Towards a Robot Perception Specification Language

Nico Hochgeschwender, Sven Schneider, Holger Voos, and Gerhard K. Kraetzschmar

I. Introduction

Domestic service robots such as PR2 [1] and Care-O-bot 3[1] must be able to perform a wide range of different tasks ranging from opening doors [1] and making pancakes [2] to serving drinks [3]. A crucial precondition to achieve such complex tasks is the ability to extract *task knowledge* about the world from the data perceived through the robot's sensors. Examples are the localization of humans [4] for navigation and interaction purposes, or the detection and recognition of objects in grayscale images [5] for the sake of manipulation by the robot. To perceive all the knowledge needed to safely and robustly perform a task, robots are equipped with a set of heterogenous sensors such as laser range finders, ToF cameras, structured light cameras and tactile sensors which provide different types of data such as distance measurements, depth images, $3D$ point clouds, and $2D$ grayscale or color images. To structure all the required processing steps on this data so called *Robot Perception Architectures* (RPAs) are required (see also Figure 1). In general, RPAs are composed of functional components processing sensory input to output which is relevant for the task in hand. Thereby, heterogenous algorithms such as filters and feature detectors are integrated in components which are then assembled to make up an RPA [6]. However, despite recent algorithmic advancements in the field of vision and perception, the development of RPAs, designed to extract meaning out of the enormous amount of data, is still a complex and challenging exercise. There is little consensus on either how such an architecture is best designed for any particular task or on how to organize and structure robot perception architectures in general, so that they can accommodate the requirements for a *wide range* of tasks. In this paper we present our work in progress towards a *Robot Perception Specification Language (RPSL)*. *RPSL* is a domain-specific language and its purpose is twofold. First, to provide means to specify the expected result (*task knowledge*) of a RPA in an explicit manner. Second, to initiate the (re)-configuration process of an RPA based on the provided specification. Here, we focus on the first objective and discuss the core language concepts which have been composed in *RPSL*.

Nico Hochgeschwender, Sven Schneider, and Gerhard Kraetzschmar are with the Department of Computer Science, Bonn-Rhein-Sieg University of Applied Sciences, Germany. Email: forename.surname@h-brs.de Nico Hochgeschwender and Holger Voos are with the Research Unit in Engineering Sciences, University of Luxembourg, Luxembourg. Email: holger.voos@uni.lu

[1] http://www.care-o-bot-research.org/

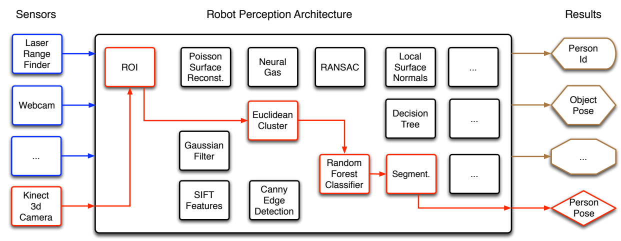

Fig. 1. Elements making up the design space of a robot perception architecture: i) heterogenous sets of sensors (blue boxes), ii) computational components (black boxes), and iii) task-relevant information and knowledge (brown boxes). The path which is visualized in red shows an instance of an existing RPA described in Hegger *et al.* [4]. *RPSL* is used to specify the task knowledge visualized in brown boxes on the left hand side.

II. Problem Statement and Motivation

Currently, robot perception architectures are developed by domain experts during design time. The design is significantly influenced by many decisions, which often remain implicit. These design decision concern the robot *platform*, the *tasks* the robot should perform, and the *environment* in which the robot operates. Some exemplary design decisions include:

- The sensor configuration (e.g., resolution or data frequency) of a particular sensor according to environment and task specifications.
- The general composition of an RPA, including the selection, configuration and organization of computational components (implementing the core sensor processing functionalities such as filters, classifiers, etc.) such that task and environment requirements are met.
- The configuration of a specific composition of an RPA for solving a particular task-relevant perception problem, e.g. determining the pose of a human. The pertains to particular dynamic connection of RPA components.

As long as task, environment, and platform specifications remain as assumed during design time, the RPA will operate properly. However, if an event concerning robot capabilities, task requirements, or environment features occurs, systematically ensuring an appropriate reaction by the RPA is a great challenge. Generally speaking, the vast majority of RPAs is static and inflexible and it is not possible

- to reconfigure parameters of computational components (e.g., the σ value of a Gaussian filter) during run time,
- to execute complete processing chains in a demand-driven manner,
- and to modify and reconfigure robot perception processing chains during run time.

To provide RPAs with the ability to reconfigure their structure and behavior one needs to model the design decisions mentioned above in an explicit and computable (during runtime available) manner. First of all the desired *task knowledge* needs to be specified. Depending on the functional component (e.g., manipulation, grasping, or decision-making) which requires the knowledge and the current task at hand this knowledge differs substantially. For instance, a decision-making component might be interested in the existence of an object whereas a grasping and manipulation component demands more sophisticated information such as spatial dimensions and shapes of an object. In both cases means to express the desired task knowledge are required. To the best of our knowledge in robotics there is no language available which allows us to encode such specifications. We observed that very often ad-hoc solutions e.g., in the form of message definitions (provided by the underlying robot software framework) are used which lack expressiveness.

III. RPSL: Robot Perception Specification Language

In the following we present the current status of *RPSL*. We identify first language requirements and then describe the different domain concepts which are part of the language. Those concepts have been identified through a domain analysis of existing RPAs and their application context in real-world scenarios.

A. Requirements and Assumptions

RPSL is aimed to be a specification language. Therefore, the language is not executable. Interestingly, from a planning point of view the specifications are comparable with goal specifications in the Planning Domain and Definition Language (PDDL) [7]. Similarly to PDDL a specification language for the perception domain should be independent of the underlying RPA just as PDDL is independent of concrete task planners. To be usable for a wide range of applications and systems, *RPSL* should be independent of

- the type of sensor data processed by the RPA, and
- the type of functional components which are assembled to make up an RPA.

To enable reuse and exchangeability of the domain concepts realized in *RPSL* (e.g., through concrete language primitives and abstraction) they should be orthogonal to each other as far as possible. Further, we assume that an environment is not actively observed (e.g., no active perception which involves movements of the robot). However, many so called table top situations in robotics are covered with our current status of *RPSL*.

B. General Approach

Based on our domain analysis we derived several core domain concepts described in the following. To model the domains we apply a model-driven engineering approach using the Eclipse Modeling Framework (EMF)[2]. Here, each domain is specified in the form of an Ecore model. Based on the Ecore models we developed an external domain-specific language (DSL) with XText[3]. As *RPSL* is work in progress we use the external DSL mainly to validate the domain concepts with experts. The next sections describe the domains and features that need to be captured by *RPSL* in more detail.

```
myConcepts: Namespace {
  myBox: Concept {
    use_domain Size
    p: Polytope {
      Point(Size.Height, 20mm)
      Point(Size.Height, 40mm)
      Point(Size.Width, 20mm)
      Point(Size.Width, 40mm)
      Point(Size.Length, 100mm)
    }
  }
}
```

Fig. 2. Concept definition of a box.

C. Object Domain

As exemplified in Figure 1 and discussed in Section II there is a huge variability in the kind of *task knowledge* potentially provided by RPAs. Ranging from diverse objects such as persons and objects of daily use such as cups, bottles, and door handles to the information about these objects themself such as center of mass, poses, color and shapes. Here, the challenge is to use a representation which enables us to model the information about objects on various levels of abstraction. In *RPSL* the object domain is based on Conceptual Spaces (CS) which is a knowledge representation mechanism introduced by Gärdenfors [8]. A conceptual space is composed of several (measurable) quality dimensions. A concept in a conceptual space is a convex region in that space. Points (also called knoxels) in a conceptual space represent concrete instances (objects) of a concept. To decide whether an instance belongs to one concept or to another we can apply similarity measures such as Euclidean distances. In Figure 2 an example is shown. Here, a `Concept` called *myBox* is specified. The concept belongs to the `Namespace` *myConcepts* which is simply a mechanism to organize different concepts as known in general-purpose programming languages such as Java or C++. The concept *myBox* uses the `Domain` *Size* which is composed of three quality dimensions, namely *Height*, *Width*, and *Length*. In *RPSL* quality dimensions with different scales such as continuous or ordinal scales are supported. A `Polytope` is further used to model the "borders" of the concept *myBox*. For instance, every box belonging to the concept *myBox* needs to have a height between $20mm$ and $40mm$. In contrast to the Conceptual Space Markup Language (CSML) introduced by Adams and Raubal [9] we use polytopes instead of a set of inequalities to define the concept region as they are easier to model. To enrich the concept *myBox* we simply refer to

[2]http://www.eclipse.org/modeling/emf/

[3]http://www.eclipse.org/Xtext/

```
myConcepts: Namespace {
  myBox: Concept {
    use_domain Size
    use_domain RGB
    p: Polytope {
      // ...
      Point(RGB.Red, 0)
      Point(RGB.Green, 0)
      Point(RGB.Blue, 100)
      Point(RGB.Blue, 130)
    }
  }
}
```

Fig. 3. Concept definition of a box with color information.

```
use Namespace myConcepts

darkBlueBox: Prototype {
  use_concept myConcepts.myBox
  v: Values {
    // ...
    Point(myBox.RGB.Blue, 139)
  }
}
```

Fig. 4. Prototype definition of a dark blue box.

another domain. For instance, in Figure 3 the concept *myBox* is enriched with color information using the RGB color coding which includes three quality dimensions, namely, *Red*, *Green*, and *Blue*. This approach allows us to model very expressive concepts as we can reuse existing domains and corresponding quality dimensions. Once concepts are defined we can model concrete instances or speaking in the conceptual space terminology: "prototypes". In Figure 4 a `Prototype` *darkBlueBox* is modeled. Instead of defining ranges as in the concept definition, prototypes have single values per quality dimension.

D. Spatial Domain

Very often it makes sense to specify the required object information with respect to the spatial surrounding. Assuming an egocentric view of the robot one could model for instance objects through spatial operators such as "behind", "next to", and "right of". In particular, for manipulation tasks it is crucial to have information not only about the object to manipulate, but also about their spatial surrounding in order to plan motions and to check for collisions. Currently, we investigate which spatial model we want to include in *RPSL* such as the region connection calculus (RCC) [10].

E. Timing Domain

With the timing domain we intend to enrich specifications about the *"when"*. More precisely, in many situations it is important to retrieve information about objects within a certain time frame e.g., to avoid a stucking robot behavior. We use the notion of a deadline to encode a particular point in time by which the specified information should be available. For instance, specification *s5* shown in Figure 5 is enriched with a `Deadline` of $3s$. Here, `Deadline` can be parameterized with the value and an time unit. From an implementation point of view once the specification is received by the RPA it will obtain a time stamp which will be used to cope with the deadline. This imposes a certain protocol between the component which emits the specification and the RPA which will not be discussed here. In future we intend to extend the timing domain with policies allowing to model strategies with missed deadlines (e.g., "when deadline X is missed try to retrieve information Y or repeat it once").

F. Dependencies

Another feature in *RPSL* is to model dependencies among specifications. That is some information is required before some other information is available. In Figure 5 specification *s4* is composed of two specifications which have a dependency. First, the amount of the *darkBlueBox* is retrieved and then the *Pose* of the *darkBlueBox* is retrieved. To model these situations the dependency meta-model is based on the concept of a directed acyclic graph (DAG). Interestingly, in the past we used the same dependency meta-model to model the sequence of component deployment [11] and robot action plots [12].

G. Composition Domain

The composition domain composes the previous domains in order to model a valid and complete specification. Some concepts such as timing and dependencies are optional whereas the object domain is mandatory. In Figure 5 some examples are shown. First, the `Namespace` *myConcepts* is used. Further, in the first specification one is interested in the amount of objects (visible in the current scene) belonging to the concept *myBox* with certain properties concerning length and width. Here, *Amount* itself is a concept with one quality dimension, namely an ordinal integer scale. As seen in the example the syntax is inspired by SQL with the difference that the data model is based on Conceptual Spaces. Similarly to SQL we support logical operators such as `AND` and `OR` as well as relational operators such as $==$, $>$ and $<=$ known from general-purpose programming languages. In the second specification *s2* the previously modeled prototype *darkBlueBox* is used. After the `where` statement a condition is modeled. Here, the condition is that only objects which look exactly like the *darkBlueBox* (similarity measured with Euclidean distance) are counted. The idea is that with the `Similarity` operator several similarity measures are supported and that we can balance the expected result according to the features provided by the measure. In future we intend to support also weighting factors which can be applied to increase or decrease the importance of quality dimensions for the similarity measure.

IV. Conclusion

We presented the work in progress of using domain-specific languages for specifying robot perception architectures. Assessing the DSLRob workshop series showed

```
use Namespace myConcepts

s1: Specification {
  d: Data {
    get Amount from myBox where myBox.Size.Width <= 20mm and myBox.Size.Length > 100mm
  }
}

s2: Specification {
  d: Data {
    get Amount from darkBlueBox where Similarity(EuclideanDistance) == 0
  }

s3: Specification {
  d: Data {
    get Pose from darkBlueBox where Similarity(EuclideanDistance) == 0
  }
}

s4: Specification {
  dg: DependencyGraph {
    s2 before s3
  }
}

s5: Specification {
  d: Data {
    get Amount from darkBlueBox where Similarity(EuclideanDistance) == 0 ensure Deadline(3s)
  }
}

}
```

Fig. 5. Some example specifications.

that *RPSL* is the first attempt to use DSLs in the sub-domain of robot perception. Even though, *RPSL* is work in progress it helped already to identify and break down the crucial domains which are involved in specifying the result of RPAs. To achieve the second objective of our language, namely the initialization of a (re)-configuration based on the specification we are currently implementing a use case which is based on simple table top scene.

Acknowledgement

Nico Hochgeschwender received a PhD scholarship from the Graduate Institute of the Bonn-Rhein-Sieg University of Applied Sciences which is gratefully acknowledged. Furthermore, the authors gratefully acknowledge the on-going support of the Bonn-Aachen International Center for Information Technology.

References

[1] W. Meeussen, M. Wise, S. Glaser, S. Chitta, C. McGann, P. Mihelich, E. Marder-Eppstein, M. Muja, V. Eruhimov, T. Foote, J. Hsu, R. B. Rusu, B. Marthi, G. Bradski, K. Konolige, B. P. Gerkey, and E. Berger, "Autonomous door opening and plugging in with a personal robot," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[2] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mösenlechner, D. Pangercic, T. Rühr, and M. Tenorth, "Robotic roommates making pancakes," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, 2011.

[3] T. Breuer, G. Giorgana Macedo, R. Hartanto, N. Hochgeschwender, D. Holz, F. Hegger, Z. Jin, C. Mueller, J. Paulus, M. Reckhaus, J. Alvarez Ruiz, P. Ploeger, and G. Kraetzschmar, "Johnny: An autonomous service robot for domestic environments," *Journal of Intelligent and Robotic Systems, Special Issue on Domestic Service Robots in the Real World.*

[4] P. G. P. Frederik Hegger, Nico Hochgeschwender and G. K. Kraetzschmar, "People Detection in 3d Point Clouds using Local Surface Normals," in *Proceedings of the 16th RoboCup International Symposium*, ser. Lecture Notes in Computer Science. Mexico City, Mexico: Springer, June 2012, to appear.

[5] T. Asfour, T. Azad, N. Vahrenkamp, K. Regenstein, A. Bierbaum, K. Welke, J. Schrder, and R. Dillmann, "Toward humanoid manipulation in human-centred environments," *Robotics and Autonomous Systems*, vol. 56, pp. 54–65, 2008.

[6] G. Biggs, N. Ando, and T. Kotoku, "Rapid data processing pipeline development using openrtm-aist," in *System Integration (SII), 2011 IEEE/SICE International Symposium on*, 2011, pp. 312–317.

[7] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl - the planning domain definition language," Yale Center for Computational Vision and Control,, Tech. Rep. TR-98-003, 1998.

[8] P. Gärdenfors, *Conceptual spaces - the geometry of thought*. MIT Press, 2000.

[9] B. Adams and M. Raubal, "Conceptual space markup language (csml): Towards the cognitive semantic web." in *ICSC*. IEEE Computer Society, 2009, pp. 253–260.

[10] D. A. Randell, Z. Cui, and A. Cohn, "A Spatial Logic Based on Regions and Connection," in *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, B. Nebel, C. Rich, and W. Swartout, Eds. San Mateo, California: Morgan Kaufmann, 1992, pp. 165–176.

[11] N. Hochgeschwender, L. Gherardi, A. Shakhimardanov, G. Kraetzschmar, D. Brugali, and H. Bruyninckx, "A model-based approach to software deployment in robotics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*, 2013.

[12] M. Reckhaus, N. Hochgeschwender, P.-G. Plöger, and G. K. Kraetzschmar, "A platform-independent programming environment for robot control," in *Proceedings of the 1st International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob) held at the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.