Utilizing SPIN Rules to Infer the Parameters for Combined Capabilities of Aggregated Manufacturing Resources

Eeva Järvenpää*, Otto Hylli, Niko Siltala*, Minna Lanz***

**Tampere University of Technology, Laboratory of Mechanical Engineering and Industrial Systems, Tampere, Finland (Tel: +358-40-8490869; e-mail:eeva.jarvenpaa@tut.fi, niko.siltala@tut.fi, minna.lanz@tut.fi)*
***Tampere University of Technology, Laboratory of Pervasive Computing, Tampere, Finland (e-mail: otto.hylli@tut.fi)*

Abstract: Fast reaction to changing requirements is a pre-requisite for modern manufacturing companies and their facilities. One enabler for such responsiveness is tools that can support the reconfiguration and system design process by semi-automatic matchmaking between product requirements and resource capabilities. This paper presents an approach, which uses SPARQL Inferencing Notation (SPIN) to infer and assert the combined capability information of two or more combined resources presented in OWL-format. We show example rules for calculating the combined capability parameters both in textual and SPIN formats, and visualize a case example as a proof of concept. This approach can be used to infer the capability parameters of existing resource combinations, or to look for new resource combinations for specific capability requirements.

Keywords: Capability model, Resource model, Combined Capabilities, SPARQL, SPIN, Semantic model, Inference

1. INTRODUCTION

Responsiveness is an important strategic goal for manufacturing companies operating in a highly dynamic environment characterized by constant change. Such responsiveness and adaptivity relates to the need to reconfigure and adjust the production and corresponding production system as efficiently as possible to the required changes in processing functions, production capacity, and the dispatching of the orders (Wiendahl 2007). The realization of these requirements for fast response calls for new methods and tools that can reduce the time and effort put into planning and implementing the alterations in a factory.

Within the past decade, there have been multiple different projects trying to provide computerized support for the reconfiguration planning process. The currently running project ReCaM[1] aims to develop a set of integrated tools for rapid and autonomous reconfiguration of production systems. The approach relies on a formal unified functional description of resources, providing a foundation for rapid creation of new system configurations through capability-based matchmaking of product requirements and resource offerings.

In ReCaM, we have developed a Manufacturing Resource Capability Ontology (MaRCO), which is an OWL-based information model for describing the capabilities of manufacturing resources. The MaRCO model aims to support rapid semi-automatic system design, both in greenfield and brownfield design scenarios. Integral part of the capability-based matchmaking with MaRCO is the generation of resource combinations (i.e. physical configurations) that can together satisfy the product's processing requirements. The production system capabilities originate from the tool and equipment level. Especially in case of modular and reconfigurable "plug-and-produce" type production systems, the resources can be combined to form various different configurations. Hence, the capability model and associated software tools need to support the automatic inference of the combined capability information of these combined resources. Pure OWL (W3C 2004) does not provide solutions for making such inference and assertions of new instances and their property values (Meditskos et al. 2013). Therefore, the OWL-based ontology needs to be enriched with semantic rules and supported with external software.

Literature reviews on existing capability and resource models have been presented in our earlier works, e.g. in (Järvenpää et al. 2016). The other existing models don't consider the automatic inference of combined capabilities at parameter level, and do not thus need such rules. Similar idea to our matchmaking was presented by Ameri and McArthur (2014), who utilized SWRL (Semantic Web Rule Language) for intelligent supplier discovery based on the services they provide. 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 does not have functionality to assert new instances (Horrocks et al. 2004; Meditskos et al. 2013), which is needed in our case. Meditskos et al. (2013) used SPIN (SPARQL Inferencing Notation) to perform temporal reasoning with context information and to assert

[1] This research has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 680759 (www.recam-project.eu).

new named individuals. Their application related to human activity recognition. Aarnio et al. (2016) exploited SPIN for situation rules in context modelling with the goal to support industrial maintenance. Other applications of SPIN in comparable context do not exist, or at least are not published.

In this paper, we will illustrate the usage of SPIN to automatically infer and assert the parameters of combined capabilities based on the parameters of the lower level capabilities represented in OWL-format. The paper is organized as follows. Section 2 will give more details of the MaRCO model, presented earlier in (Järvenpää et al. 2016; 2017) and introduce the SPIN rule language. Section 3 discusses the combined capability rules and their implementation with SPIN. In section 4, a case study of calculating the combined capabilities of a resource combination is presented. Section 5 concludes the paper.

2. INTRODUCTION TO RELATED MODELS AND TECHNOLOGIES

2.1 Manufacturing Resource Capability Ontology (MaRCO)

Capability Model is a data model for describing capabilities of resources. The capability concept name indicates the natural name of the capability, such as “Moving”, “FingerGrasping”, “Drilling” and “Screwing”. Capability parameters describe the characteristics of a capability, e.g. the “Moving” capability is characterized by *“speed”* and *“acceleration”* parameters, among others. The Capability Model divides the capabilities into simple and combined capabilities. Combined capabilities are upper level capabilities, which can be divided by functional decomposition into simple, lower level capabilities (*part_of* hierarchy). Combined capabilities are combinations of two or more (simple or combined) capabilities. For instance, in order to transport an item the system needs to be able to move within some workspace and to grasp the item or to hold it by gravity.

The capabilities, modelled as classes in the ontology, form the Capability Catalogue, which consists of the pool of capabilities that may exist in a production system. Capability parameters are implemented as datatype and object properties, depending on the nature of the parameter, and are associated to the capabilities by property restrictions. The capabilities can be assigned to resources through the Resource Model, which imports the Capability Model. The resource specific capabilities are saved as instances of the specific capability classes and filled with the resource specific parameter values. Based on the defined relations between the simple and combined capabilities, the resource combinations contributing to a certain combined capability can be identified. Figure 1 introduces the most relevant classes of the Resource Model, including only devices (i.e. machines and tools) and excluding e.g. human resources and factory areas (i.e. stations, cells and lines). The black arrows illustrate the class hierarchy, while the blue arrows illustrate the relations between the concepts in the OWL-model.

The class **Device** is a parent class for different device-related classes. It has two subclasses **DeviceBlueprint** and **IndividualDevice**. **DeviceBlueprint** class contains the catalogue information of devices. The instances of **DeviceBlueprint** relate to specific **Capability** instances through *hasCapability* object property. The instances of **IndividualDevice** class represent the actual individual devices existing on the factory floor. The individual devices have reference to the **DeviceBlueprint** through *hasDeviceBlueprint* object property. This class stores updated capability information through *hasCapabilityUpdated* object property. **DeviceCombination** class includes instances, which represent combinations of multiple devices or device combinations. These instances refer to the instances of **IndividualDevice** or **DeviceCombination** through *hasIndividualDevicesOrDeviceCombinations* object property. Furthermore, for the **DeviceCombination** instance, combined capability information can be saved through *hasCalculatedCapability* object property.

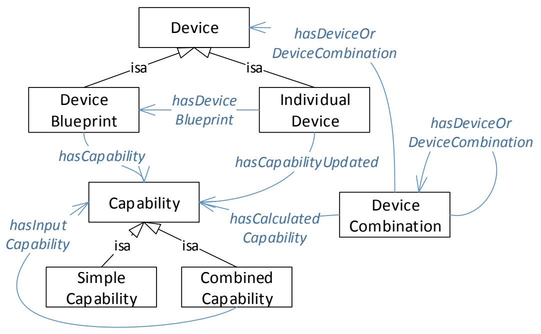


Fig. 1. Resource Model (reduced version).

When two or more resources are combined together as a functional unit, combined capabilities emerge from the simple capabilities assigned to the individual resources. The parameters for the combined capabilities need to be reasoned out based on the capabilities and properties of the resources involved in the combination. These 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. For this, combined capability rules implemented with semantic rule language are needed.

2.2 SPIN – SPARQL Inferencing Notation

SPIN (SPARQL Inferencing Notation) is a W3C Member Submission that has become the de-facto industry standard to represent SPARQL rules and constraints on Semantic Web models (SPIN working group, 2017). SPARQL is an abbreviation of SPARQL Protocol and RDF Query Language. It is a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework (RDF) format and OWL-ontologies. It was made a standard by the RDF Data Access Working Group of the W3C consortium and it is recognized as one of the key technologies of the semantic web. (W3C 2008.)

SPIN can be used to link class definitions with SPARQL queries to capture constraints and rules that formalize the expected behavior of those classes. A suitable reasoner tool such as SPIN API can then infer the extra information created by the rules and use it for example in SPARQL query execution (Knublauch 2016). This mechanism enables to organize SPARQL queries in an object-oriented way. It makes the rules accessible and easy to maintain, extend and share. 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.)

SPIN can be used to calculate the value of a property based on other properties - for example, area of a rectangle as a product of its length and width. SPIN 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. The main advantage of SPIN is that it provides meta-modeling capabilities that allow users to define their own reusable SPARQL query templates and functions. Templates are parametrized SPARQL queries that can be customized by instantiating them with the argument values of a new context. SPIN function is a special kind of template query that returns only one result value, and can be used as a part of another SPARQL query. SPIN also includes a ready to use library of common functions. (SPIN working group 2017.)

3. COMBINED CAPABILITY RULES

There are different ways how the combined capability parameters are formed: (1) Directly inheriting a parameter value from one of the involved lower level capabilities as such; (2) Calculating the parameter value by arithmetic operations from two or more involved lower level capabilities; (3) Defining the parameter value by comparing the values of the involved lower level capabilities and selecting max or min value, depending on the specific capability parameter; (4) Calculating a completely new capability parameter and value by applying arithmetic operations on two or more involved lower level capabilities. The term "lower level capability" is used to refer to capabilities that are on lower level in the *part-of* hierarchy. The term "simple capability" can not be used, as the capabilities involved in the combinations may be either simple or combined capabilities.

We first defined the combined capability rules in informal textual format, based on domain expert knowledge. After that, we implemented them with SPIN. The following sections will show examples of some of the most representative cases. We intend to keep the examples simple, so that the reader can understand the approach without detailed knowledge on the MaRCO model itself.

3.1 *Example rules in textual form*

"Transporting" capability consists of "Moving" and "Grasping" capabilities. Table 1 shows example rules for "Transporting" capability in case when the grasping is achieved through "FingerGrasping". "Picking" capability consists of "Grasping" and "Moving" capabilities. "Placing" requires also "Releasing" capability. For these capabilities, the same rules for combined capabilities apply as with "Transporting" for the *"payload"* and *"itemSize_min/max"* properties. Other rules depend on which method is used for grasping, i.e. is it "FingerGrasping" or "VacuumGrasping". Table 2 shows these rules for the case of "FingerGrasping".

Table 1. Example rules for Transporting capability.

Property	Rule
payload	*"payload"* property of "Moving" capability minus the *"mass"* of the gripper, OR *"payload"* property of "FingerGrasping" capability. The smaller value is dominating.
itemSize_max	*"fingerOpening_max"* property of the "FingerGrasping" capability. Saved to the "width" property of *itemSize_max*. Similar rule for *itemSize_min*.
dof	*"dof"* property of the "Moving" capability.
speed_x_max	*"speed_x_max"* property of the "Moving" capability. Same applies to other movement directions.
accuracy	*"accuracy"* property of the "Moving" capability.
repeatability	*"repeatability"* property of the "Moving" capability.
workspaceTypeAndDimensions	*"hasWorkspaceTypeAndDimensions"* property of "Moving" capability.

Table 2. Rules for Picking and Placing capabilities.

Property	Rule
accuracy	*"accuracy"* property of the "Moving" capability + *"accuracy"* property of the "FingerGrasping" capability (Represents the worst case scenario.)
repeatability	*"repeatability"* property of the "Moving" capability + *"repeatability"* property of the "FingerGrasping" capability (Represents the worst case scenario.)

3.2 *Implementation of the rules with SPIN*

For each parameter of a combined capability, there is a SPIN rule that is attached to that specific capability class in the ontology. There are a lot of similarities between the rules. In these cases the meta-modeling features of SPIN, i.e. functions and templates, are exploited. Every combined capability parameter rule has to get at least one of the lower level capabilities that produced the combined capability. Then the rule has to retrieve one of the parameters of that lower level capability. For this purpose, a SPIN function that gets the specified lower level capability instance of the given combined capability instance, was created (Example 1 in **Fig. 2**). The function can be used in another SPARQL query by giving some values to its arguments, as shown in Example 3. There is also another function that gets the parameter directly, but it cannot always be used since SPIN functions can only return one value and some parameters can have multiple values. SPARQL rules are defined in terms of a CONSTRUCT and a WHERE clause. The former defines the set of triple patterns that should be added to the underlying resource model upon the successful pattern matching of the triple patterns in the WHERE clause.

A very common kind of rule relates to inheriting a simple capability parameter. For example, the "*accuracy"* of "Transporting" is the same as the "*accuracy"* of "Moving", and same goes with the "*dof"* (degrees of freedom) of the "Transporting". In both cases the logic of the rule is the same, just the name of the capability and the parameter are different. In this kind of situation, we can use a SPIN template that is defined separately and given a name – inheritCapabilityParameter in this case (Example 2). A template has a body consisting of a SPARQL query and arguments whose values will be inserted into the query. The actual rule is then an instance of this template where the arguments are given concrete values. For example, the inheritCapabilityParameter template has arguments named

Example 1: SPIN function for getting the capability instance of interest - getPartCapability

```
    SELECT ?result
    WHERE {
(1)   ?deviceCombination
      rm:hasCalculatedCapability ?arg1 .
(2)   ?deviceCombination
      rm:hasIndividualDeviceOrDeviceCombination)*
      ?part .
(3)   ?part rm:hasDeviceBlueprint ?blueprint .
(4)   ?blueprint rm:hasCapability ?result .
(5)   ?result a ?arg2 .
    }
```

Comments related to the function:

Purpose: Get an instance of the given lower level capability (arg2) for the given combined capability instance (arg1).

(1) Get the device combination which has the given combined capability instance (?arg1).
(2) Get a device from which the device combination or one of its device combinations consists of.
(3) Get a blueprint for the device.
(4) Get a capability instance of the blueprint.

Example 2: SPIN rule template for inheriting the capability parameter - inheritCapabilityParameter

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

Comments related to the template:

Purpose: Assign the value of the given parameter of the given lower level capability to the combined capability instance in the variable ?this.

(1) The construct part defines how we use the query results to modify our ontology. Give the parameter value to the combined capability instance.
(2) Get an instance of the given lower level capability for the combined capability instance (?this).
(3) Check if the function returned a result. If not, this does nothing.
(4) Assign the result to a different variable. Using the result variable directly would cause problems since if it is unbound after the function call it can bind to other capability instances.
(5) Get the value of the parameter for the lower level capability instance.

Example 3: SPIN rule for calculating the *accuracy* of Transporting capability

```
CONSTRUCT {
    ?this cm:accuracy ?value .
}
WHERE {
    BIND (:getPartCapability(?this, cm:Moving ) AS ?result) .
    FILTER bound(?result) .
    BIND (?result AS ?resourceCapability) .
    ?resourceCapability cm:accuracy ?value .
```

Example 4: SPIN rule for calculating the *payload* of Transporting, Picking and Placing capabilities

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

Comments related to the rule:

(1) Give the combined capability instance (?this) the payload determined below by the rule.
(2) Get the instance of FingerGrasping for the combined capability instance.
(3) Check if the function returns a result.
(4) Assign the result to a different variable. Using the result variable directly would cause problems since if it is unbound after the function call it can bind to other capability instances.
(5) Get the device blueprint whose capability we just got.
(6) Get the basic resource information of the blueprint.
(7) Get the mass of the blueprint from the basic resource information.
(8) Get the FingerGrasping payload.
(9) Get the lower level capability instance of Moving for the combined capability instance.
(10) Get the payload for moving.
(11) Subtract FingerGrasping blueprint mass from Moving payload and save to variable alternative.
(12) The lesser value of the ?alternative or ?gripperPayload is the combined capability payload.

Fig. 2. SPIN examples: functions, templates and rules.

"capability" and "parameter" and when it is used to create the combined capability parameter rule for defining the "*accuracy*" of "Transporting", these arguments are given the corresponding values "Moving" and "accuracy". From this template instance, the SPIN rule engine executing the rules will then create the rule shown in Example 3.

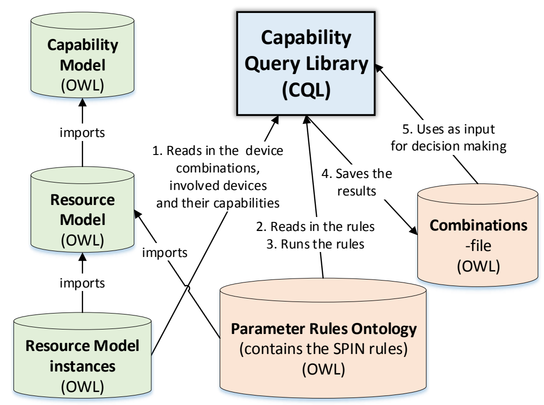
Also other similar templates have been defined, e.g. for comparing and selecting the greater or lesser value of two lower level capability parameter values, and for summing up the parameter values. The latter can be used e.g. to calculate the "*repeatability*" or "*accuracy*" of the "Picking" and "Placing" capabilities by summing up the "*repeatability*" and "*accuracy*" values of both "FingerGrasping" and "Moving" capabilities.

Some rules are the same for different capabilities. For example, the payload rule of "Transporting", "Picking" and "Placing" is the same, and therefore the same SPIN rule (Example 4) can be reused in each case.

3.3 Running the rules

For working with the MaRCO model and the rules we have implemented a software tool called Capability Query Library (CQL). It is a Java based application that offers a command line interface and a Java API for other applications to work with the MaRCO model. 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. Jena or Pellet themselves do not support SPIN, so another open source library, that builds on top of Jena, called SPIN API (Knublauch 2016) is used to execute the SPIN rules.

The defined SPIN rules, including the templates and functions the rules use, have been added to a separate Parameter Rules ontology used by the CQL. This ontology imports the Resource Model ontology. CQL reads in the Resource Model instances ontology and defines the combined capabilities of the device combinations on concept name level. This combined capability information is saved to a temporary ontology inside CQL, which imports the original Resource Model instances ontology and the Parameter Rules ontology. The combined capability parameters are then inferred by SPIN API for that ontology. This information is then saved to a separate ontology file (Combinations), which can be used later for example during the capability matchmaking process. **Fig. 3** illustrates the explained procedure.

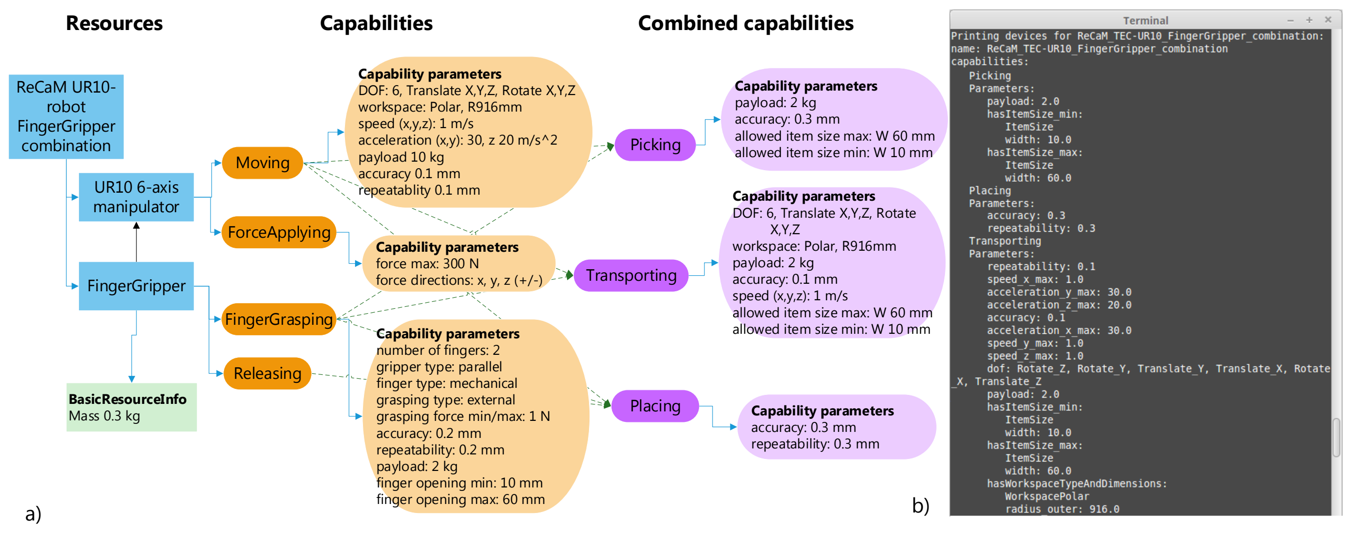


Capability Model (OWL)
imports
Resource Model (OWL)
imports
Resource Model instances (OWL)
Capability Query Library (CQL)
1. Reads in the device combinations, involved devices and their capabilities
imports
2. Reads in the rules
3. Runs the rules
4. Saves the results
5. Uses as input for decision making
Combinations -file (OWL)
Parameter Rules Ontology (contains the SPIN rules) (OWL)

Fig. 3. Combined capability calculation procedure.

4. CASE

We have tested the MaRCO model and SPIN rules in a simple case application. **Fig. 4**a shows example resource combination consisting of UR10 6-axis robot and a 2-finger gripper. In the middle, it shows the (simple) capabilities of these resources and their parameters. On the right, it shows the expected results of applying the combined capability rules. **Fig. 4b** is a screenshot form the CQL command line application showing the inference results produced by the presented SPIN rules.



```
Printing devices for ReCaM_TEC-UR10_FingerGripper_combination:
name: ReCaM_TEC-UR10_FingerGripper_combination
capabilities:
   Picking
   Parameters:
      payload: 2.0
      hasItemSize_min:
         ItemSize
         width: 10.0
      hasItemSize_max:
         ItemSize
         width: 60.0
   Placing
   Parameters:
      accuracy: 0.3
      repeatability: 0.3
   Transporting
   Parameters:
      repeatability: 0.1
      speed_x_max: 1.0
      acceleration_y_max: 30.0
      acceleration_z_max: 20.0
      accuracy: 0.1
      acceleration_x_max: 30.0
      speed_y_max: 1.0
      speed_z_max: 1.0
      dof: Rotate_Z, Rotate_Y, Translate_Y, Translate_X, Rotate
_X, Translate_Z
      payload: 2.0
      hasItemSize_min:
         ItemSize
         width: 10.0
      hasItemSize_max:
         ItemSize
         width: 60.0
      hasWorkspaceTypeAndDimensions:
         WorkspacePolar
         radius_outer: 916.0
```

Fig. 4. a) Example resources and their capabilities; b) Combined capabilities calculated based on the SPIN rules in CQL.

5. CONCLUSIONS

We presented an approach for calculating the combined capability parameters based on the parameters of the lower level capabilities. An OWL-based ontology was used to model the capability- and resource-related concepts, properties and relations between these concepts. As OWL is not able to infer and assert new instances to the ontology, nor to perform complex arithmetic operations, we used SPARQL Inferencing Notation (SPIN) to extend the reasoning abilities of pure OWL-ontology and to perform the needed calculations as well as instance and property assertions.

Research publications using SPIN in practical applications are still rare, whereas use of SWRL is more common. However, SPIN has many advantages over SWRL, including its expressiveness and meta-modeling abilities. The ability to create templates and functions reduces the effort of creating rules. Furthermore, SPIN is based on established SPARQL, which has good tool support (e.g. engines and databases).

The presented approach will be used as part of the capability matchmaking process, which aims to support rapid system configuration scenarios generation for specific product requirements. In this context, the rules are used in two scenarios: 1) Inferring and asserting the parameter values for the capabilities of an existing resource combination; 2) Searching for suitable combinations of resources for a specific capability requirement. In our future work, we will concentrate on defining similar rules for comparing the product requirements against the combined capabilities that were inferred by utilizing the method presented in this paper.

Often in real production environment the properties of the combined capabilities emerge as a behavior of the machine or station as a whole in a certain context and environment, and they cannot be decomposed into the properties of the various components (i.e. simple capabilities). Furthermore, some of the capabilities depend on the physical location between the combined resources. This information is not handled with the Capability Model, and can not thus be taken into consideration. Even though the combined capability rules can sometimes produce only crude estimations of the combined capabilities, we expect this approach to reduce the workload of a system designer and reconfiguration planner. Our approach can automatically filter out unsuitable resources and suggest viable alternative configurations for the system design and reconfiguration from large search spaces. This is expected to reduce the time used for searching and evaluating alternative resources, and to open up possibility to find new unexpected solutions.

REFERENCES

Aarnio, P., Vyatkin, V. and Hastbacka, D. (2016). Context modeling with situation rules for industrial maintenance. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, p. 9.

Ameri, F. and McArthur, C. (2014). Semantic rule modelling for intelligent supplier discovery. *International Journal of Computer Integrated Manufacturing*, Vol. 27 No. April, pp. 570–590.

Apache Software Foundation (2017). Apache Jena – A free and open source Java frawework for building Semantic Web and Linked Data applications. Available in: https://jena.apache.org/ [Accessed 10.8.2017].

Horrocks, I. et al. (2004). SWRL: A Semantic Web Rule Language – Combining OWL and RuleML. *W3C Member Submission.* Available in: http://www.w3.org/Submission/SWRL/ [Accessed 10.3.2016].

Järvenpää, E., Siltala, N., and Lanz, M. (2016). Formal Resource and Capability Descriptions Supporting Rapid Reconfiguration of Assembly Systems. In *Proceedings of the 12th Conference on Automation Science and Engineering, and International Symposium on Assembly and Manufacturing*, IEEE, 2016. p. 120-125.

Järvenpää, E., Lanz, M., and Siltala, N. (2018). Formal Resource and Capability Models supporting Re-use of Manufacturing Resources. *Procedia Manufacturing*, Vol. 19, pp. 87-94.

Knublauch, H. (2016). *The TopBraid SPIN API*. Available in: http://topbraid.org/spin/api/ [Accessed 1.4.2017].

Knublauch, H., Hendler, J.A., and Idehen, K. (2011). SPIN – Overview and motivation. *W3C Member Submission.* Available in: https://www.w3.org/Submission/spin-overview/ [Accessed 15.12.2016].

Meditskos, G., Dasiopoulou, S., Efstathiou, V. and Kompatsiaris, I. (2013). SP-ACT : A Hybrid Framework for Complex Activity Recognition Combining OWL and SPARQL Rules. *IEEE Workshop on Context Modeling and Reasoning 2013*, pp. 25–30.

Sirin, E., Parsia, P. Cuenca Grau, B., Kalyanpur, A., and Katz, Y. (2007). Pellet: A Practical OWL-DL Reasoner. *Journal of Web Semantics*, 5(2), 51–53.

SPIN working group. (2017). SPIN – SPARQL Inferencing Notation. Available in: http://spinrdf.org/. [Accessed 15.10.2017].

Wiendahl, H.-P. et al. (2007). Changeable Manufacturing - Classification, Design and Operation. *CIRP Annals*, 56, 783-809.

W3C (2004). *OWL Web Ontology Language – Reference*, Available in: http://www.w3.org/TR/owl-ref/ [Accessed 1.11.2015].

W3C (2008). *SPARQL Query Language for RDF*. Available in: http://www.w3.org/TR/rdf-sparql-query/ [Accessed 10.3.2016].