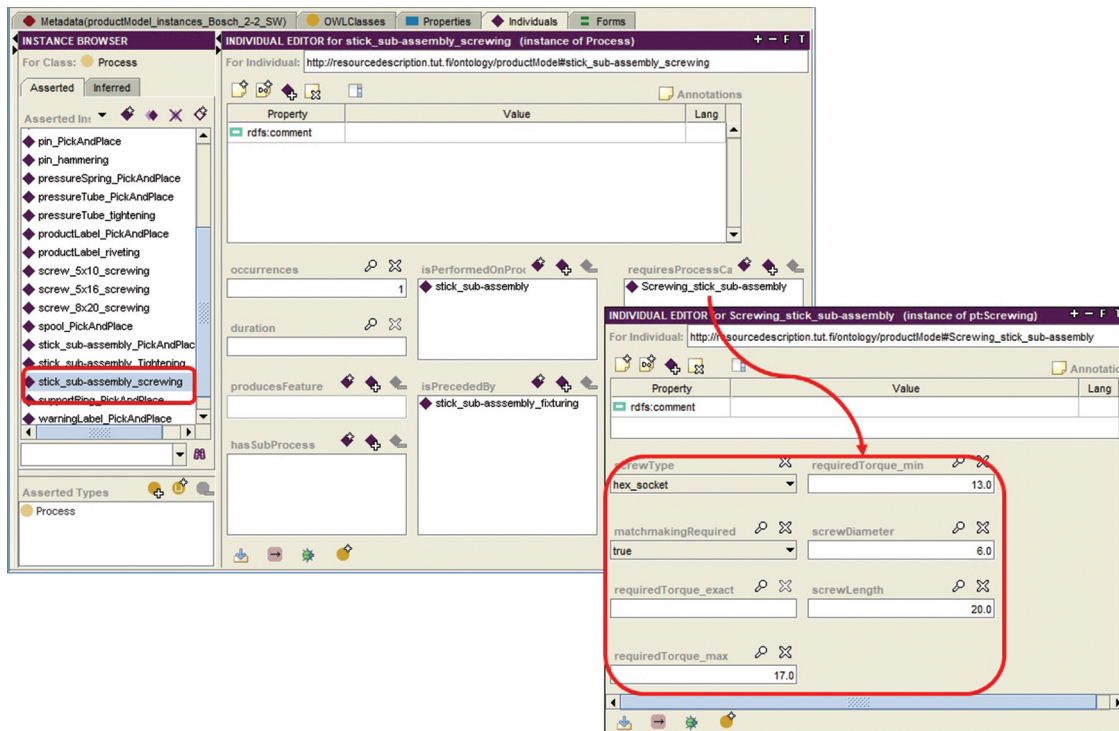


Figure 10. Example PRD modelled with Protégé ontology editor.

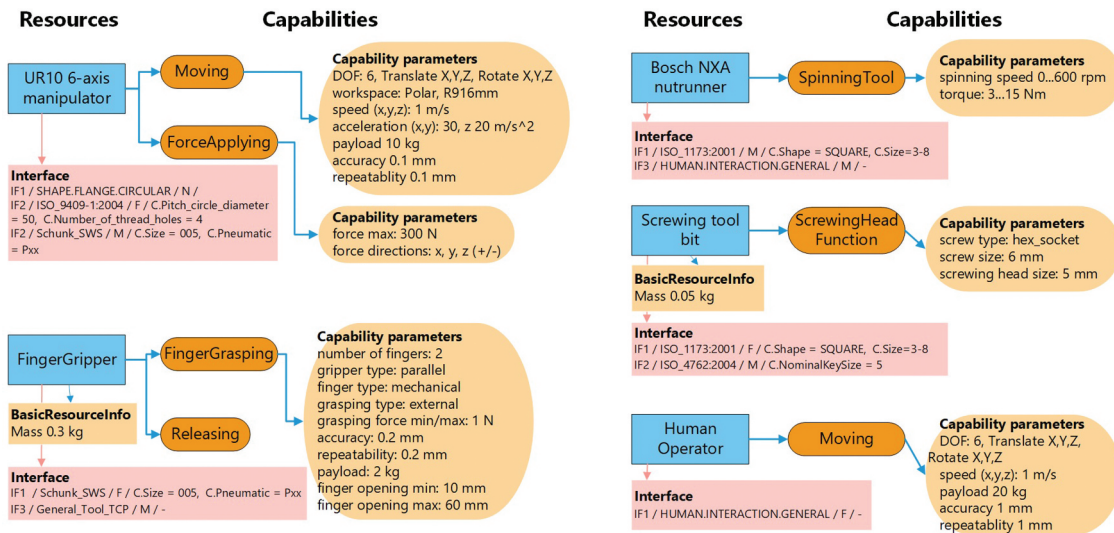


Figure 11. Sample instances in the resource pool.

illustrated in the left of the figure, while the first phase presents four different combinations (i–iv) created by the matchmaking software, matching on the capability name level. While creating these combinations, matchmaking software will simultaneously check that the interfaces of the resources are compatible, and the resources can be physically connected. In the case of the resource

combination (ii) in the second phase, the tool bit does not fit into the screw driver's tool interface, so the incompatible combination from the interface perspective is filtered out.

Figures 13 and 14 show sample resource combinations created during the matchmaking process for the 'Screwing' and 'PickAndPlace' capabilities in more detail. The figures include information about the

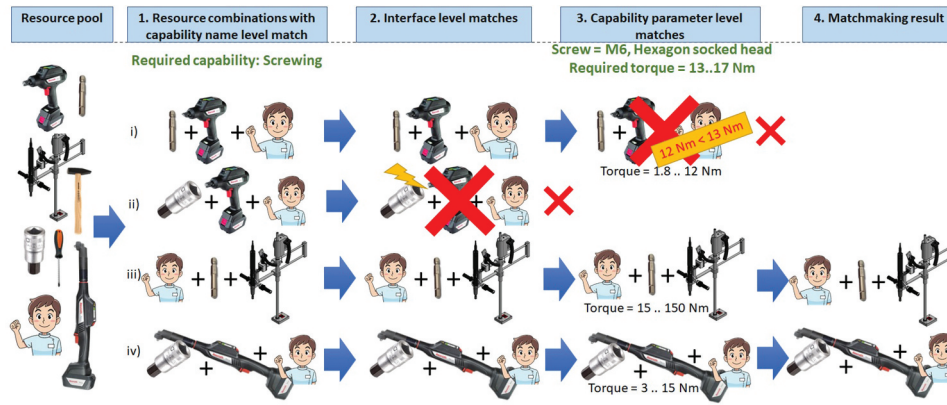


Figure 12. Internal reasoning procedure of the matchmaking system.

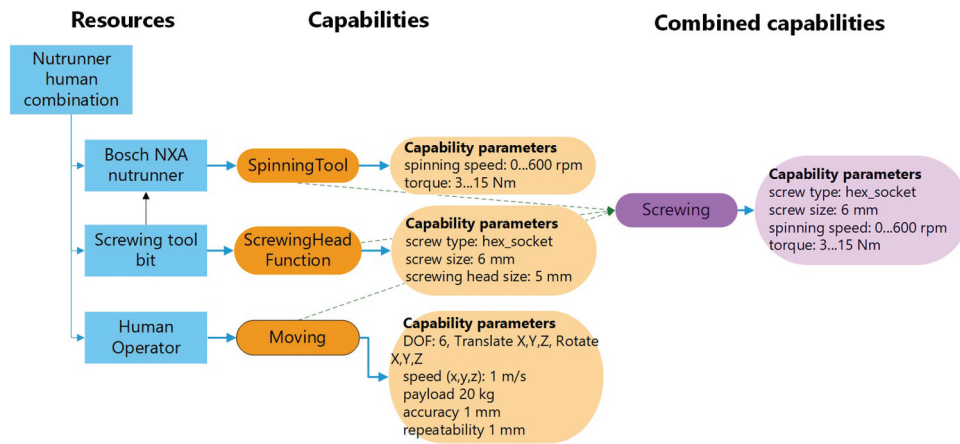


Figure 13. Created resource combination for screwing and its inferred capability parameters.

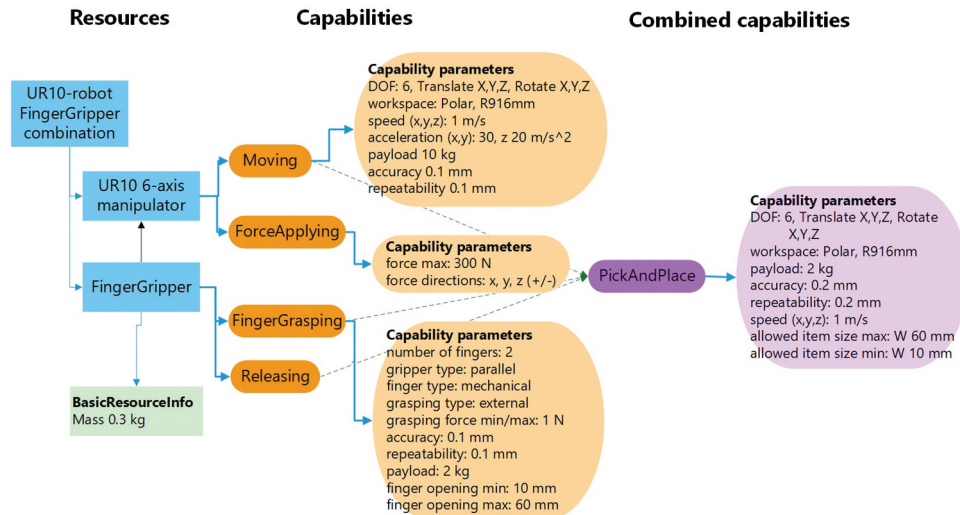


Figure 14. Created resource combination for PickAndPlace and its inferred capability parameters.

combined capability parameters inferred during the matchmaking process based on the combined capability SPIN rules. In the case of the 'PickAndPlace' Rule example 3 has been used. Figure 13 corresponds to the combination (iv) in Figure 12.

In the final phase, the matchmaking SPIN rules compare the parametric requirements of the product with the capability parameters. In the example case, Rule example 6 checks that the screw type and screw head size match the tool type and size and that the required torque is within the provided range. Unsuitable resources and resource combinations are again filtered out. In this case, the combination possibility (i) (in phase three of Figure 12) does not provide enough torque, and it is eliminated from this result. In the end, only two resource combinations remain as feasible suggestions in the matchmaking result. The alternative (iv) is the same and appears as a selected result in Figure 9.

6.2. Validation

For validating the functionality of the Matchmaking software and rules, various search spaces were used to run several matchmaking scenarios for six case products. The case products included three valve products from the process and agriculture sector, a manifold and pitch trimmer from the aviation sector and a simple demonstration product with a disc stacking process from an industrial laboratory environment. Sixty-seven (67) resources were included in the resource pool, including robots, grippers, screwdrivers, tool bits, presses, fixtures, and human operators. The software validation and testing process and their results appeared in (Järvenpää et al. 2019b).

During the validation each match suggested by the matchmaking result was manually reviewed to check if the suggested resource combinations for each process step were valid. Based on the information models and defined rules, certain results were expected, and

the validity of the matchmaking results was evaluated based on those expectations. If the matchmaking system produced expected results, these results were considered valid. Thus, the limitations of the models and rules were considered and the expectations were set according to those limitations. In other words, the authors tested that the rules and software behaved as expected but acknowledged the fact that such rules and information representations are always simplifications of reality. For example, matchmaking does not consider all constraints deriving from factory facilities and other manufacturing resources. If the product requirement description requested the 'Fixturing' process, the matchmaking system could suggest a press that has an integrated fixture, because in addition to 'Pressing' it also has 'Fixturing' capability, as requested by the product. In this case the result is valid according to the set criteria, but still not feasible in practice. Therefore, the suggestions provided by the matchmaking system must be evaluated by the designer, to make sure of their feasibility for the specific application and context.

In addition to the full Matchmaking software testing, the rules were tested right after their implementation in TopBraid Composer, a semantic web modelling tool. It allowed rapid validation of the rules, without the need to run the full matchmaking cycle through the Matchmaking Web Service. TopBraid Composer was also used to make larger matchmaking tests to check that the results provided by the Matchmaking Web Service were similar to those inferred directly by the semantic rules. Multiple PRD files were included in the testing simultaneously.

In case of clearly invalid matchmaking results (i.e. not expected), the root cause of the problem was examined. This was done by checking the software code, input data, and matchmaking rules. Two types of invalid results were recognized: 1) a found match that should not be a match and 2) a known match not

Table 1. Inputs (resource pools) and results of the matchmaking tests.

Resource pools used for matchmaking (search space)							Matchmaking test results			
Resource Pool	Device Blueprints	Individual Devices	Device Combinations	Capabilities	Layout Fixed?	Created Resource Combinations	Combined Capabilities	Found Matches	Process Steps without Matches	Processing Time (mm:ss)
1 ResPool 1	67	1	0	127	N/A	44	76	24	1	12:33
2 SysLayout 1	25	26	9	55	T	0	26	13	5	00:58
3 SysLayout 1	25	26	9	55	F	18	49	27	1	01:58
4 SysLayout 1 + ResPool 1	67	26	9	127	F	44	87	31	1	15:12

found. It was easy to detect type 1 errors. In the case of type 2 errors, some unexpected combinations may have been neglected because of being overlooked during the analysis. The root causes were fixed, and the matchmaking tests run iteratively until no invalid results appeared with the given search spaces.

6.3. Performance evaluation

The performance of the Matchmaking system during the validation tests has been described in detail in (Järvenpää et al. 2019b). In this section, a summary of those results will be provided from the perspective of the switch valve case product discussed in Section 6.1. The total number of process steps requiring matchmaking was 14. These processes required seven different capabilities: Fixturing, Pick&Place, Hammering, BlindRiveting, Screwing, Pressing and MountingO-ring.

Table 1 presents the inputs and results of four matchmaking scenarios. In all these scenarios, the input product was the same, but the resource pool varied. ResPool 1 contained 67 catalogue resources ('Device Blueprints'), and SysLayout 1 described the existing production system layout for another product, not originally for the switch valve. The layout included the physically existing resource instances ('Individual Devices'), their catalogue representation ('Device Blueprints') and combinations of these individual instances ('Device Combinations') according to the production system layout. The resource pools included various robots, grippers, fixtures, screwdrivers, drills and their associated bits, presses, O-ring mounting devices, and hammers, as well as human operators. The column 'Capabilities' indicate how many capabilities had been asserted for these resources in their resource description.

Table 1 shows the test results from the matchmaking with different search spaces: 1) all catalogue resources; 2) only the existing layout and its resource combinations (without permission to break and reconfigure the layout); 3) individual resources in the existing layout (with permission to break and reconfigure the layout); and 4) existing resources in the current layout supplemented with all catalogue resources. The last column indicates the duration of the matchmaking process with the given search space, ranging from less than a minute to over 15 minutes. The time seems to depend on the number of

new resource combinations generated during the process, which again depends on the number of resources included in the search space, their capabilities, and the number of different capabilities required by the product.

7. Discussion

Designing a manufacturing environment (workstation, cell, line) is a challenging, multi-faceted, and time-consuming task for a human. The designer uses a lot of time for searching and filtering potential resources for the system. Finding feasible resources from scattered resource catalogues (each resource provider having its own catalogue(s), more or less digital) requires a vast amount of work, and the comparison is difficult because the datasheets and properties on those are not standardized across the resource providers. Thus, the number of considered resource solutions is usually limited, somewhat random, and often dependent on the already established relations and collaboration with a few resource providers.

This article presented the capability matchmaking concept, procedure, rules, software, and underlying information models that can partly automate the search and filtering activities done during the production system design and reconfiguration planning process. The approach relies on formalized descriptions of resources and products by OWL-ontologies, which proved a suitable technology for encoding the information and knowledge traditionally described in an unstructured way. The presented MaRCO model provides a standard vocabulary for vendor-neutral representation of resource capabilities, thus supporting the matchmaking in a multi-vendor system design and reconfiguration context.

The performed case study proves that it is possible to present the requirements of the product, and capabilities and interfaces of the resources, in a way that allows automatic searching of feasible resources and resource combinations to specific product requirements. The central elements of this matchmaking process are the semantic rules implemented with the SPIN rule language. SWRL has been the more popular rule language in similar research. However, compared to SWRL, SPIN has many advantages, including its expressiveness and metamodeling features. The utilisation of templates and functions reduces the efforts needed for rule creation and

maintenance. Moreover, SPIN builds on top of established SPARQL, which has good tool support (e.g. engines and databases) and doesn't require learning a new language. However, the main reason for selecting SPIN over SWRL is that the presented application requires the ability to assert new instances, which is not supported by SWRL (Ian et al. 2004; Meditskos et al. 2013).

A particularly interesting and novel contribution in the presented approach is the ability to automatically infer combined capabilities, including their parameters, of combined resources. Thus, there is no need to create these descriptions manually. Instead, the resource combinations, and their descriptions, can be dynamically created for specific requirements based on the resource descriptions of single resources. Furthermore, the existing resource combinations can be decomposed into single resources, which provides the basis for the automatic reasoning methods to provide suggestions for the reconfiguration actions. Such suggestions may relate, for instance, to changing the tool tip of a screwdriver to allow screwing of different size screws, changing the magazine of a tube feeder to allow different size parts to be fed, or to changing the gripper of a robot to manipulate parts of different dimensions. These mechanisms are valuable in plug-and-produce scenarios, both in greenfield and brownfield contexts. Furthermore, the capability-oriented description of resources allows the separation of the functionality provided by the resource from the actual technical implementation. This means that different resources may provide similar functionality and thus contribute to the same capability. For instance, both a milling machine and drill can perform the drilling process, when combined with a proper tool.

The SPIN rules attached to the ontology classes follow similar inheritance like other properties in OWL. This means that the rules attached to a specific class also apply to its subclasses. This makes it possible to have flexibility in the definition of the product requirements. For instance, it may be defined that some feeding or joining is required without specifying the exact production method. However, it was noted during the definition of the rules that it is often difficult to define rules for the parent classes that would also be feasible for all the lower-level classes. Feeding is a good example of a relatively straightforward case, as, for all feeding

methods, the common requirements are the size of the object and feed rate. Joining is another extreme, as different joining methods, such as riveting, screwing, and gluing, each have their very specific characteristics. Thus, it is not possible to make matchmaking with capability parameters without specifying the desired joining method. However, it is possible to practice concurrent engineering and design for assembly (DFA) principles and use matchmaking to scan potential alternatives (e.g. in the current system, in-house, or easily available), before making the final selection of the preferred joining method.

There is always a trade-off between the comprehensiveness of the model and the complexity of its use. A model is a model and can never exactly mirror the complex real-world situation. The authors aimed for as simple models as possible without compromising the ability to provide useful matchmaking results. But it is evident, that the matchmaking system, in its current development stage, can only provide crude estimations of feasible resource combinations for specific needs. One reason is that the combined capability calculation and matchmaking cannot consider complex real-world properties, such as temperature influences or friction coefficients. The combined capabilities and their properties emerge as a behaviour of the machine or station as a whole in a specific context and environment. It is impossible to decompose these properties into the properties of the various resources within the station (i.e. simple capabilities). Furthermore, some capabilities depend on the physical distance between the co-operating resources. At this moment, the resource combinations are considered as a set, not as a connected graph (layout or hierarchy) and the approach can not consider the relative physical positions between the resources in the layout and their interface reservations. Thus, it is not possible to analyse e.g. the reachability of the robot arm to the components. Auxiliary and supporting resources (such as frames and structural bodies), which don't directly contribute to the processing capabilities, are also not considered. In addition, a feasible physical connection between the manipulated part and the resource is not considered. Thus, the capability matchmaking relies on coarse size definition (bounding box) when comparing e.g. the suitability of a gripper for grasping a specific object.

For the above-mentioned reasons, a human designer must validate the matchmaking results and make the final resource selection. For instance, in the presented case study, the provided matchmaking results were reviewed by the researchers to analyse their feasibility. It turned out that all the found matches were technically possible, but some suggestions were more practically reasonable than others. For instance, as discussed in the validation section, for the fixturing process, the matchmaking suggested, as one option, a press with an integrated fixture. If pressing is not required, it is not practical to select such a resource. In such a case, the designer has to consider this and select from the suggestion list a resource that is a better fit.

Despite these limitations, the authors believe that the presented matchmaking approach can provide a valuable aid for the system designer to find appropriate resource solutions from a large amount of input data. By automating the search and filtering of feasible resources and resource combinations for specific product requirements from large search spaces, and automating the identification of required reconfiguration actions on the current system layout, the matchmaking system is expected to provide at least the following benefits: 1) Letting the designers and reconfiguration planners use their time for the actual design and planning tasks, instead of cumbersome search and filtering of feasible resources and resource combinations from various catalogues; 2) Reducing human errors in resource search and filtering; 3) Increasing the number of alternative resource solutions considered, leading potentially to more efficient production system configurations; 4) Reducing the time used for system design and reconfiguration planning activity, and thus lowering the design costs. As was shown in the performance evaluation section, the time consumed by producing the matchmaking result for the case product varied from 1 minute to 15 minutes. It is clear, that even though the Matchmaking System has not yet been optimized for performance, it still outperforms humans in the resource search and filtering in terms of time.

The new paradigm of Industry 5.0 proposes a shift from technology-oriented viewpoint of Industry 4.0 to the human-centric viewpoint in which humans and machines collaborate, rather

than compete against each other (Nahavandi 2019). In (Wilson and Paul 2018) this collaboration is referred to as Collaborative Intelligence, in which Artificial Intelligence and human intelligence are used in a complementary way to enhance each other's strengths and rise above the limitations. The proposed work presents a human-centric automation approach that places human in the control of decision making, and lets the machine perform routine tasks requiring a lot of processing power. Humans can reason about situation-specific criteria, blending their pre-existing knowledge, experience, professional judgment, and subjective perception, to create context-aware insights. The capability matchmaking system assists the designer (machine assisting human) by providing high data processing capacity and data-driven insights at a key point of the system design process, letting the human designer take informed decisions with a greater set of feasible alternatives than before. The machine does what it does best, processing a vast amount of data to provide alternative options for humans, and humans do what they do best, exercise their intuition and judgment to select the best fit from a set of choices as envisioned by (Wilson and Paul 2018).

8. Conclusions

This article intended to present the capability matchmaking concept and system and their underlying information models, procedure and rules. Such matchmaking can be exploited in greenfield and brownfield system design to partly automatize the search for suitable resources and resource combinations for product requirements. The approach is based on semantic reasoning with formal ontologies representing manufacturing resources and products in OWL format. However, as pure OWL is not able to infer and assert new instances to the ontology, nor to perform complex arithmetic operations, which are required during the matchmaking, the ontologies were enriched with semantic rules. For rule implementation SPIN (SPARQL Inferencing Notation) was used. These SPIN rules calculate the capabilities of combined resources and compare the requirements of the product with the capabilities of the resources to find matches and save that information back to the ontology. The utilisation of ontologies and other

Semantic Web technologies allows automatic reasoning and inference of new information not explicitly stated before. Such information can then be used in subsequent reasoning activities.

The main scientific contributions of the work are the following. The presented MaRCO model and associated combined capability rules contribute to the existing resource and capability modelling research by providing an approach for inferring and modelling the combined capabilities of multiple cooperating resources. It is a unique approach and implementation, which other researchers have not presented. In the field of capability matchmaking, similar conceptual ideas have been presented, e.g. in (Ameri and McArthur 2014). The main difference, however, is the ability to automatically aggregate simple capabilities into combined capabilities and to infer their parameters. This automatic inference of implicit information allows the resources to be described at lower level of granularity and eliminates the need to describe the combined capabilities manually for each possible resource combination. Thus, it contributes towards reconfigurable plug-and-produce scenarios. Furthermore, the research contributes to the application of the SPIN rule language, which has not been much discussed in scientific articles. This paper gives detailed examples on the usage of SPIN in the context of capability matchmaking.

Currently, a graphical user interface (GUI) is being developed for the Matchmaking service to allow easier testing and utilisation of the service without an external design system client. Also, a connection to a 3D simulation environment is being built to allow the loading of the suggested resources directly into a simulation canvas for validation and feasibility checks. In the future, new industrial projects will be established to test the models and associated capability matchmaking in wider industrial settings covering a larger number of different resources and process capabilities. Consequently, new capability classes and their associated properties, and rules for combined capability calculation and matchmaking, will be implemented to increase the capability catalogue when needed. Currently, human resources are modelled in the same way as machine resources. In the future, human modelling will be further elaborated upon to include characteristics of humans and

to make the description approach more acceptable. The authors see that one of the biggest issues hindering large-scale industrial exploitation of the presented approach is the lack of formalized resource descriptions and global resource catalogues. The possibilities of Artificial Intelligence technologies for automatic generation of resource descriptions out of resource datasheets and other unformalized data to the format required by matchmaking should be investigated.

Acknowledgments

This research has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 680759 with project title ReCaM (Rapid Reconfiguration of Flexible Production Systems through Capability-based Adaptation, Autoconfiguration and Integrated Tools for Production Planning) and under grant agreement no. 952003 with project title AI REGIO (Regions and DIHs alliance for AI-driven digital transformation of European Manufacturing SMEs).

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the European Commission [680759,952003].

ORCID

Eeva Järvenpää  <http://orcid.org/0000-0001-6513-135X>

Niko Siltala  <http://orcid.org/0000-0001-6456-1251>

Hasse Nylund  <http://orcid.org/0000-0002-1486-2301>

Minna Lanz  <http://orcid.org/0000-0003-2182-4669>

References

- Aarnio, P., V. Vyatkin, and D. Hastbacka. 2016. "Context Modeling with Situation Rules for Industrial Maintenance." In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 9. doi:10.1109/ETFA.2016.7733539.

- Ameri, F., C. Urbanovsky, and C. McArthur. 2012. "A Systematic Approach to Developing Ontologies for Manufacturing Service Modeling." In *Proceedings of the Workshop on Ontology and Semantic Web for Manufacturing* July 24, 2012 Graz, Austria, 1–14.
- Ameri, F., and C. McArthur. 2014. "Semantic Rule Modelling for Intelligent Supplier Discovery." *International Journal of Computer Integrated Manufacturing* 27 (6): 570–590. doi:10.1080/0951192x.2013.834467.
- Apache Software Foundation. 2017. "Apache Jena - A Free and Open Source Java Framework for Building Semantic Web and Linked Data Applications." <https://jena.apache.org/>
- Backhaus, J., and G. Reinhart. 2017. "Digital Description of Products, Processes and Resources for Task-Oriented Programming of Assembly Systems." *Journal of Intelligent Manufacturing* 28 (8): 1787–1800. doi:10.1007/s10845-015-1063-3.
- Bassiliades, N. 2018. "SWRL2SPIN: A Tool for Transforming SWRL Rule Bases in OWL Ontologies to Object-Oriented SPIN Rules." *ArXiv ID 1801.09061*, 13.
- Bengel, M. 2009. "Model-Based Configuration – A Workpiece-Centred Approach." In *ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots*, 689–695. <http://ieeexplore.ieee.org/document/5173901/>
- Bortolini, M., F. Gabriele Galizia, and C. Mora. 2018. "Reconfigurable Manufacturing Systems: Literature Review and Research Trend." *Journal of Manufacturing Systems* 49: 93–106. September. doi:10.1016/j.jmsy.2018.09.005.
- Cao, Q., F. Giustozzi, C. Zanni-Merk, F. De Beuvron Bertrand, and C. Reich. 2019. "Smart Condition Monitoring for Industry 4.0 Manufacturing Processes: An Ontology-Based Approach." *Cybernetics and Systems* 50 (2): 82–96. doi:10.1080/01969722.2019.1565118.
- CO2PE! 2010. "CO2PE! - Taxonomy." 2010. <http://www.mech.kuleuven.be/co2pe!/taxonomy.php>
- Cutting-Decelle, A. F., R. I.M. Young, J. J. Michel, R. Grangel, J. Le Cardinal, and J. P. Bourey. 2007. "ISO 15531 MANDATE: A Product-Process-Resource Based Approach for Managing Modularity in Production Management." *Concurrent Engineering Research and Applications* 15 (2): 217–235. <https://doi.org/10.1177/1063293X07079329>.
- Doulaverakis, C., V. Koutkias, G. Antoniou, and I. Kompatsiaris. 2017. "Applying SPARQL-Based Inference and Ontologies for Modelling and Execution of Clinical Practice Guidelines: A Case Study on Hypertension Management." In *Knowledge Representation for Health Care*, edited by D. Riaño, R. Lenz, and M. Reichert. Cham: Springer. doi:10.1007/978-3-319-55014-5_6.
- Efthymiou, K., K. Sipsas, D. Mourtzis, and G. Chrysosouris. 2015. "On Knowledge Reuse for Manufacturing Systems Design and Planning: A Semantic Technology Approach." *CIRP Journal of Manufacturing Science and Technology* 8: 1–11. doi:10.1016/j.cirpj.2014.10.006.
- Horrocks, I., P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof, and M. Dean. 2004. "SWRL: A Semantic Web Rule Language Combining OWL and RuleML." *W3C Member Submission*. <https://www.w3.org/Submission/SWRL/>
- Jardim-Goncalves, R., A. Grilo, and K. Popplewell. 2016. "Novel Strategies for Global Manufacturing Systems Interoperability." *Journal of Intelligent Manufacturing* 27 (1): 1–9. doi:10.1007/s10845-014-0948-x.
- Järvenpää, E., N. Siltala, O. Hylli, and M. Lanz. 2017. "Capability Matchmaking Procedure to Support Rapid Configuration and Re-Configuration of Production Systems." *Procedia Manufacturing* 11. doi:10.1016/j.promfg.2017.07.216.
- Järvenpää, E., O. Hylli, N. Siltala, and M. Lanz. 2018a. "Utilizing SPIN Rules to Infer the Parameters for Combined Capabilities of Aggregated Manufacturing Resources." *IFAC-Papers Online* 51 (11): 84–89. doi:10.1016/j.ifacol.2018.08.239.
- Järvenpää, E., N. Siltala, O. Hylli, and M. Lanz. 2018b. "Product Model Ontology and Its Use in Capability-Based Matchmaking." *Procedia CIRP* 72: 1094–1099. doi:10.1016/j.procir.2018.03.211.
- Järvenpää, E., N. Siltala, and O. Hylli. 2019. "Product, Manufacturing Resource and Capability Ontologies." <http://urn.fi/urn:nbn:fi:csc-kata20190225153111925507>
- Järvenpää, E., N. Siltala, O. Hylli, and M. Lanz. 2019a. "The Development of an Ontology for Describing the Capabilities of Manufacturing Resources." *Journal of Intelligent Manufacturing* 30 (2): 959–978. doi:10.1007/s10845-018-1427-6.
- Järvenpää, E., N. Siltala, O. Hylli, and M. Lanz. 2019b. "Implementation of Capability Matchmaking Software Facilitating Faster Production System Design and Reconfiguration Planning." *Journal of Manufacturing Systems* 53: 261–270. October. doi:10.1016/j.jmsy.2019.10.003.
- Järvenpää, E., N. Siltala, O. Hylli, and M. Lanz. 2021. "Capability Matchmaking Software for Rapid Production System Design and Reconfiguration Planning." *Procedia CIRP* 97: 435–440. doi:10.1016/j.procir.2020.05.264.
- Knublauch, H. 2013. "SPIN - SPARQL Syntax." *W3C Member Submission*. <https://spinrdf.org/sp.html>
- Knublauch, H. 2016. "The TopBraid SPIN API." <http://topbraid.org/spin/api/>
- Köcher, Aljosh, Constantin Hildebrandt, , and Alexander Fay. 2020. "A Formal Capability and Skill Model for Use in Plug and Produce Scenarios." In 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). <https://doi.org/10.1109/ETFA46521.2020.9211874>.
- Leitão, P., A. W. Colombo, and S. Karnouskos. 2016. "Industrial Automation Based on Cyber-Physical Systems Technologies: Prototype Implementations and Challenges." *Computers in Industry* 81: 11–25. doi:10.1016/j.compind.2015.08.004.
- Li, Z., W. M. W. Xiaowu Zhou, G. Huang, Z. Tian, and S. Huang. 2018. "An Ontology-Based Product Design Framework for Manufacturability Verification and Knowledge Reuse." *The International Journal of Advanced Manufacturing Technology* 99: 2121–2135. doi:10.1007/s00170-018-2099-2.
- Lohse, N., T. Maraldo, and J. Barata. 2008. "EUPASS Std-0007: Assembling Process Ontology Specification." *EUPASS Project Specification*: 38.

- Lu, Y., H. Wang, and X. Xu. 2016. "ManuService Ontology: A Product Data Model for Service-Oriented Business Interactions in A Cloud Manufacturing Environment." *Journal of Intelligent Manufacturing* 1–18. doi:10.1007/s10845-016-1250-x.
- Lu, Y., and X. Xu. 2017. "A Semantic Web-Based Framework for Service Composition in A Cloud Manufacturing Environment." *Journal of Manufacturing Systems* 42: 69–81. doi:10.1016/j.jmsy.2016.11.004.
- Lu, Y., and X. Xu. 2018. "Resource Virtualization: A Core Technology for Developing Cyber-Physical Production Systems." *Journal of Manufacturing Systems* 47: 128–140. April. doi:10.1016/j.jmsy.2018.05.003.
- Luo, Y., L. Zhang, F. Tao, L. Ren, Y. Liu, and Z. Zhang. 2013. "A Modeling and Description Method of Multidimensional Information for Manufacturing Capability in Cloud Manufacturing System." *International Journal of Advanced Manufacturing Technology* 69 (5–8): 961–975. doi:10.1007/s00170-013-5076-9.
- Maleki, E., F. Belkadi, N. Boli, B. Jan Van Der Zwaag, K. Alexopoulos, S. Koukas, M. Marin-perianu, and A. Bernard. 2018. "Ontology-Based Framework Enabling Smart Product-Service Systems: Application of Sensing Systems for Machine Health Monitoring." *IEEE Internet of Things Journal* 5 (6): 4496–4505.
- Meditoskos, G., S. Dasiopoulou, V. Efstathiou, and I. Kompatsiaris. 2013. "SP-ACT: A Hybrid Framework for Complex Activity Recognition Combining OWL and SPARQL Rules." In *IEEE Workshop on Context Modeling and Reasoning 2013*, 25–30. doi:10.1109/PerComW.2013.6529451.
- Nahavandi, S. 2019. "Industry 5.0 - A Human-Centric Solution." *Sustainability* 11 (16): 4371. doi:10.3390/su11164371.
- The OWL Working Group. 2004. "OWL Web Ontology Language Overview." *W3C Recommendation*. <https://www.w3.org/TR/owl-features/>
- Pfrommer, J., D. Stogl, K. Aleksandrov, V. Schubert, and B. Hein. 2014. "Modelling and Orchestration of Service-Based Manufacturing Systems via Skills." In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 1–4. doi:10.1109/ETFA.2014.7005285.
- Pintzos, G., M. Matsas, and G. Chryssolouris. 2012. "Defining Manufacturing Performance Indicators Using Semantic Ontology Representation." *Procedia CIRP* 3: 8–13. doi:10.1016/j.procir.2012.07.003.
- Rampersad, H.K. 1994. *Integrated and Simultaneous Design for Robotic Assembly*. Chichester: John Wiley & Sons Ltd 212 0471954667.
- Siltala, N., E. Järvenpää, and M. Lanz. 2018. "Value Proposition of a Resource Description Concept in a Production Automation Domain." *Procedia CIRP* 72: 1106–1111. doi:10.1016/j.procir.2018.03.154.
- Siltala, N., E. Järvenpää, and M. Lanz. 2019a. "A Method to Evaluate Interface Compatibility during Production System Design and Reconfiguration." *Procedia CIRP* 81: 282–287. doi:10.1016/j.procir.2019.03.049.
- Siltala, N., E. Järvenpää, and M. Lanz. 2019b. "Creating Resource Combinations Based on Formally Described Hardware Interfaces." *IFIP Advances in Information and Communication Technology* 530: 29–39. doi:10.1007/978-3-030-05931-6_3.
- Siltala, N., E. Järvenpää, and M. Lanz. 2021. "Resource Interface Matchmaking as a Part of Automatic Capability Matchmaking." *IFIP Advances in Information and Communication Technology* 51–62. doi:10.1007/978-3-030-72632-4_4.
- Sirin, E., B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz. 2007. "Pellet: A Practical OWL-DL Reasoner." *Journal of Web Semantics* 5 (2): 51–53. doi:10.1016/j.websem.2007.03.004.
- SPIN Working Group. 2017. "SPIN - SPARQL Inferencing Notation." <https://spinrdf.org/>
- Sun, W., Q.-Y. Ma, and T.-Y. Gao. 2009. "An Ontology-Based Manufacturing Design System." *Information Technology Journal* 8 (5): 643–656. doi:10.3923/itj.2009.643.656.
- Sure, Y., S. Staab, and R. Studer. 2009. "Ontology Engineering Methodology." In *Handbook on Ontologies*. 2nd ed., edited by S. Staab and R. Studer, Heidelberg: Springer Berlin. 135–152.
- Thoben, K.-D., S. Wiesner, and T. Wuest. 2017. "'Industrie 4.0' and Smart Manufacturing – A Review of Research Issues and Application Examples." *International Journal of Automation Technology* 11 (1): 4–16. doi:10.20965/ijat.2017.p0004.
- Tolio T, Ceglarek D, ElMaraghy H, Fischer A, Hu S, Laperrière L, Newman S and Váncza J. 2010. SPECIES—Co-evolution of products, processes and production systems. *CIRP Annals* 59 (2): 672–693. doi:10.1016/j.cirp.2010.05.008
- W3C SPARQL Working Group. 2013. "SPARQL 1.1 Query Language." *W3C Recommendation*. <https://www.w3.org/TR/sparql11-query/>
- Wilson, H. J., and R. D. Paul. 2018. "Collaborative Intelligence: Humans and AI are Joining Forces." *Harvard Business Review*, July–August.
- Yahya, M., J. G. Breslin, and M. Intizar Ali. 2021. "Semantic Web and Knowledge Graphs for Industry 4.0." *Applied Sciences* 11 (11): 5110. doi:10.3390/app11115110.
- Yuan, M., K. Deng, and W. A. Chaovallitwongse. 2017. "Manufacturing Resource Modeling for Cloud Manufacturing." *International Journal of Intelligent Systems* 32 (4): 414–436. doi:10.1002/int.21867.