Empowering the Model-driven Engineering of Robotic Applications using Ontological Semantics and Reasoning

Stefan Zander[1], Nadia Ahmed[1] and Yingbing Hua[2]
[1]*FZI Research Center for Information Technology, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany*
[2]*Karlsruhe Institute of Technology, Engler-Bunte-Ring 8, 76131 Karlsruhe, Germany*

Keywords: Knowledge Representation, Semantic Technologies, Ontologies, Cyber-physical Systems, Robotics.

Abstract: This work discusses two scenarios in which the model-driven engineering of robotic applications can be improved using ontological semantics and reasoning. The objective of the presented approach is to facilitate reuse and interoperability between cooperating software and hardware components. Central to the presented approach is the usage of ontologies and description logics as knowledge representation frameworks for the axiomatic description of component metadata models. In the first scenario, we show how application templates can be created using the concept of placeholders in which requirements for integrating external components can be axiomatically specified and eligible components can be computed using subsumption reasoning. The second scenario extends this idea for the inference of compatibilities between cooperating components. The practical applicability of the approach is demonstrated by a concrete use case from the ReApp project.

1 INTRODUCTION

The development of robot applications is usually an expensive and time-consuming task, resulting from the inherent heterogeneity and complexity of involved elements (data, algorithms, interfaces, protocols etc.) and the required technical and domain expert knowledge. These aspects, among others, hinder the broad usage of robotic systems, in particular in small and medium-sized enterprises (SMEs) and industries with high production variability (cf. the ReApp project[1]). In order to tackle these issues, the integration of model-driven engineering (MDE) principles into the software- and tool-development processes of robotic applications revealed promising improvements (cf. (Schlegel et al., 2009; Alonso et al., 2010; Gherardi and Brugali, 2014)). However, the positive effects of MDE approaches can be further increased, if they are synthesized with ontological semantics and formal reasoning methods, as new forms of tool-support and assistance can be provided to software developers, system integrators and end users likewise (Zander et al., 2016).

In this paper, we present two scenarios that corroborate our hypothesis that ontologies used as knowledge representation framework can improve the model-driven engineering of robotic applications in terms of reusability and utilization. In one scenario, we show how the utilization and interoperability of software components can be improved by axiomatically describing functional requirements for external components that are needed by a given component in order to provide its full service (e.g., a path planing component combined with an object detection software). We therefore define the concept of application templates in which axiomatically expressed requirements using description logics (cf. (Baader et al., 2003; Krötzsch et al., 2014)) are encoded in component descriptions (Gil, 2005). These axioms can be processed by a reasoner in order to deduce eligible components, which are hosted e.g. in application marketplaces[2]. This concept allows a software component or robotic application to explicitly express necessary requirements for external or third-party applications it requires for its correct execution on a functional level (see (Zander and Awad, 2015)). These requirements can being specified by the developers at design time to exploit formal reasoning for inferring recommendations of suitable components.

A second scenario demonstrates how *compatibil-*

[1]www.reapp-projekt.de

[2]In the context of the ReApp project, a marketplace for robotic components was developed, the so-called *ReApp-Store* (Bastinos et al., 2014).

Zander, S., Ahmed, N. and Hua, Y.
Empowering the Model-driven Engineering of Robotic Applications using Ontological Semantics and Reasoning.
DOI: 10.5220/0006086201920198
In *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016) - Volume 2: KEOD*, pages 192-198
ISBN: 978-989-758-203-5

ities between cooperating software components can be automatically inferred by a reasoner on basis of the formal model-theoretic semantics employed by the ontology language upon which a component's metadata model is described. In contrast to many rule-based approaches in which compatibility is determined by evaluating expressions encoded in rule bases, we compute compatibility between components through subsumption reasoning, i.e., by interpreting the axioms encoded in the terminological part of a knowledge base (cf. (Zander and Awad, 2015)).

We employ the Resource Description Framework (RDF)[3] (W3C, 2014) as knowledge representation framework for encoding ontological information and the Web Ontology Language (OWL)[4] (Patel-Schneider et al., 2009; W3C OWL Working Group, 2012) for describing a model's formal semantics using description logics (DL) fragments as they exhibit well-understood reasoning complexity and tractability (cf. (Baader et al., 2003; Krötzsch et al., 2014)).

2 RELATED WORKS

2.1 Model-driven Engineering

During the evolution of software engineering, model-driven software development has achieved remarkable success. In the domain of robotics, a distinction can be made between approaches that use existing general purpose languages for modeling tasks and those that use domain specific languages (DSL) (Brugali, 2015). SmartSoft (Schlegel et al., 2009) supports both platform-independent and platform-dependent artifacts and provides a model-driven toolchain that features a coordination language called *SmartTCL* to model runtime communication orchestration between components. V^{3}CMM (Alonso et al., 2010) provides a meta-model for robotic component architectures that provides three different UML views of a system regarding structure, coordination, and algorithms. The HyperFlex toolchain (Gherardi and Brugali, 2014) demonstrates how reference architectures can be used for the development of robotic applications using component models that abstract from a specific implementation framework. The idea is to further enable reuse for entire mature subsystems. Examples for robotic DSL are the Architecture Analysis and Design Language AADL (Ramaswamy et al., 2014a), the RobotML language developed by the french research project PROTEUS (Farges, 2009), the MontiArcAutomation framework (Ringert et al., 2013), and the CoSTAR framework modeling language (Guerin et al., 2015). AADL plays a vital role in the SafeRobots framework (Ramaswamy et al., 2014c; Ramaswamy et al., 2014b) as a solution space modeling language.

Although all these approaches enable modeling of robotic components and systems to a certain extent, yet most of them neglect the semantic interoperability, since there is barely any explicit semantic interpretation of component descriptions, which means the requirements and interactions between them are only defined syntactically using proprietary and varying grammars. To be able to use certain frameworks, a user is required to dive deeply into the concepts of the underlying technologies, which significantly reduces the practicability of the approach.

2.2 Ontologies in Robotics

In the Web of Data (aka semantic Web), ontologies have proven their usefulness for describing concepts and relationships between resources as they serve as crucial constituents of a semantic interoperability infrastructure built upon standards proposed by the W3C. A core ontology containing common knowledge about robotics and automation was developed by the recently established working group "ontologies for robotics and automation" within the IEEE-RAS (Prestes et al., 2013) committee. In addition to the Semantic Sensor Network Ontology (Compton et al., 2012), which is widely used in the cyber-physical systems and IoT domain, Ion-Mircea and Gerd (Diaconescu and Wagner, 2014) developed a sensor and actuator systems ontology specifically for the Web of Things. Carbonera et al. (Carbonera et al., 2013) defined an ontology to describe robot positioning as part of the core ontology concept by the IEE RAS working group. Nilsson et al. (Nilsson et al., 2009) developed a robotic component ontology to demonstrate the possible improvements and gains in robustness when designing robotic systems using task-oriented models. An extensive investigation into the benefits of combining ontologies and model-driven approaches is carried out by Assmann et al. (Aßmann et al., 2006). Semantic technologies could also be found in model-driven robotic frameworks; PROTEUS (Lortal G, 2011), for example, introduced a methodology for using ontologies on the system level where knowledge transfer between experienced users is supported. The lack of fine-grained information models limits the platform's usability.

[3]https://www.w3.org/RDF/
[4]https://www.w3.org/OWL/

3 APPROACH

In the first part of this section, we describe how application templates and placeholders can be axiomatically expressed in form of ABox, TBox and RBox axioms. The second part demonstrates the usage of terminological knowledge for computing compatibilities between components using subsumption reasoning. A central concept of both parts is the notion of *formal requirements*, which we elaborate in detail in the first part.

3.1 Expressing Requirements for Application Templates

In this work, we define an application template as a composition of placeholders that can be filled by concrete software or hardware components. An application template is composed of at least one placeholder which is expressed as a set of ABox axioms that follow a specific schema (see Axioms 1-6) and define requirements using terms from hardware, software and capability ontologies[5]. For example, a robotic application for picking-up an object from a conveyor belt can be specified using the template concept in order to express the requirement of three necessary components including their respective capabilities: (1) a *camera* that captures the image of the object to be picked; (2) a *position detection* component that detects the target position for controlling the robot movement; (3) a *path calculation* component that calculates the trajectory for the robot from its initial to the target position. A strong point of the presented approach is that it allows eligible components to be expressed not only directly via their respective types but rather by the functionalities they provide. With regard to the previous example, this would be axiomatically expressed as follows:

$$\mathsf{2DCamera} \sqsubseteq \exists \mathsf{hasCapability}.\{\mathsf{2DImageCapturing}\}$$
$$\mathsf{Vision} \sqsubseteq \exists \mathsf{hasCapability}.\{\mathsf{PositionDetection}\}$$
$$\mathsf{PathCalculation} \sqsubseteq$$
$$\exists \mathsf{hasCapability}.\{\mathsf{TrajectoryCalculation}\}$$

The above mentioned functionalities are encoded in the terminological part of the capability ontology[6] and are represented via the classes: ImageCapturing, PositionDetection and TrajectoryCalculation. Capabilities are linked to hardware and software components via *role restriction axioms* that act along a specified property (hasCapability). This axiom type can be used for the formulation of placeholders; the placeholder for a component that offers an ImageCapturing capability can be expressed as follows:

$$\mathsf{RobotApplication}(\mathsf{application1}) \tag{1}$$
$$\mathsf{PlaceholderExpression}(\mathsf{p1}) \tag{2}$$
$$\mathsf{hasPlaceholder}(\mathsf{application1}, \mathsf{p1}) \tag{3}$$
$$\mathsf{Requirement}(\mathsf{req1}) \tag{4}$$
$$\mathsf{requires}(\mathsf{p1}, \mathsf{req1}) \tag{5}$$
$$\mathsf{requiresCapability}(\mathsf{req1}, \{\mathsf{ImageCapturing}\}) \tag{6}$$
$$\{\mathsf{ImageCapturing}\} \sqsubseteq \mathsf{Capability} \tag{7}$$
$$\mathsf{requiresCapability} \sqsubseteq \mathsf{hasCapability} \tag{8}$$

Axiom 1 states that a concrete robot application-template represented by application1-individual, member of the class RobotApplication, has a placeholder (Axiom 2) represented by p1-individual member of the class PlaceholderExpression (Axiom 2). This p1-individual is related to a member of Requirement-class denoted by req1 via requires-property (Axiom 3). Additionally to express the requirement information by mean of functionalities, we define the property requiresCapability as expressed in Axiom 6, this property relates req1-individual to the DL nominal ImageCapturing. Axiom 7 denotes that ImageCapturing is subsumed by Capability. Additionally, Axiom 8 transforms the property requiredCapability into a hasCapability expression when inferring eligible components via subsumption reasoning.

Assuming a robotic system contains a specific hardware component (e.g. Sick IVC-2D camera), which is represented by the individual myIVC2DCamera and classified as a member of the 2DCamera class. Based on the following axioms

$$\mathsf{2DCamera}(\mathsf{myIVC2DCamera})$$
$$\mathsf{2DCamera} \sqsubseteq \exists \mathsf{hasCapability}.\{\mathsf{2DImageCapturing}\}$$

a reasoner is able to infer that each member of 2DCamera is also a member of the abstract class $\exists\mathsf{hasCapability}.\{\mathsf{2DImageCapturing}\}$ and offers the capability 2DImageCapturing, i.e., it participates in a hasCapability-role with the nominal represented by 2DImageCapturing.

$$\exists \mathsf{hasCapability}.\{\mathsf{2DImageCapturing}\} \hookleftarrow$$
$$(\mathsf{myIVC2DCamera})$$

Since 2DImageCapturing is subsumed by ImageCapturing (as stated in the capability ontology), a reasoner can infer that hardware components, which are

[5] In the course of the ReApp project (www.reapp-projekt.de) two ontologies for classifying hardware and software components together with one ontology for expressing capabilities of such components have been developed and published (see http://ipe-id.fzi.de/ontologies/reapp/).

[6] http://ipe-id.fzi.de/ontologies/reapp/doc/Capability.htm

members of the class 2DCamera fulfill the placeholder's requirements expressed in Axioms 1-8. Besides, if a component provides a capability, e.g., Fast2DImageCapturing, which is subsumed by 2DImageCapturing, the reasoner can also deduce that this component fulfills the placeholder's requirements.

Furthermore, placeholder's requirements are also extended with constraints in form of ABox axioms that eligible component must satisfy. In order to express within a placeholder that the *image resolution value must be at least equal to 10 Mega Pixel* we define the following axioms.

$$\mathsf{Constraint}(\mathsf{cons1}) \tag{9}$$
$$\mathsf{hasConstraint}(\mathsf{req1}, \mathsf{cons1}) \tag{10}$$
$$\mathsf{requiresAttribute}(\mathsf{cons1}, \{\mathsf{Resolution}\}) \tag{11}$$
$$\mathsf{requiresAttribute} \sqsubseteq \mathsf{hasAttribute} \tag{12}$$
$$\{\mathsf{Resolution}\} \sqsubseteq \mathsf{Attribute} \tag{13}$$
$$\mathsf{hasOperator}(\mathsf{cons1}, \{\mathsf{GreaterThan}\}) \tag{14}$$
$$\{\mathsf{GreaterThan}\} \sqsubseteq \mathsf{Operator} \tag{15}$$
$$\mathsf{hasConstraintValue}(\mathsf{cons1}, \text{‘}10\text{‘}) \tag{16}$$
$$\mathsf{hasUnitOfMeasurement}(\mathsf{cons1}, \{\mathsf{MegaPixel}\}) \tag{17}$$

Axioms 9 and 10 state that req1-individual (previously defined in Axiom 4) is related to cons1, member of the class Constraint via hasConstraint-property. Since a constraint expression consists of an attribute, an operator and a value, cons1-individual, is related to individuals, members of the class Attribute via requiresAttribute-property (Axiom 12), members of the class Operator via hasOperator-property (Axiom 14) and members of the class Value via hasConstraintValue-property (Axiom 16). Operators are expressed at TBox level such as the class GreaterThan (Axiom 15). Axiom 16 and 17 state that the value of cons1-individual is equal to 10 and is related to MegaPixel-nominal via the hasUnitOfMeasurement-property. The retrieval of components that fulfill a placeholder's requirements is performed by comparing the attribute values that a component provide with the constraints required by a given placeholder. Whereby, concrete instances of attributes that are appended to placeholder's requirements as well as to specific components are classified by the reasoner. The fact that a specific component, classified as a camera, has the attribute resolution can be encoded as follows:

$$\mathsf{Sensor} \sqsubseteq \exists \mathsf{hasAttribute}.\{\mathsf{Resolution}\} \tag{18}$$
$$\{\mathsf{Resolution}\} \sqsubseteq \mathsf{Attribute} \tag{19}$$
$$\mathsf{Camera} \sqsubseteq \mathsf{Sensor} \tag{20}$$

Axiom 18 denotes that members of the class Sensor have at least one relationship to the nominal Resolution, which is a subconcept of Attribute (Axiom 19). Since Camera is a subclass of Sensor, the reasoner deduces that individuals, members of the class Camera are also related to individuals of the class resolution via hasAttribute-property. Summarized, each component classified as a camera also has the resolution attribute.

3.2 Semantic Compatibility

After eligible components that satisfy a placeholder's requirements are inferred by a reasoner, an additional test is required to assure that two cooperating components are compatible in terms of their technical interface specifications. For example, a visual odometry component subscribes to a ROS[7] image-topic published by a camera access component (the camera driver). Connecting these two components via the image-topic merely based on syntactical parameters (e.g. the name of the ROS topic) does in most cases not ensure their technical compatibility and satisfiability of functional requirements; e.g., the visual odometry component might require a depth image while the camera component provides an intensity image. In order to infer whether an eligible component also satisfies technical requirements, requirement information is appended to the component model.

Requirement information related to a placeholder (as described in 3.1) is also used to describe requirements for components in order to check whether an eligible component satisfies the requirements of a given component. Therefore, requirement information is appended to a component analogically to placeholders (see Axioms 4-6). For example, asserting that a component requires a 3DIntensityImageCapturing capability is encoded as follows:

$$\mathsf{SoftwareComponent}(\mathsf{objectDetection}) \tag{21}$$
$$\mathsf{Requirement}(\mathsf{req2}) \tag{22}$$
$$\begin{aligned}&\mathsf{requiresCapability}(\mathsf{objectDetection},\\&\quad\{\mathsf{3DIntensityImageCapturing}\})\end{aligned} \tag{23}$$

Axiom 21 states that a component represented by an individual objectDetection is a member of the class SoftwareComponent and is related to the req2 individual, which is a member of the class Requirement (Axiom 22) via the requiresCapability-property (Axiom 23). In the given case, the class 3DIntensityImageCapturing is a subclass of ImageCapturing; a reasoner thus is able to infer additional requirement information related to the ImageCapturing class. Requirement information for a concrete component are also extended by constraints information and are axiomatically expressed as described in Axiom 9-10.

[7]http://wiki.ros.org/Messages

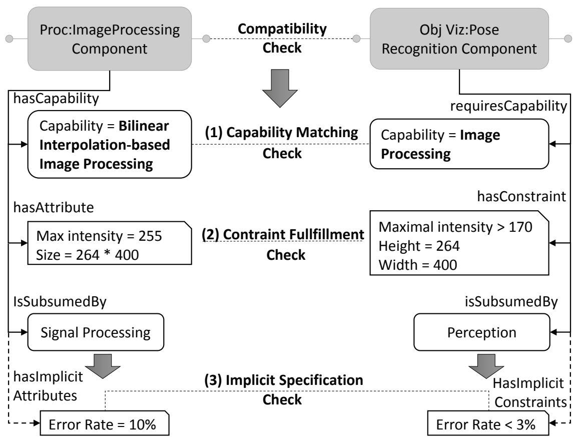

Figure 1: Compatibility calculation between two exemplary components on the basis of their attributes together with the different levels of checks: (1) capability matching, (2) constraints fulfillment, and (3) implicit specifications checks.

Therefore, a compatibility check can be performed by comparing the provided attribute values with the constraints required by a given component.

In order to check the compatibility between an image processing component and a pose recognition component (see Figure 1), three different levels of checks are required. At the first level, the capability of the provided side ImageProcessing must be subsumed by the capability of the component at the required side PoseRecognition. Secondly, the constraints must be fulfilled. That means the provided values of the attributes of ImageProcessing must satisfy the constraints specified by the required side. On this level, a class membership of attribute individuals (denoted in Figure 1 as Max intensity and Maximal intensity) is also calculated in order to identify that they belong to the same Attribute-class. In order to unify the representation of attributes, they are also represented as TBox axioms.

Afterward, a numerical comparison is performed in order to check whether the values of the provided attributes fulfill the given constraints. Finally, implicitly deduced information about additional capabilities such as inferred capability of the component SignalProcessing is also used for further checks. Implicit constraint information deduced from the constraint of the subsuming capability PoseRecognition can also be used for additional checks.

As depicted in Figure 1, given an image processing component, represented by proc-individual, member of the class SoftwareComponent, this component provides the capability BilinearInterpolation-basedImageProcessing. On the other side, given a pose recognition component represented by an individual denoted by ObjViz, member of the class SoftwareComponent, this component requires the capability ImageProcessing. The required capability is formalized as illustrated in Axioms 4-8 and 21-23. Since BilinearInterpolation-basedImageProcessing is subsumed by ImageProcessing, the capability level of both components is matching. Additionally, the reasoner infers implicit attributes related to the subsuming class SignalProcessing such as the ErrorRate is equal to 10%. Analogically, on the required side (pose recognition), implicit constraints are derived from the subsuming capability Perception. For example, the error rate must be smaller than 3%. This improves the checking process by delivering more knowledge about the properties that a component exhibits or requires, which are not explicitly stated at design time.

4 USE CASE

In this section, we demonstrate the applicability of the proposed semantically enhanced model-driven approach by means of a use case, in which the application developer will be assisted during the engineering phase of a robotic system in the electronic industry. In this use case, a robotic system should solder LED stripes on printed circuit boards (PCB) while the human operator holds the connection wires between the soldering points, thus also enabling a close human to machine cooperation[8].

Aiming at a flexible automation solution that can be reconfigured for potential soldering applications with different process requirements, the application developer will make a template for the robotic system comprising various hardware and software components. For example: A robot arm should be used as actuator to drive the soldering tool to the target position. By utilizing the placeholder concept, the application developer can axiomatically formulate necessary requirements, e.g., an actuator with 3-dimensional movement capability and add them to the semantic metadata model of its application:

$$\mathsf{PlaceholderExpression}(\mathsf{p1}) \tag{24}$$
$$\mathsf{Requirement}(\mathsf{req1}) \tag{25}$$
$$\mathsf{requires}(\mathsf{p1}, \mathsf{req1}) \tag{26}$$
$$\mathsf{requiresCapability}(\mathsf{req1}, \mathsf{3DMovement}) \tag{27}$$
$$\mathsf{3DMovement} \sqsubseteq \mathsf{Capability} \tag{28}$$
$$\mathsf{requiresCapability} \sqsubseteq \mathsf{hasCapability} \tag{29}$$

The placeholder will be semantically processed and the reasoner is capable to find suitable com-

[8]This is one of the three use cases in the ReApp project.

ponents from a knowledge base that fulfill the requirements, e.g. a robot of type UR5 and UR10. Moreover, the application developer can further restrict the component to have a certain accuracy (Repeatability $<= 0.02$mm) for the soldering process and a minimum payload to carry the soldering tool (Payload $>= 7$kg). Such constraints can be by defined axiomatically as follows (excerpt):

$$\mathsf{Requirement}(\mathsf{req1}) \tag{30}$$
$$\mathsf{hasConstraint}(\mathsf{req1},\mathsf{cons1}) \tag{31}$$
$$\mathsf{Constraint}(\mathsf{cons1},\mathsf{cons2}) \tag{32}$$
$$\mathsf{requiresAttribute}(\mathsf{cons1},\{\mathsf{Repeatability}\}) \tag{33}$$
$$\mathsf{hasOperator}(\mathsf{cons1},\{\mathsf{SmallerOrEqualThan}\}) \tag{34}$$
$$\mathsf{hasConstraintValue}(\mathsf{cons1},\text{‘}0.02\text{‘}) \tag{35}$$
$$\mathsf{hasUnitOfMeasurement}(\mathsf{cons1},\{\mathsf{mm}\}) \tag{36}$$
$$\mathsf{hasConstraint}(\mathsf{req1},\mathsf{cons2}) \tag{37}$$
$$\mathsf{requiresAttribute}(\mathsf{cons2},\{\mathsf{Payload}\}) \tag{38}$$
$$\mathsf{hasOperator}(\mathsf{cons2},\{\mathsf{GreaterOrEqualThan}\}) \tag{39}$$
$$\mathsf{hasConstraintValue}(\mathsf{cons2},\text{‘}7\text{‘}) \tag{40}$$
$$\mathsf{hasUnitOfMeasurement}(\mathsf{cons2},\{\mathsf{kg}\}) \tag{41}$$

Both candidates UR5 and UR10 fulfill the *Repeatability* constraint, yet not the *Payload* constraint, since UR5 can only carry tools up to 5 kg. As a result, the reasoner will infer that the UR5 does not satisfy all constraints and answers a placeholder query only with the robot UR10.

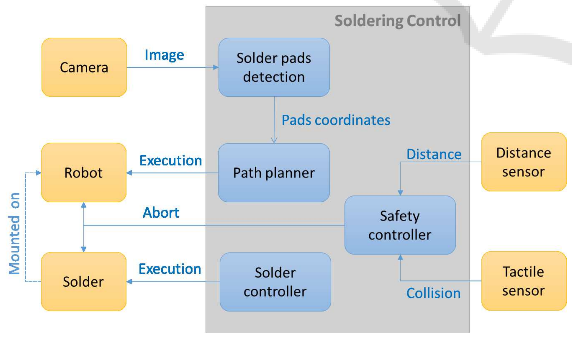

Figure 2: An interaction model of the components in the soldering application.

To ensure the safety of the production, the application developer needs a safety controller which has to monitor possible collisions between solder tip and obstacles, and will be fed by on-line distance and tactile information (Figure 2). Moreover, to assure that the emergency stop can be triggered in time, if any collision is about to occur, the safety controller requires incoming sensor data with a least frequency of 500 Hz. During the engineering, to connect the safety controller with appropriate sensor drivers, the data type of published distance and tactile information must be standard ROS message types to guarantee syntactic compatibility; On the other hand, the frequency of the sensor drivers can be modeled additionally as functional requirement via formal semantics:

$$\mathsf{hasConstraint}(\mathsf{req2},\mathsf{cons3}) \tag{42}$$
$$\mathsf{hasAttribute}(\mathsf{cons3},\{\mathsf{Frequency}\}) \tag{43}$$
$$\mathsf{hasOperator}(\mathsf{cons3},\{\mathsf{GreaterOrEqualThan}\}) \tag{44}$$
$$\mathsf{hasConstraintValue}(\mathsf{cons3},\text{‘}500\text{‘}) \tag{45}$$
$$\mathsf{hasUnitOfMeasurement}(\mathsf{cons3},\{\mathsf{Hz}\}) \tag{46}$$

The compatibility computation thus provide results upon which a decision can be made whether a given sensor driver fulfills the specified requirements.

5 CONCLUSION

This work demonstrates how the model-driven engineering of robotic applications can be synthesized with ontological semantics and reasoning in order to enhance the reuse and interoperability of hardware- and software components. We used description logic grounded ontologies as knowledge representation framework as they employ well-understood reasoning complexity and tractability. In a first scenario, the concepts of application templates and placeholders were introduced that allow for the axiomatic expression of requirements and constraints external components have to satisfy for collaboration. By interpreting the formal semantics of those axioms, eligible components can be inferred using subsumption reasoning. We extended this idea in a second scenario and illustrated, how compatibilities between cooperating components can be inferred by a reasoner based on the axiomatic descriptions of constraints and features of components. A distinguishing feature of the presented approach is that it makes extensive uses of terminological knowledge in order to fully exploit the formal model-theoretic semantics of the underlying ontology language in the reasoning process rather than using rule-based language frameworks.

ACKNOWLEDGEMENTS

This work has been partially funded by the German Federal Ministry for Economic Affairs and Energy through the project ReApp (no. 01MA13001) and by the Ministerium für Wirtschaft, Arbeit und Wohnungsbau Baden-Württemberg through the Digital Innovation Center (DIZ Digitales Innovationszentrum).

REFERENCES

Alonso, D., Vicente-chicote, C., Ortiz, F., Pastor, J., and Alvarez, B. (2010). V3CMM: a 3-View Component Meta-Model for Model-Driven Robotic Software Development. *Journal of Software Engineering for Robotics (JOSER)*, 1(January):3–17.

Aßmann, U., Zschaler, S., and Wagner, G. (2006). Ontologies, meta-models, and the model-driven paradigm. In *Ontologies for software engineering and software technology*, pages 249–273. Springer.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Bastinos, A. S., Haase, P., Heppner, G., Zander, S., and Ahmed, N. (2014). ReApp Store - a semantic appstore for applications in the robotics domain. In *International Semantic Web Conference (Industry Track)*.

Brugali, B. D. (2015). Model-Driven Software Engineering in Practice. pages 155–166.

Carbonera, J. L., Fiorini, S. R., Prestes, E., Jorge, V. a. M., Abel, M., Madhavan, R., Locoro, A., Goncalves, P., Haidegger, T., Barreto, M. E., and Schlenoff, C. (2013). Defining positioning in a core ontology for robotics. *IEEE International Conference on Intelligent Robots and Systems*, pages 1867–1872.

Compton, M., Barnaghi, P. M., Bermudez, L., Garcia-Castro, R., Corcho, Ó., Cox, S. J. D., Graybeal, J., Hauswirth, M., Henson, C. A., Herzog, A., Huang, V. A., Janowicz, K., Kelsey, W. D., Phuoc, D. L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K. R., Passant, A., Sheth, A. P., and Taylor, K. (2012). The ssn ontology of the w3c semantic sensor network incubator group. *Journal of Web Semantics*, 17:25–32.

Diaconescu, I.-M. and Wagner, G. (2014). Towards a general framework for modeling, simulating and building sensor/actuator systems and robots for the web of things. In *First Workshop on Model-Driven Robot Software Engineering (MORSE)*.

Farges, J.-L. (2009). Robotic Ontology and Modelling - 3rd version.

Gherardi, L. and Brugali, D. (2014). Modeling and reusing robotic software architectures: The HyperFlex toolchain. *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6414–6420.

Gil, Y. (2005). Description logics and planning. *AI Magazine*, 26(2):73–84.

Guerin, K. R., Lea, C., Paxton, C., and Hager, G. D. (2015). A Framework for End-User Instruction of a Robot Assistant for Manufacturing. pages 6167–6174.

Krötzsch, M., Simančík, F., and Horrocks, I. (2014). Description logics. *IEEE Intelligent Systems*, 29:12–19.

Lortal G, Dhouib S, G. S. (2011). Integrating ontological domain knowledge into a robotic dsl. In *2010 international conference on Models in software engineering MODELS?10*, pages 401–414.

Nilsson, a., Muradore, R., Nilsson, K., and Fiorini, P. (2009). Ontology for robotics: A roadmap. *2009 International Conference on Advanced Robotics*.

Patel-Schneider, P. F., Motik, B., and Grau, B. C. (2009). OWL 2 Web Ontology Language Direct Semantics. W3C recommendation, W3C.

Prestes, E., Carbonera, J. L., Fiorini, S. R., Jorge, V. A. M., Abel, M., Madhavan, R., Locoro, A., Goncalves, P., Barreto, M. E., Habib, M., Chibani, A., Gérard, S., Amirat, Y., and Schlenoff, C. (2013). Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11):1193 – 1204. Ubiquitous Robotics.

Ramaswamy, A., Monsuez, B., and Tapus, A. (2014a). Architecture Modeling and Analysis Language for Designing Robotic Architectures. *International Conference on Control Automation Robotics & Vision (ICARCV)*, 2014(December):10–12.

Ramaswamy, A., Monsuez, B., and Tapus, A. (2014b). SafeRobots : A Model Driven Framework for Developing Robotic Systems. (Iros):1517–1524.

Ramaswamy, A., Monsuez, B., and Tapus, A. (2014c). SafeRobots: A Model-Driven Approach for Designing Robotic Software Architectures. *Collaboration Technologies and Systems (CTS), 2014 International Conference on*, pages 131 – 134.

Ringert, J. O., Rumpe, B., and Wortmann, A. (2013). MontiArcAutomaton: Modeling Architecture and Behavior of Robotic Systems. *Workshops and Tutorials Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA*, pages 10–12.

Schlegel, C., Hassler, T., Lotz, A., and Steck, A. (2009). Robotic software systems: From code-driven to model-driven designs. In *Advanced Robotics (ICAR) 2009 Intl. Conf. on*, pages 1–8.

W3C (2014). RDF 1.1 Concepts and Abstract Syntax. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

W3C OWL Working Group (2012). OWL 2 Web Ontology Language Document Overview (Second Edition) - W3C Recommendation 11 December 2012.

Zander, S. and Awad, R. (2015). Expressing and reasoning on features of robot-centric workplaces using ontological semantics. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*. to be published.

Zander, S., Heppner, G., Neugschwandtner, G., Awad, R., Essinger, M., and Ahmed, N. (2016). A model-driven engineering approach for ROS using ontological semantics. *CoRR*, abs/1601.03998.