DOI: 10.3217/978-3-85125-524-9-07

Task-Dependent Configuration of Robotics Systems

Alexander Pagonis[1] and Clemens Mühlbacher[1] and Gerald Steinbauer[1] and Stefan Gspandl[2] and Micheal Reip[2]

***Abstract*— To solve a task, a robotics system uses several different hardware and software components. Each of these components solves a specific subtask to allow the overall task to be solved. Thus, the proper selection of the set of components is crucial for the success of performing a task. This selection can become complex if one needs to consider that each of these components has its own dependencies which need to be fulfilled to work properly. Due to this complexity, the proper selection of components is time-consuming and error prone. Additionally, domain knowledge is necessary to consider all dependencies correctly.**

To proper choose the components without the need of a domain expert one can follow a model based approach. In this paper, we show how such a model-based approach can be used. We present a tool that, based on a domain model, automatizes the selection of the necessary components to implement a set of given tasks. Due to this automatic selection mechanism, one can either simply check if a robotic system can perform a task or which components need to be added to allow the robot to perform the given task.

I. Introduction

A robotics system consists of several hardware and software components which interact with each other to achieve a given task. The selection of the hardware and software components is often done by a domain expert, ensuring that the task can be fulfilled with the given selection. This is a time-consuming task, as one needs to know which dependency each component has, e.g. a computer vision algorithm depends on a camera but does not specify which camera exactly. Additionally, one possible needs to consider many possibilities how a dependency can be met to find an optimal selection. Following simple scenario is used to highlight these difficulties: The task the robot must fulfill is to localize itself. One could now use a localization which is based on a laser or a localization which is based on the camera. In case there is a Kinect camera [1] available but no laser, a camera-based localization approach would probably be preferred. But one could use the depth image to simulate a laser scanner and thus use also the localization based on a laser scanner. This simple example already shows that one needs to consider several possibilities and necessary dependencies to allow a robot to solve a task.

[1]Alexander Pagonis, Clemens Mühlbacher and Gerald Steinbauer are with the Institute for Software Technology, Graz University of Technology, Graz, Austria. {apagonis,cmuehlba,steinbauer}@ist.tugraz.at
This work is partly supported by the Austrian Research Promotion Agency (FFG) under grant 843468.
[2]Stephan Gspandl and Michael Reip are with incubedIT, Hart bei Graz, Austria. {gspandl,reip}@incubedit.com

Instead of choosing the hardware and software components manually, one can follow a model-based approach for the robotic system as it was outlined in [2]. The idea is to use a model that describes the task as well as the available hardware and software components, their capabilities and dependencies. By using this model one can automatically generate a list of components that are necessary to fulfill a task. The model does not only allow to generate a list of components to fulfill a task but it also allows the robot to check if a task can be executed with the given hardware and software. Furthermore, the robot can use the model to decide which alternative software and hardware modules to use if one part of the system does not work correctly. Such a reconfiguration is of special interest if one considers complex tasks which can be achieved through several means.

In this paper, we present a tool which allows performing such a model-based configuration of a robotic system automatically. The tool can be used to derive which set of components needs to be present to allow fulfilling a task. Furthermore, the tool allows checking if a given robotic configuration can fulfill a task. Additionally, all possible component compositions that allow solving the given task can be viewed. This allows checking which alternatives are possible and which components are redundant in the system. To allow an easy configuration the tool does not only suggests possible configurations but also allows to interactively vary the given configuration. This makes the configuration process easy and allows for a quick decision on the best fitting set of components.

The remainder of the paper is organized as follows. In the next section, we discuss the design of the configuration tool. This description comprises the used knowledge base, the method which is used to derive a correct configuration, and the description of the user interface. The proceeding section discusses a simple example scenario and presents how the tool can be used. This is followed by a section discussing the limitations of the approach. Afterward, we will discuss some related research. Finally, we conclude the paper and point out some future work.

II. The Configuration Tool

As we motivate above using a model one can automate the generation of a configuration for a given task. This generation uses the model to determine the dependencies between software component and hardware component. Furthermore, the model describes the different possibilities to resolve a dependency. To ensure that the model can answer a query in a timely manner and to allow still the model to be expressive we use an Ontology for the model. With the help of the model, the tool can derive the dependencies which need to be met to fulfill a task.

To derive which configuration fulfills the dependencies a separate reasoning process is performed. This separate reasoning process uses the data contained in the model to yield a minimal configuration. Through this separation, the model can be capped simply by avoiding the "complex" reasoning for a minimal configuration.

Using the information from the ontology and the reasoning to derive a minimal configuration the tool can present a possible configuration to the user through a graphical user interface. The interface allows selecting tasks to perform, which components are used as well as which configuration would be minimal. In the remainder of the section, we will discuss each part in more detail.

A. Ontology

To model the relationships between tasks and necessary components, an ontology describing this relationship is needed. The ontology we use for the implementation is an open-source knowledge base and can be found at [3]. In this ontology, tasks are referred as capabilities. Each capability can be comprised of other basic capabilities. The ontology also describes the relationship of capabilities to hard- and software components that are needed for their implementation. Some of these components may be compulsory and do not include alternatives while others may be chosen from a pool of similar components that may all be used to fulfill the same task.

The information, stored in the ontology can be loaded and queried with an appropriate tool. We use the framework Jena [4] to load the ontology into a model. The model can be queried using the SPARQL query language. The Jena framework allows multiple ontologies to be loaded into a single model. The base ontology we use already contains references to the sub-ontologies, including descriptions of robot components. Therefore, it is enough to load the base ontology as all sub-ontologies will be loaded into the model automatically by the framework.

B. Calculation of Configuration

With the help of the ontology mentioned above, we can define the dependencies which need to be met to perform a task. The above calculate gives as a set of capabilities Cap, which can be requested to be fulfilled directly or indirectly. We use the variables X and Y in the remaining subsections for variables with the domain of capabilities $dom(X) = dom(Y) = Cap$. Besides the capabilities, we have additionally the set of components $Comp$. These components describe a software component, e.g. a laser-based localization or a hardware component, e.g. a laser scanner. We use the variable Z in the remainder of the subsection for variables with the domain of components $dom(Z) = Comp$. As the description of the components is rather abstract one needs a concrete implementation/realization of such a component, e.g. a Sick LMS100 for a laser scanner. To describe this implementation/realization of components the ontology above yields the set $ImplComp$. In the remainder of the subsection, we use the variable W for variables with the domain of the implementations of components $dom(W) = ImplComp$.

Beside the sets of possible capabilities, components and their implementation we additionally have four different functions describing the dependencies which need to be fulfilled for a capability, component and its implementation. The function $capReqCap : Cap \rightarrow 2^{Cap}$ describes which set of capabilities needs to be fulfilled by the robot to implement a certain capability. For example, the capability $liftObject$ depends on two other capabilities $moveArm, graspObject$ which is described as follows $capReqCap(liftObject) \rightarrow \{moveArm, graspObject\}$. To describe the dependencies between capabilities and components the function $capReqComp : Cap \rightarrow 2^{Comp}$ is used. For example, the capability $liftObject$ depends on two components $arm, gripper$ which is described as follows $capReqComp(liftObject) \rightarrow \{arm, gripper\}$. Each requested component can be implemented differently to link a component and an implementation we use the predicate $implComp : Comp \times ImplComp \rightarrow \{\top, \bot\}$. Like a capability a component can depend on capabilities, we use the function $compReqCap : Comp \rightarrow 2^{Cap}$ to describe this dependency. Additionally, a component can depend on other components which define through the following function $compReqComp : Comp \rightarrow 2^{Comp}$. Using this functions and the predicate we can define the dependencies which need to be met to implement a certain capability.

As we are interested in a configuration of the system which is minimal we need a specific reasoning to derive such a configuration. This is done by first extracting all dependencies of a task together with every possibility to meet this dependency. The model does not store all dependencies in a single level. Instead, some dependencies may result from other dependencies. Therefore, a recursive extraction of dependencies must be performed.

Once all these dependencies are extracted, a constraint problem can be defined to find (all) minimal configurations which fulfill the dependencies. This is done as follows. For each capability Y which is required the predicate $reqCap(Y)$ is used to describe the capabilities and components which are needed by the robot.

$$\begin{aligned} reqCap(Y) \rightarrow & \bigwedge_{X \in capReqCap(Y)} reqCap(X) \wedge \\ & \bigwedge_{Z \in capReqComp(Y)} reqComp(Z) \end{aligned}$$

Through this equation, one can simply resolve the recursive dependencies on the capabilities. Some of these capabilities might need components. As several hard- or software instances can implement a specific component we use an equation for the required capabilities to resolve components. If a component W implements a required component Z we define the following constraint.

$$reqComp(Z) \rightarrow implComp(Z, W) \wedge comp(W)$$

Like capabilities, components can have dependencies. Thus, to model these dependencies we use another constraint.

$$comp(W) \rightarrow \bigwedge_{X \in compReqCap(W)} reqCap(X) \wedge \bigwedge_{z \in compReqComp(W)} reqComp(z)$$

Using the ontology, we can instantiate the constraints automatically. The instantiated constraints are gathered in the set $\mathcal{C}$. As we want to derive a minimal configuration for a given task X we first need to add $reqCap(X)$ to the set $\mathcal{C}$. Afterward, we need to find a minimal set of components $\mathcal{W}$ to fulfill this requirement. This is achieved by the following minimization problem.

$$\underset{\mathcal{W}}{\operatorname{argmin}} \left(|\{W | comp(W) \wedge W \in \mathcal{W}\}|\right) s.t.\mathcal{C}$$

The solution to this minimization problem is a minimal set of components to use to guaranty that all dependencies are met.

With the above-defined constraint problem, the minimal configuration can be generated. To realize these constraints in an efficient manner the constraint solving is split into two parts. The first part is the parsing of the ontology to extract the minimal dependency for one component. The second part uses this extracted data to find a minimal configuration in a very efficient way through a constraint solver.

The first part is done by extracting the minimal set of necessary capabilities, for a chosen task, by recursively traversing the referenced capabilities of the chosen tasks. Using the resulting set of capabilities all mandatory components are extracted. Additionally, during the recursion one creates a separate set of mandatory components for each alternative realization. After this extraction, the minimal set of necessary capabilities, as well as the mandatory components, are already determined. Therefore, only the extraction of the various combinations of alternative components, such that the required tasks still can be fulfilled, must be done. We solved this problem by using the constraint solver choco [5]. To represent the constraints, we generated a matrix $\mathcal{H}$. Each row in the matrix represents one abstract component descriptions. Each row vector $\mathcal{V}$ describes a component that can implement these abstract component descriptions. With the help of this representation the data can be modeled as follows:

- For all entries i of all vectors $\mathcal{V}$ in the matrix $\mathcal{H}$, a variable $\mathcal{E}(i)$ is generated
- The domains of these Variables $\mathcal{E}(i)$ are restricted, based on the component they implement. It is limited to the domain $\{0, \mathcal{B}(\mathcal{V})^{\mathcal{K}(\mathcal{H})+1}\}$, where $\mathcal{B}(\mathcal{V})$ denotes the maximum number of entries of all vectors $\mathcal{V}$ and $\mathcal{K}(\mathcal{H})$ is the index of the vector $\mathcal{V}$ within the matrix $\mathcal{H}$.

With this model all combinations of alternative components that are necessary to fulfill the given task can be determined using the following constraint:

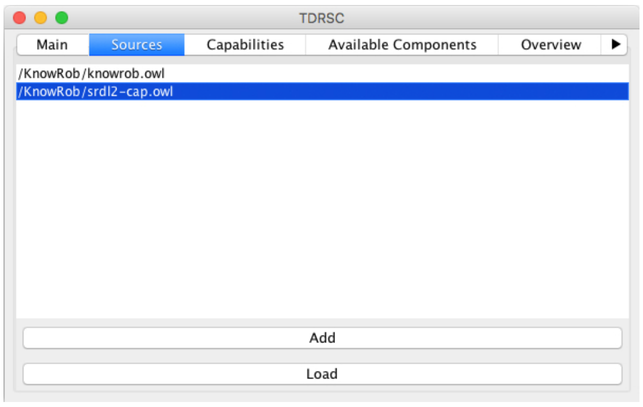
$$\sum_{i=0}^{\mathcal{N}(\mathcal{H})-1} \mathcal{E}(i) \overset{!}{=} \sum_{i=1}^{\mathcal{M}(\mathcal{H})} \mathcal{B}(\mathcal{V})^i$$

Here, $\mathcal{N}(\mathcal{H})$ denotes the total number of entries of all vectors $\mathcal{V}$ within $\mathcal{H}$ while $\mathcal{M}(\mathcal{H})$ denotes the number of rows within the matrix $\mathcal{H}$. This ensures that all values of $\mathcal{E}(i)$ are zero, except for a single entry between all entries $\mathcal{E}(i)$ that share the same domain. This single non-zero entry is equal to the chosen component among all component alternatives that implement the same main component. To retrieve the components represented by the values $\mathcal{E}(i)$ the index i is used as an index in an array containing the component names.

C. User Interface

In this section, the relevant components of our graphical user interface are described. The GUI itself is structured into separate tabs which all fulfill different tasks.

1) Defining Source Ontologies: The GUI features a tab that empowers the user to load any desired ontology (Figure 1). The definition of more than one ontology will result in a single model that contains all relationships. For this, the user is provided with a list that contains all ontologies added so far at the top of the tab. At the bottom, there are two buttons, one to add another ontology and another button to load the defined ontology files. Upon a click on the Load button, the ontologies will be loaded and scanned for capabilities, components and other important information. For this, the user may define keywords that identify components and capabilities, within the ontology, in the Settings tab. This generic approach should guarantee that the software can also incorporate different ontology sets.



/KnowRob/knowrob.owl
/KnowRob/srdl2-cap.owl

Fig. 1. The Sources tab of the GUI. Here the user may define the paths to the ontologies which are to be loaded into the model.

2) Defining Capabilities: After having loaded the desired ontologies, the user may define the desired tasks. There may be multiple of them or just one. This can be configured in the Capabilities tab as depicted in Figure 2. In this tab, desired tasks may be added using the Add button at the bottom. This will add a combo box with all the previously extracted capabilities, of which the desired one may be chosen. Additionally, the list of chosen capabilities may be saved to disk.

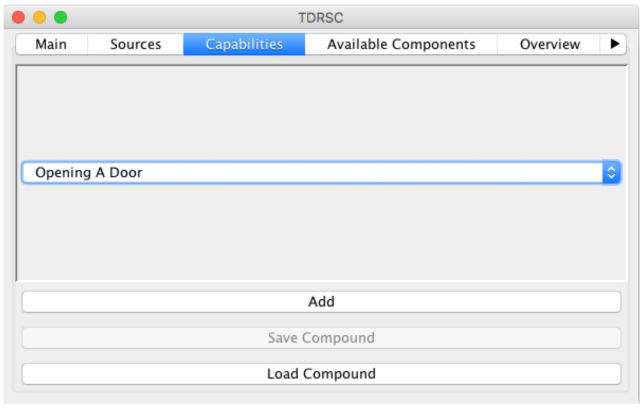
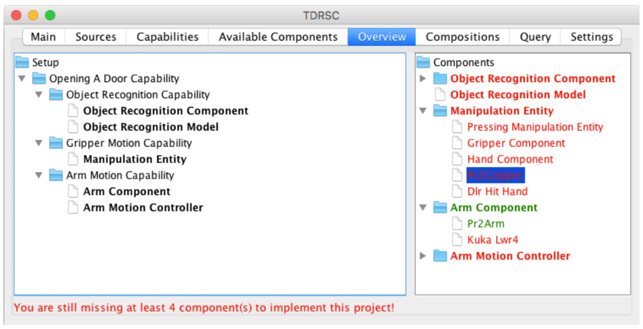


Fig. 2. The Capabilities tab of the GUI. Here the user may define tasks the robot should be capable of.

3) Defining Available Components: This tab is structured exactly like the capabilities tab. Here the available components (hardware as well as software) may be defined.

4) Feedback: Our program automatically analyses the situation anytime the user makes a change to the task requirements or available components. The result of this analysis is depicted in the Overview tab (Figure 3). It is divided into two sections including tree views. The left tree depicts the relationships between the chosen tasks and any subtask that describes parts of it. Also, it shows which components are necessary to implement these subtasks. On the right side, the user is provided with an overview of all components that need to be available to implement the desired tasks. In case the program could find several components that can be used to implement the same task, the component may be expanded and checked for the available options. In this view, missing components are depicted in red while available components (as defined within the Components tab) appear in green color (Figure 4).



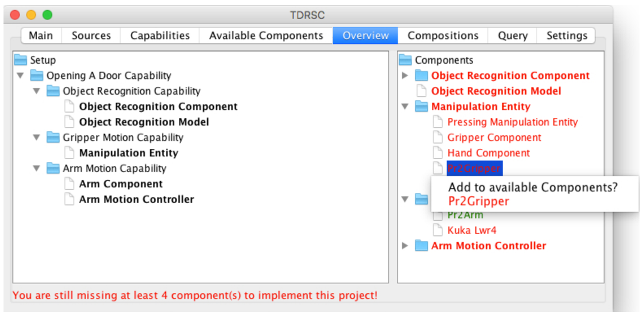
Setup
Opening A Door Capability
Object Recognition Capability
Object Recognition Component
Object Recognition Model
Gripper Motion Capability
Manipulation Entity
Arm Motion Capability
Arm Component
Arm Motion Controller

Components
Object Recognition Component
Object Recognition Model
Manipulation Entity
Pressing Manipulation Entity
Gripper Component
Hand Component
Pr2Gripper
Dlr Hit Hand
Arm Component
Pr2Arm
Kuka Lwr4
Arm Motion Controller

You are still missing at least 4 component(s) to implement this project!

Fig. 3. The Overview tab of the GUI. Here the user gets a feedback about the given situation.

5) Configuration Proposals: As the desired tasks, may be implemented with a wide variety of different constellations



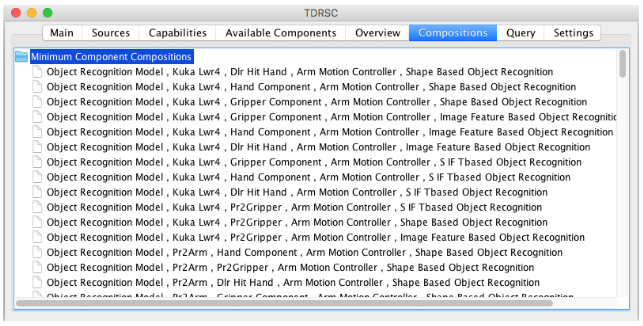
Setup
Opening A Door Capability
Object Recognition Capability
Object Recognition Component
Object Recognition Model
Gripper Motion Capability
Manipulation Entity
Arm Motion Capability
Arm Component
Arm Motion Controller

Components
Object Recognition Component
Object Recognition Model
Manipulation Entity
Pressing Manipulation Entity
Gripper Component
Hand Component
Pr2Gripper
Add to available Components?
Pr2Gripper
Pr2Arm
Kuka Lwr4
Arm Motion Controller

You are still missing at least 4 component(s) to implement this project!

Fig. 4. Adding available components from within the Overview tab, using a pop-up menu.

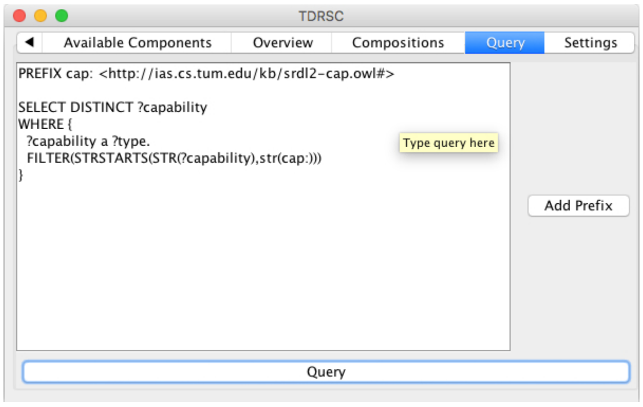
of components, the GUI also features a tab that suggests possible component setups that fulfill the desired tasks. This is depicted in Figure 5.



Minimum Component Compositions
Object Recognition Model , Kuka Lwr4 , Dlr Hit Hand , Arm Motion Controller , Shape Based Object Recognition
Object Recognition Model , Kuka Lwr4 , Hand Component , Arm Motion Controller , Shape Based Object Recognition
Object Recognition Model , Kuka Lwr4 , Gripper Component , Arm Motion Controller , Shape Based Object Recognition
Object Recognition Model , Kuka Lwr4 , Gripper Component , Arm Motion Controller , Image Feature Based Object Recognitic
Object Recognition Model , Kuka Lwr4 , Hand Component , Arm Motion Controller , Image Feature Based Object Recognition
Object Recognition Model , Kuka Lwr4 , Dlr Hit Hand , Arm Motion Controller , Image Feature Based Object Recognition
Object Recognition Model , Kuka Lwr4 , Gripper Component , Arm Motion Controller , S IF Tbased Object Recognition
Object Recognition Model , Kuka Lwr4 , Hand Component , Arm Motion Controller , S IF Tbased Object Recognition
Object Recognition Model , Kuka Lwr4 , Dlr Hit Hand , Arm Motion Controller , S IF Tbased Object Recognition
Object Recognition Model , Kuka Lwr4 , Pr2Gripper , Arm Motion Controller , S IF Tbased Object Recognition
Object Recognition Model , Kuka Lwr4 , Pr2Gripper , Arm Motion Controller , Shape Based Object Recognition
Object Recognition Model , Kuka Lwr4 , Pr2Gripper , Arm Motion Controller , Image Feature Based Object Recognition
Object Recognition Model , Pr2Arm , Hand Component , Arm Motion Controller , Shape Based Object Recognition
Object Recognition Model , Pr2Arm , Pr2Gripper , Arm Motion Controller , Shape Based Object Recognition
Object Recognition Model , Pr2Arm , Dlr Hit Hand , Arm Motion Controller , Shape Based Object Recognition

Fig. 5. The Compositions tab of the GUI. Here the user is provided with a list of all configuration that can solve the desired task.

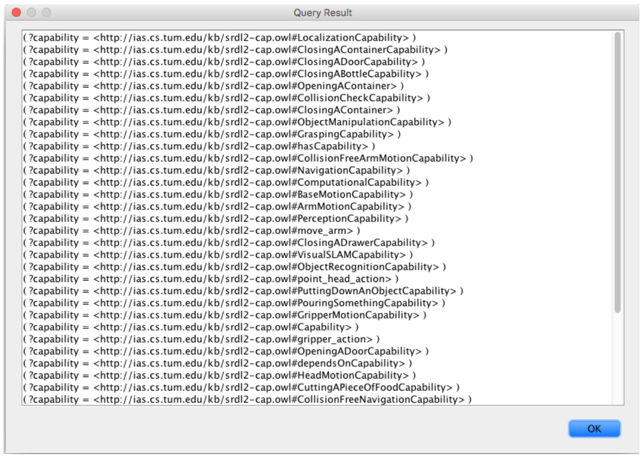
6) Manually Query Data: To manually query the loaded data, the GUI also features a Query tab (Figure 6). In this tab, the user may define custom queries on the loaded model. For convenience, the program extracts all available prefixes that can be added with just a few buttons clicks. The result of the query will be displayed in a separate pop-up window as depicted in Figure 7. 5.



```
PREFIX cap: <http://ias.cs.tum.edu/kb/srdl2-cap.owl#>

SELECT DISTINCT ?capability
WHERE {
 ?capability a ?type.
 FILTER(STRSTARTS(STR(?capability),str(cap:)))
}
```

Fig. 6. The Query tab of the GUI. The user may query the data manually using this tab and the SPARQL query language.



```
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#LocalizationCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#ClosingAContainerCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#ClosingADoorCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#ClosingABottleCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#OpeningAContainer> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#CollisionCheckCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#ClosingAContainer> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#ObjectManipulationCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#GraspingCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#hasCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#CollisionFreeArmMotionCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#NavigationCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#ComputationalCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#BaseMotionCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#ArmMotionCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#PerceptionCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#move_arm> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#ClosingADrawerCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#VisualSLAMCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#ObjectRecognitionCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#point_head_action> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#PuttingDownAnObjectCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#PouringSomethingCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#GripperMotionCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#Capability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#gripper_action> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#OpeningADoorCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#dependsOnCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#HeadMotionCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#CuttingAPieceOfFoodCapability> )
( ?capability = <http://ias.cs.tum.edu/kb/srdl2-cap.owl#CollisionFreeNavigationCapability> )
```

Fig. 7. The result of a user defined query is depicted in this pop-up window.

III. Example Scenario

Before we discuss the related research, we will discuss a simple running example, showing the different steps of a configuration. If one wants to know the minimal set of required components, necessary to implement a robot that can open a door one can use our program through performing the following steps:

- In order to identify the necessary components (hardware as well as software) using our program, first the ontology stated in section II-A, using the Source tab as depicted in Figure 1 must be loaded. After the loading process is done, the program has identified the capabilities and components, defined in the given ontologies. For the example, we use the ontology of [6] which contains all necessary capabilities.
- Now the capability ”Open a door“ should be available for selection in the Capabilities tab. It can be selected, by using the Add button at the bottom of the tab and selecting it in the newly created combo box as depicted in Figure 2.
- The program analyses the given situation online. Therefore, now, the user can already check for the necessary components to achieve the task in the Overview tab. If one has already components in mind which should be used, one may define them in the Available Components Tab. Assumed there is a ”Pr2Arm“ component that should be used. One can add these components before or after checking the necessary components.
- Now the user may want to check the necessary components to implement this task. If the ”Pr2Arm“ was added in advance the output of the Overview tab will be as depicted in Figure 3. There is also a possibility to add available components directly from within this overview. For this one may simply right-click the desired component and click the pop-up menu (Figure 4)
- Alternatively one may also check all possible constellations to implement this task by looking at the Compositions tab as depicted in Figure 5.

Optionally, custom SPARQL queries on the loaded model may be performed using the query tab. An example query to retrieve all available capabilities of the loaded model is depicted in figure 6. The result (all available capabilities) is shown in a pop-up as shown in Figure 7.

IV. Limitations of the Approach

The approach presented allows an easy specification of the robotics requirements and its resulting configuration. Additionally, one can generate a minimal configuration for the robot. These calculations are based on an ontology which describes the necessary dependencies to perform a task. Due to this specification, one may encounter several problems.

First, the ontology used in the example specifies the requirement for a home like an environment. The requirements may differ in a factory environment or on a planetary mission. To cope with this problem one could argument the requirements with a specification which environment the robot is operating in. Thus, one could add the information of the environment to the ontology to derive the proper set of requirements.

Another important limitation is the abstraction of the ontology. Let’s consider the example which specifies that one needs a robotic arm to fulfill the task. Thus, one can choose an arbitrary arm which may not be possible in practice as the arm does not allow to create enough force to perform the task or is too heavy to be placed on the robot. To tackle this problem one need to add additional constraints which need to be considered like the force which needs to be applied, maximum weight, ... Such constraints may not be simply integrated into the ontology reasoning. Instead one may add an additional layer of reasoning to check these constraints. Thus, one could find a configuration per the ontology and afterward check the additional constraints to rule out not applicable configurations.

V. Related Research

We start our discussion of related research with the semantic robot description language (SRDL) published in [7]. The description language allows describing the capabilities of the robot as well as the hardware and software components. Furthermore, dependencies of the capabilities and the components can be described. This description allows the robot to check if the dependencies are met for a specific capability. Additionally, the robot can enumerate all components which are missing for a specific capability. Thus, the first step for an automatic configuration of the robot is possible. To use this description in a robotic system SRDL was integrated into a general knowledge base for a robot through KnowRob [8]. This integration was used in the RoboEarth language [6] to allow an easy transfer of action recipes to perform a task. With the help of the SRDL, the robot could check if a certain action recipe to perform a task can be used. In this paper, we used SRDL as a basis for our tool to allow the derivation of a minimal configuration. Thus, instead of just checking if a robot can perform a capability our tool also allows getting a minimal configuration such that the robot can perform a capability.

Another method which uses an ontology to describe the environment was presented in [9]. The method uses a description of the environment which is based on an ontology together with a description of skills the robot can perform. Using this description, the robot can plan a sequence of skills which need to be executed to perform a certain task. Such a task was presented in [10] where the robot had to place parts of an industrial kitting operation.

To plan which robot can perform which task in a heterogeneous group of robots the method outlined in [11] can be used. The method defines capabilities which have preconditions which need to be met to allow the execution as well as information which need to be provided by the robotic system to allow the capability to be executed. Thus, the robot can plan which capabilities need to be executed to perform a task. Furthermore, the robot can use the hardware description to check if such an execution is possible. As many capabilities, e.g. grasp an object, may only work under certain restrictions, e.g. size of the object, one can add an approximation description to each capability which defines which conditions need to hold approximately to allow the execution of the capability. This allows to define capabilities in more detail and thus allow a better distribution of tasks among the robotic group. The method outlined in [11] focus on distributing tasks in a group of robots whereas the tool we present in this paper focuses on the configuration of the robotic system during the design phase. Thus, instead of planning which capabilities to use to fulfill a task, we show which different minimal configurations of the robotic system allow the robot to execute the capability. This allows the developer of the robotic system to choose the best fitting set of capabilities.

The method presented in [12] extends AutomationML [13] to allow the modeling of a robotic system. This is done by extending the given concepts with robotic specific concepts such as actuators or sensors. Furthermore, the method allows an automatic conversion of AutomationML specifications into an ontology which can be used to check for consistency. This can be used to model a robotic system with AutomationML and afterward check through the transformation to an ontology if every dependency is met or if a component is used which does not solve any given task. Beside these checks, further checks can be applied to verify that this system can be realized with the help of ROS [14]. This allows the developer to ensure that the modeled robot can be realized. After these checks are performed one can apply a model-to-text transformation to generate code stubs which ensure proper communication and operation per the modeled system. This allows a faster creation of a robotic system following a model-based approach like the method outlined in [2]. In contrast to our approach, the method does not offer the possibility to check which components are necessary to perform a capability. Thus, the method presented in [12] is also not able to specify a minimal configuration which fulfills the need for a specific capability as our tool can.

Besides the description of tasks for configuration, such a description is also often used to assign a task, e.g. in a multi-robot system. One such example is which uses an ontology to assign tasks is presented in [15]. The method uses an ontology per robot to describe which roles can be performed and how these roles are performed. Additionally, tasks are described in the ontology and how they can be executed through a role which is assigned to a robot. The system uses this ontology to find a matching robot and assigns different roles for different robots to fulfill the specified task.

VI. Conclusion and Future Work

The proper configuration of a robotic system for a given task is a time consuming and tedious task. Especially one needs an expert to perform this task to consider all possibilities as well as all dependencies. To address this problem, in this paper we presented a tool for the automatic configuration of a robotic system for a given task. The tool uses an ontology-based knowledge base, allowing to reuse publicly available knowledge bases, to describe which dependencies, exist between a task and software and hardware components. Furthermore, we have presented a method to derive a minimal set of software and hardware components to fulfill a certain task. This allows the user to simply find a possible configuration of the robotic system, that allows the robot to fulfill its task. To allow an easy interaction the tool has a graphical user interface which allows the user to select tasks as well as used components. Thus, the user can specify the currently used components on the robot to check if a new task can be achieved by the robot, or which components need to be added to allow the robot to achieve a given task.

Currently, the tool can only be used by a human to decide which components to use to allow the execution of a specific task. It is left for future work to allow the robot itself to use the tool. This would open the possibility that the robot finds alternative solutions to a task during runtime. Thus, the robot could reconfigure itself to react to a fault or changes in its task. Furthermore, currently only a minimal number of components is searched for the configuration, neither computation costs nor investment or development costs are considered in the configuration. It is left for future work to integrate these costs to allow to find a configuration which minimizes the computation effort or to minimize the investment costs.

References

[1] Z. Zhang, “Microsoft kinect sensor and its effect,” *IEEE multimedia*, vol. 19, no. 2, pp. 4–10, 2012.

[2] G. ”Steinbauer and C. Mühlbacher, “Hands off - a holistic model-based approach for long-term autonomy,” in *Workshop on AI for Long-Term Autonomy, 2016 IEEE International Conference on Robotics and Automation (ICRA)*.

[3] Knowrob.org. Capability ontology - knowrob. [Online]. Available: http://knowrob.org/kb/srdl2-cap.owl

[4] Apache.org. Jena framework - apache. [Online]. Available: https://jena.apache.org/

[5] choco solver.org, “choco-solver.” [Online]. Available: http://www.choco-solver.org/

[6] M. Tenorth, A. C. Perzylo, R. Lafrenz, and M. Beetz, "The roboearth language: Representing and exchanging knowledge about actions, objects, and environments," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1284–1289.

[7] L. Kunze, T. Roehm, and M. Beetz, "Towards semantic robot description languages," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 5589–5595.

[8] M. Tenorth and M. Beetz, "Knowrob: A knowledge processing infrastructure for cognition-enabled robots," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 566–590, 2013.

[9] F. Rovida and V. Krüger, "Design and development of a software architecture for autonomous mobile manipulators in industrial environments," in *Industrial Technology (ICIT), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3288–3295.

[10] A. S. Polydoros, B. Großmann, F. Rovida, L. Nalpantidis, and V. Krüger, "Accurate and versatile automation of industrial kitting operations with skiros," in *Conference Towards Autonomous Robotic Systems*. Springer, 2016, pp. 255–268.

[11] J. E. Buehler, "Capabilities in heterogeneous multi robot systems." in *Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI) Doctoral Consoritum*. AAAI, 2012, pp. 2380–2381.

[12] Y. Hua, S. Zander, M. Bordignon, and B. Hein, "From automationml to ros: A model-driven approach for software engineering of industrial robotics using ontological reasoning," in *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*. IEEE, 2016, pp. 1–8.

[13] R. Drath, A. Luder, J. Peschke, and L. Hundt, "Automationml-the glue for seamless automation engineering," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. IEEE, 2008, pp. 616–623.

[14] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[15] F. Amigoni and M. A. Neri, "An application of ontology technologies to robotic agents," in *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*. IEEE, 2005, pp. 751–754.