2017 First IEEE International Conference on Robotic Computing

Managing the functional variability of robotic perception systems

Davide Brugali
Department of Computer Engineering
University of Bergamo
24044 Dalmine, Italy
Email: brugali@unibg.it

Nico Hochgeschwender
Department of Computer Science
Bonn-Rhein-Sieg University
53757 Sankt Augustin, Germany
Email: nico.hochgeschwender@h-brs.de

***Abstract*—Control systems for autonomous robots are concurrent, distributed, embedded, real-time and data intensive software systems. A real-world robot control system is composed of tens of software components. For each component providing robotic functionality, tens of different implementations may be available.**

The difficult challenge in robotic system engineering consists in selecting a coherent set of components, which provide the functionality required by the application requirements, taking into account their mutual dependencies. This challenge is exacerbated by the fact that robotics system integrators and application developers are usually not specifically trained in software engineering.

Current approaches to variability management in complex software systems consists in explicitly modeling variation points and variants in software architectures in terms of Feature Models.

The novel contribution of this paper is the description of the integration of two modeling languages and toolkit, namely HyperFlex [14] for functional variability modeling and the Robot Perception Specification Language (RPSL) [17], a Domain-specific Language (DSL) enabling domain experts to express the architectural variability of robot perception systems.

I. Introduction

Robot control systems are typically designed as (logically) distributed component-based systems (see [9] for a survey). A real-world robot control system is composed of tens of software components. For each component providing a robot functionality, tens of different implementations may be available. The initial release of the Robot Operating System (ROS) [1] in year 2010 already contained hundreds of open source packages (collections of nodes) stored in 15 repositories around the world.

Clearly, building complex control applications is a matter of system integration more than of capabilities implementation. The difficult challenge consists in selecting a coherent set of components that provide the required functionality taking into account their mutual dependencies.

System configuration is a crucial phase, which requires to select, integrate, and fine tune the robot functionalities (developed by domain experts) according to the available resources (requiring maintenance by qualified engineers), the environment conditions (often beyond the control of the application engineer), and the task to be performed (often specified by unskilled users).

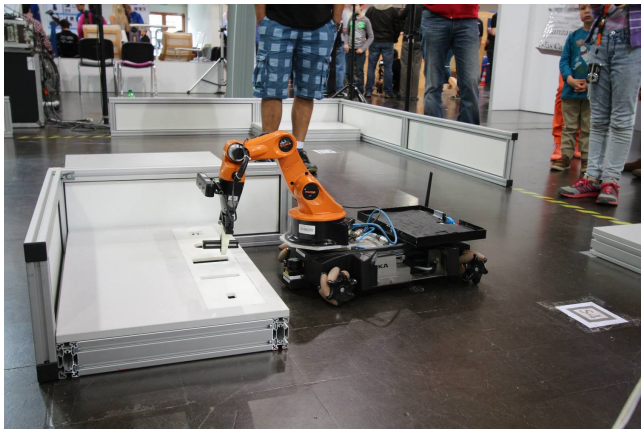

Fig. 1. A youBot mobile manipulation robot performing a precision placement task.

In previous papers [14], [10] we have presented the HyperFlex Model-driven toolchain and approach for the design of software product lines for autonomous robotic systems.

The key characteristics of HyperFlex are the support to the design and composition of architectural models of component-based functional subsystems, the possibility to symbolically represent the variability of individual functional subsystems using the Feature Models formalism [22], and the automatic configuration of functional subsystems according to selected features. The HyperFlex approach builds on our experience in developing software architectures for robotic control systems in the context of the EU FP7 BRICS project [6].

The novel contribution of this paper is the description of the integration of HyperFlex with the Robot Perception Specification Language (RPSL) [17], a Domain-specific Language (DSL) enabling domain experts to express the architectural and functional variability of robot perception systems.

The integration has two main benefits. At one side, it specializes the HyperFlex toolkit with a domain specific architectual modeling language and demonstrates the flexibility of the overall approach. At the other side, it enables the easy composition of perception subsystems, modeld with the RPSL framework, with other functional subsystems, such as manipulation, by explicitly modeling functional dependencies.

978-1-5090-6724-4/17 $31.00 © 2017 IEEE
DOI 10.1109/IRC.2017.20

IEEE computer society

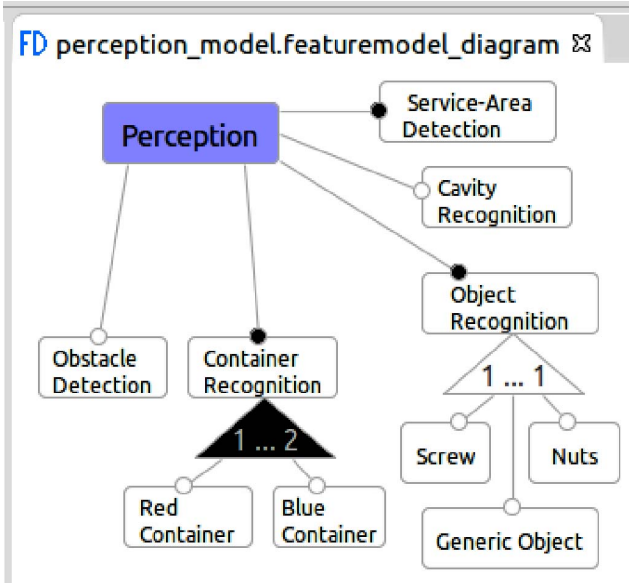

Fig. 2. Perception capabilities of the RoboCup@Work scenario.

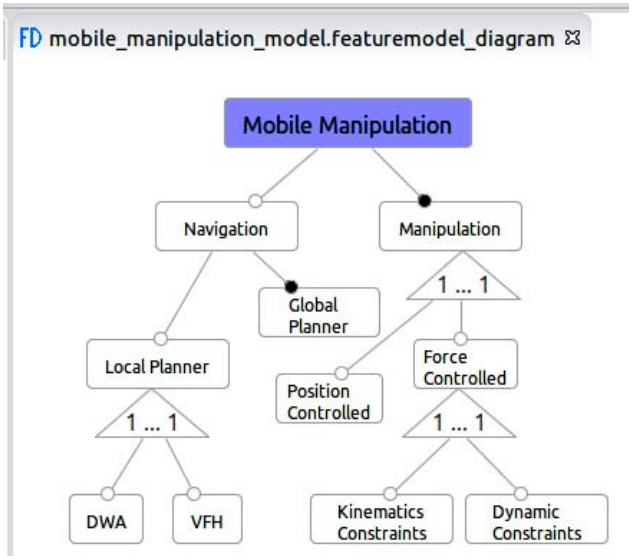

Fig. 3. Manipulation and navigation capabilities of the RoboCup@Work scenario.

The paper is structured as follows. Section II illustrates the case study used to illustrate the proposed approach. Section III presents the modeling languages supported by the HyperFlex toolkit and exemplifies their use in the context of the RoboCup@Work scenarios. Section IV presents the RPSL approach, the integration with HyperFlex, and the architectural modeling of the perception capabilities required in the RoboCup@Work scenarios. Section V reports on the related works. The relevant conclusions are presented in Section VI.

II. Case Study

In this section we motivate our approach with the help of a case study. The case study has been developed in the context of two recent scientific robot competitions, namely RoboCup@Work [18] and RoCKIn [4].

In those competitions mobile manipulation robots are expected to perform a wide range of manipulation, assemby and logistic tasks in factory-like environments. In our case study, a youBot mobile manipulation robot (see Fig. 1) is deployed in an environment which is composed of service areas. Here, each service area represents a region of the factory having a specific purpose for a particular task. For example, areas to pick objects, to insert objects into object-specific cavities (see Fig. 1), to place objects into containers, to operate machines and so forth. Those service areas differ also in terms of height, width and so forth.

Additionally, some environments includes static obstacles whereas others are free of obstacles or even include dynamic obstacles such as other robots and human workers. In the context of this paper we focus on three possible manipulation tasks, namely a simple table-top pick, placement and precision placement of a set of predefined objects in object-specific cavities.

Developing and configuring the robot software for such an application is a challenging exercise. All the task and environment requirements need to be considered in the selection and configuration of crucial robot capabilities such as manipulation, navigation and perception. For example, simple placement of an object on a service area requires only a standard plane and free space detection algorithm whereas for detecting the object-specific cavities more elaborated and possibly object-specific algorithms are required.

III. Variability Composition and Abstraction with HyperFlex

HyperFlex [14] is a Model-driven engineering (MDE) toolchain [7] that supports the development of flexible and configurable robotic control systems. It consists in a set of Eclipse plugins for the definition and manipulation of three types of software models, which are completely orthogonal, i.e. they can vary independently:

- *Architectural Models* represent the structure of control systems in terms of component interfaces, component implementations, and component connectors. The HyperFlex approach promotes the design and composition of domain-specific software architectures for common robotic functionality (e.g. robot navigation), which capture the variability in robotic technologies (e.g. various algorithms for trajectory generation).
- *Feature Models* symbolically represent the variant features [22] of a control system; symbols may indicate individual robot functionality (e.g. marker-based localization) or concepts that are relevant in the application domain, such as the type of items that the robot has to transport (e.g. liquid, fragile, etc.), which affect the configuration of the control system.

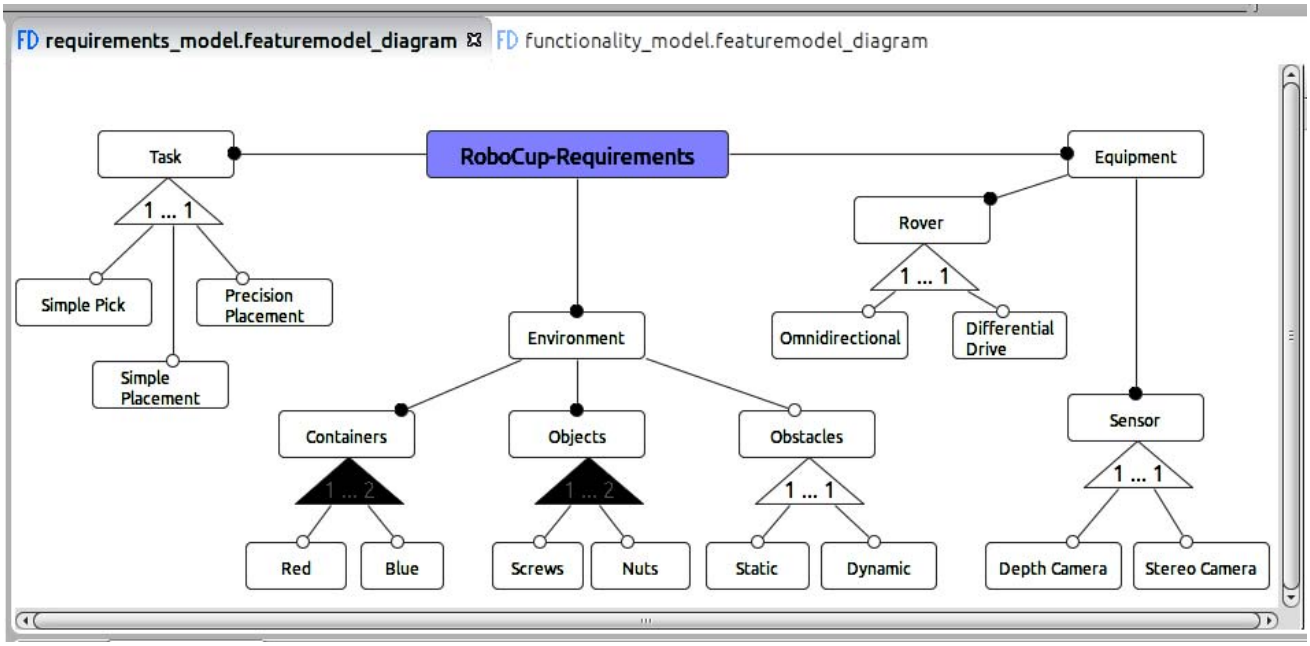

Fig. 4. Requirements of the RoboCup@Work scenario

- *Resolution Models* define model-to-model transformations, which allow to automatically configuring the architecture and functionality of a control system based on selected features. Eventually, the configured architectural model is used to deploy the control system on a specific robotic platform.

An interesting challenge that needs to be faced when using feature models to represent the variability of a software product line is the definition of an appropriate vocabulary for naming variation points and variants. The clear separation of the symbolic representation of the system variability from its architectural model allows the definition of multiple Features Models for the same software system that are meaningful for system integrators with different needs and expertise.

Our aim is to simplify the system configuration phase by supporting the definition of feature models at multiple levels of abstraction using specialized vocabularies for each expert involved in system configuration.

The community of researchers, who keep implementing new algorithms for common robot functionalities as open source libraries, need tools that simplify the configuration of robotic control systems during test trials in various operational conditions. System integrators, who are expert in specific application domains, need tools for the configuration of robot control systems according to specific application requirements.

For this purpose, HyperFlex supports the composition of Feature Diagrams representing variability at different level of abstractions. At each level the feature names abstract the relevant concepts of the specific domain: low-level names represent functional and technical terms while high level names are closer to the application requirements. This approach ensures that the terminology is well known by the system integrators that operates on a specific level.

During the variability resolution process, the application domain expert operates only on the highest-level Feature Model and the selected features trigger the automatic selection of features in the lower levels Feature Models.

A. Feature Model Composition

Typically, the expert in robotic functionalities is interested in a representation of the control system variability that highlights the different algorithms implemented in the robot control system. For example, in [8] we have analyzed the variability in software library that implement motion planning algorithms. In this context, the relevant features are the type of bounding-box used by the collision-detection algorithm, the sampling strategy, and the type of kinematic model (e.g. single chain, multiple end-effectors).

Figures 2 and 3 shows a screenshot of the HyperFlex Toolkit that represents the Feature Models of the system capabilities for the RoboCup@Work scenarios. The former represents the perception capabilities and the latter expresses the manipulation and navigation capabilities.

The Feature Model is structured as a tree, where each node represents a system feature. A feature could correspond to a variation point (e.g. the *Local Planner* functionality) or a concrete variant (e.g. the *DWA* algorithm for local planning). A black circle on a child node (e.g. *Global Planner*) indicates that the Feature is mandatory, while a white circle indicates that the Feature is optional.

White triangles indicate that the child features are mutually exclusive, while black triangles indicate the cardinality of the

OR containement association. For example, the perception system can be configured with algorithms that can recognize only one specific type of object (e.g. *Screw*, *Nut*) or a generic object. Similarly, it can be configured to recognize only one or both types of Containers.

The application domain expert is interested in a representation of the system variability that specifies the application requirements supported by the robot control system more than its specific functionality.

Figure 4 depicts the Feature Model of the application requirements for the RoboCup@Work scenarios. It is structured around three main dimensions of variability, namely the type of task that the robot should perform, the characteristics of operational environment, and the available equipment.

For example, the perception system could be a *Depth Sensor* (e.g. a Kinect) or a *Stereo Camera* (e.g. a BumbleBee sensor).

According to the operational environment, the robot should be configured with different algorithms: a slow and complete motion planner is adequate for moving among static obstacles in narrow passages; instead, a fast and approximate motion planner is needed for dynamic environments.

Clearly, the system integrators should focus on the specification of the application requirements and should not be concerned with the functionality that implement them.

HyperFlex provides a tool that allows to link the Feature Model of the application requirements and the Feature Model of the system capabilities. For example, the system designer can specify that the feature *Precision Placement* in the *Requirements* Feature Model is linked to the feature *Dynamic Constraints* in the *Capabilities* Feature Model. Similarly, the feature *Static* obstacles is linked to feature *DWA* local planner.

B. Feature Refinement Models

HyperFlex allows the composition of Feature Models and the automatic generation of their instances according to the composition strategies described in the previous sections.

The proposed appraoch consists in defining a new transformation model (called *Feature Refinement Model*) that specifies links between the features of a parent Feature Model and the features of its child Feature Models. Figure 5 shows an example, where *FM_A* is a parent Feature Model and *FM_B* and *FM_C* are child Feature Models.

When a new instance of the parent Feature Model is created, the instances of the child Feature Models should be empty, i.e. none of the features is selected. This condition allows to create instances of the child feature models incrementally.

When a feature of the parent FM is selected, all the linked features should be included in the instance of the child FM.

A feature of the parent FM (e.g. feature a5 in Fig. 5) can be linked to several features of different child FMs (e.g. features b4 and c3). Similarly, several features of the parent FM (e.g. features a2 and a5 in Fig. 5) can be linked to the same feature of a child FM (e.g. feature b4).

It should be noted that some features of the parent FM (e.g. feature a6 in Fig. 5) might not be linked to any feature of the child FMs and vice versa (e.g. feature b3).

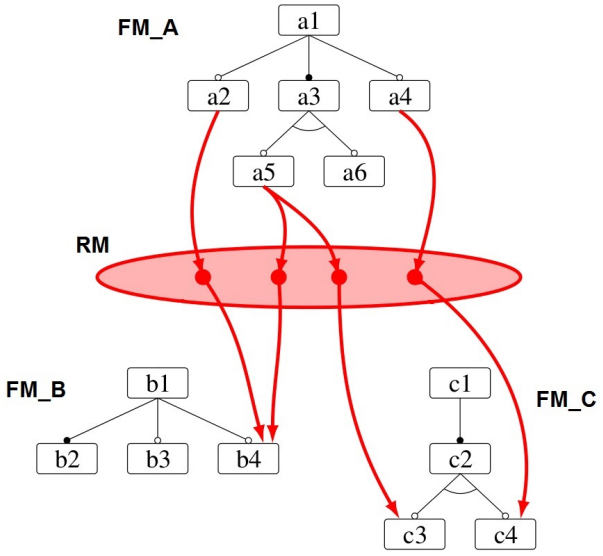

Fig. 5. A generalized visualization of the different feature model composition approaches available in HyperFlex. In the context of the case study Feature Model *FM_A* represents the requirements Feature Model (see Fig. 4) and *FM_B* and *FM_C* expresses the perception Feature Model (see Fig. 2) respectively the manipulation Feature Model (see Fig. 3).

The former case corresponds to the situation where the parent FM is used to configure directly some properties of a functional subsystem. For example, the selection of feature *Depth Camera* could be associated to a set of parameters that configure the perception system.

The latter case requires manual selection of some features of the child FM (e.g. feature *Cavity Recognition*).

Feature Models can include constraints that limit the set of possible combinations of selected features. For examples, features *c3* and *c4* in Figure 5 are mutually exclusive. It is not necessary to replicate the constraint in the parent Feature Model (i.e. *FM_A*), because the HyperFlex tool is able to report constraint violations in child FM to the user with the indication of the selected features in the parent FM that caused them.

The Feature Refinement Model defines a tree structure between a parent Feature Model and a set of child Feature Models. Starting from a manual selection of features in the parent FM, the HyperFlex tool generates instances of the child Feature Models automatically. This structure can be extended to trees with an arbitrary number of levels by connecting Feature Refinement Models hierarchically. Here, the hierarchy imposes an order according to which the Feature Refinement Models are processed in order to create an instance of each intermediate and leaf Feature Model.

The HyperFlex toolchain includes an Eclipse Wizard that supports the model designer in defining the Feature Refinement models by means of a set of intuitive Eclipse Forms.

IV. RPSL: Robot Perception Specification Language

The Robot Perception Specification Language (RPSL) [17] is a Domain-specific Language (DSL), implemented as an internal, textual DSL in Ruby, which provides suitable abstractions enabling domain experts to express the architectural variability of robot perception systems.

The RPSL enables a domain expert to model multi-stage perception systems by composing sensing and processing components in a perception graph which yields a directed, acyclic graph (DAG) where sensor and processing components are nodes. Here, sensing components represent sensors such as cameras and laser scanners and processing components encapsulate perception-related functions (e.g. filters, feature descriptors, and so forth). The sensing components solely produce data whereas processing components produce and consume data in a flow-oriented manner.

In our previous work [13] the abstract syntax (metamodel) and structural constraints of RPSL have been formalized using the Alloy formal modeling language. The Ruby-based RPSL implementation conforms to this formalization and ensures that all the specified constraints are satisfied which in turn yields well-formed RPSL domain models. To this end, checks are implemented to (a) ensure that the perception graph is a DAG, (b) connected ports have the same type, (c) input ports are connected, and so forth.

The applicability of feature models in the domain of robot perception [13] strengthened our vision to employ HyperFlex not only as a method, but also as a tool to model, configure and compose a robotic system based on several sub-functionalities which in turn are modeled by domain-specific approaches such as RPSL. In order to conceptually and technically integrate HyperFlex with RPSL the following ingredients are required:

- *Architectural Models* are specified in RPSL in order to emphasize the domain-specific, architectural aspects.
- *Feature Models* are specified in HyperFlex in order to encode the variant features of robot perception system.
- *Resolution Models* are defined in order to enable a model-to-model transformation, which allows to automatically configure the RPSL architecture based on the selected features in the aforementioned feature model. Such a configuration eventually requires to modify parameters of components or even to replace components from existing perception graphs.

In the context of the case study (see Sec. II) a different selection of task, environment and platform requirements significantly effects the perception architecture itself. Let us consider, for example, two applications with varying requirements.

The first application includes an omnidirectional robot equipped with a RGB-D sensor (e.g. feature *Depth Camera* selected) which is expected to place objects in containers located at service areas (e.g. feature *Simple Placement* selected). Further, the robot is deployed in an environment with static obstacles (e.g. feature *Static* selected). The second application differs in the task requirements where the robot is expected to precisely place objects (e.g. feature *Precision Placement* selected) in object-specific cavities (e.g. peg-in-hole task).

```
rpsl.feature_resolution do
   name "resolution"
   resolve "CavityRecognition"

   with "contour_detection"
   with "template_matching"
end
```

Fig. 6. An example of a resolution model which resolves the feature `Cavity Recognition` with `contour_matching` and `template_matching` components (see Fig. 7.)

Clearly, both applications require different perception capabilities in order to robustly perform the tasks. For example, in order to place objects in containers the container on a service area needs to be detected (e.g. Features *Service-Area Detection* and *Container Recognition* selected) whereas for the peg-in-hole task cavities need to be detected (e.g. Feature *Cavity Recognition* selected).

A. Integration of HyperFlex with RPSL

In order to achieve such a systematic composition on feature model level we employ the techniques described in Sec. III-B. Basically, three different feature model composition situations can be distinguished. Firstly, a feature of a parent feature model is linked to several features of different child feature models (see feature *a5* in Fig. 5). In the context of the case study the feature *Simple Placement* in Fig. 4 is linked both to the feature *Container Recognition* in Fig. 2 and a manipulation feature *Position Controlled* expressing a standard approach to place objects. Secondly, several features of the parent feature model are linked to the same feature of a child feature model (see feature *a5* in Fig. 5). Again, in the context of the case study the feature *Container Recognition* is linked to two features in the requirements feature model, namely the feature *Container* in case the environment is equipped with containers and for the task *Simple Placement*. Further, we link the feature *Precision Placement* with the feature *Cavity Recognition* which ensures that for this particular task the required perception capability is automatically selected. Thirdly, some features of the parent feature model are not linked to any features of the child feature models and vice versa (see feature *b3* in Fig. 5). For example, in the case study the feature *Rover* is not linked to any perception-related feature in Fig. 2.

In the next step, each feature belonging to the perception capability (see Fig. 2) is resolved in terms of one or more perceptual components or even perception graphs modeled with RPSL (see Fig. 7).

The corresponding resolution model governs a model-to-model transformation where the source model is an instantiated feature model containing the selected feature and the target model is an architectural specification of the configured perception graphs realizing the set of selected features. For

example, the feature *Cavity Recognition* is resolved by two components (see Fig. 6), namely *Contour Detection* in order to detect the object-specific cavities and *Template Matching* component which matches the detected contours with a set of a priori defined object-specific contours. The *Template Matching* component also computes the pose of each cavity using the centroid of the 3D contour with the x-axis along its principal axis. Nevertheless, for the precision placement task also other features are selected and the model-to-model transformation takes those resolutions into account and ensures that the resolved components are composable. To this end, it is checked whether their output and input types are compatible. For example, the mandatory feature *Service Area Detection* is resolved by the *RANSAC Plane Detection* component (see Fig. 7) which provides a plane which in turn is required by the contour detection component required for the *Cavity Recognition* feature.

The *Container Recognition* feature on the other hand is resolved by a perception graph composed of three components, namely *Euclidian Clustering* to cluster objects lying on the detected plane, *Bounding Box Detection* to compute a bounding box for each cluster which in turn are classified in containers in the *Container Recognition* component by using dimension and color criteria. It is important to note that both architectures are instantiated with the *Depth Camera* and *RANSAC Plane Detection* component as they are resolved by the feature *Depth Camera* (see Fig. 4) and the mandatory perception capability *Service Area Detection*.

```
rpsl.sensor_component do
   name "depth_camera"
   add_port :out, "out_port", "point_cloud"
end

rpsl.processing_component do
   name "ransac_plane_detection"
   add_port :in, "in_port", "point_cloud"
   add_port :out, "out_port", "plane"
end

rpsl.processing_component do
   name "contour_detection"
   add_port :in, "in_port", "plane"
   add_port :out, "out_port", "contours"
end

rpsl.processing_component do
   name "template_matching"
   add_port :in, "in_port", "contours"
   add_port :out, "out_port", "poses"
end

rpsl.perception_graph do
   name "precision_placement"
   connect "depth_camera", "out_port",
           "ransac_plane_detection", "in_port"
   connect "ransac_plane_detection", "out_port"
           "contour_detection", "in_port"
   connect "contour_detection", "out_port"
           "template_matching", "in_port"
end

rpsl.perception_graph do
   name "simple_placement"
   connect "depth_camera", "out_port",
           "ransac_plane_detection", "in_port"
   connect "ransac_plane_detection", "out_port"
           "euclidian_clustering", "in_port"
   connect "euclidian_clustering", "out_port"
           "bounding_box_detection", "in_port"
   connect "bounding_box_detection", "out_port"
           "container_recognition", "in_port"
end
```

Fig. 7. An excerpt of the models encoding the perception graphs required for the simple and precision placement task. One sensor component is modeled (`depth_camera`) and three processing components are modeled, namely `ransac_plane_detection`, `contour_detection` and a `template_matching`. Those components are connected in the `precision_placement` perception graph yielding a structurally complete specification of the perception capability required for the precision placement task. Both the `depth_camera` and `ransac_plane_detection` components are also used in the perception graph `simple_placement`. For the sake of readability configuration parameters of the components (e.g. sensor properties), data type definitions (e.g. `contours`, `plane`, and so forth) and the missing components for the simple placement graph are omitted.

V. Related Works

The following subsections illustrates the related works in three areas: (i) approaches to variability modeling, (ii) approaches to Feature Models composition, and (iii) variability modeling approaches in robotics.

A. Design Space Exploration

Modeling the architectures resolving features by domain-specific approaches as done in this paper raises the question how to systematically explore the resulting design space. That is, checking whether the resolved architectures are compatible from a structural, behavioral, functional and non-functional point of view. In [21] Saxena and Karsai already showed that design space exploration can benefit from MDE-based approaches. Their framework enables domain experts to define specification languages and the exploration of design spaces defined in these languages. In our work, we argue it is more desirable to reuse already existing tools, languages and technologies rather than implementing a framework from scratch.

B. MDE for software variability management

A survey of recent papers that propose techniques for Feature Model composition can be found in [3]. The surveyed approaches mostly focus on model composition techniques that are dedicated to support semantics preserving model composition. HyperFlex is a complementary approach, as it focuses on the automatic generation of Feature Model instances in a tree of variability models that are assumed to be semantically coherent and correct.

In GenArch [11] the variability model and the configuration model are represented using the same meta-model, while in OMG CVL [16] the variability model and the resolution model are not explicitly separated.

The *Compositional Variability* [2] approach supports the hierarchical composition of architectural models and feature models. The associations between a high-level feature model and a low level feature models are defined by means of the so called *Configuration Links*, which are similar to the feature dependencies defined in the HyperFlex *Refinement Model*. Differently from HyperFlex, this approach defines an abstract

component model and does not provide the capabilities for modeling domain-specific component-based systems.

The approach described in [15] defines three modeling categories. The *Commonality* describes the architecture of a system, in terms of components, sub-components, ports and connectors. These architectural elements can be enriched with variation points, which represent the *Variability* and define how the common parts can be configured. For example, a variant for a component variation point can specify that a new sub-component has to be included in the component. Finally, the *Configuration* describes the selection of variants for all the variation points. The architectural model and the configuration conform to the MontiArc meta-model. Differently from HyperFlex, this approach condenses all the information in a single model.

C. Variability Modeling Approaches in Robotics

In recent years, several model-driven approaches and tools for the development of robotic systems have been proposed, such as OpenRTM [5], Proteus [12], and Smartsoft [19].

In particular, the SmartSoft model-driven approach supports robotics variability management by modeling functional and non-functional properties of robot control system. The approach addresses two orthogonal levels of variability by means of two domain specific languages: (a) the variability related to the operations required for completing a certain task and (b) the variability associated to the quality of service.

These two variability levels are more related to the execution of a specific application (in the paper the example is a robot delivering coffee), while the HyperFlex approach supports modeling the variability of functional systems and the variability of the family of applications resulting from the composition of these functional systems.

VI. Conclusions and future works

In this paper, we presented the integration of HyperFlex, a model-driven toolchain for composing Feature models according to different composition strategies, with RPSL, a DSL to express architectural variability of robot perception systems. We demonstrated the feasibility of the approach by means of a realistic case study. The integrated tooling has been conceived for symplifying not only the configuration and deployment of complex control systems of autonomous robots on a functional, but also on an architectural level. Lastly, we would like to emphasize that this work motivates us to consider further integration activities of robotic DSLs [20] as both HyperFlex and RPSL where initially developed independently from each other, yet their integration was feasible and showed to be beneficial for structuring the overall development process.

References

[1] ROS: Robot Operating System. http://www.ros.org, 2007.

[2] A. Abele, H. Lönn, M.-O. Reiser, M. Weber, and H. Glathe. Epm: a prototype tool for variability management in component hierarchies. In *Proc. of the 16th Int. Software Product Line Conference-Volume 2*, pages 246–249. ACM, 2012.

[3] M. Acher, P. Collet, P. Lahire, and R. France. Comparing approaches to implement feature model composition. In *Modelling Foundations and Applications*, pages 3–19. Springer, 2010.

[4] F. Amigoni, E. Bastianelli, J. Berghofer, A. Bonarini, G. Fontana, N. Hochgeschwender, L. Iocchi, G. K. Kraetzschmar, P. U. Lima, M. Matteucci, P. Miraldo, D. Nardi, and V. Schiaffonati. Competitions for benchmarking: Task and functionality scoring complete performance assessment. *IEEE Robot. Automat. Mag.*, 22(3):53–61, 2015.

[5] N. Ando, S. Kurihara, G. Biggs, T. Sakamoto, H. Nakamoto, and T. Kotoku. Software deployment infrastructure for component based rt-systems. *Journal of Robotics and Mechatronics*, 23(3):350–359, 2011.

[6] R. Bischoff, T. Guhl, E. Prassler, W. Nowak, G. Kraetzschmar, H. Bruyninckx, P. Soetens, M. Haegele, A. Pott, P. Breedveld, et al. Brics-best practice in robotics. In *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1–8. VDE, 2010.

[7] D. Brugali. Model-driven software engineering in robotics. *IEEE Robot. Automat. Mag.*, 22(3):155–166, 2015.

[8] D. Brugali, W. Nowak, L. Gherardi, A. Zakharov, and E. Prassler. Component-based refactoring of motion planning libraries. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ Int. Conference on*, pages 4042–4049. IEEE, 2010.

[9] D. Brugali and P. Scandurra. Component-based robotic engineering (part i)[tutorial]. *Robotics & Automation Magazine, IEEE*, 16(4):84–96, 2009.

[10] D. Brugali and M. Valota. *Software Variability Composition and Abstraction in Robot Control Systems*, pages 358–373. Springer International Publishing, Cham, 2016.

[11] E. Cirilo, U. Kulesza, and C. Lucena. A product derivation tool based on model-driven techniques and annotations. *Journal of Universal Computer Science*, 14(8):1344–1367, 2008.

[12] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi, and M. Ziane. Robotml, a domain-specific language to design, simulate and deploy robotic applications. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 149–160. Springer, 2012.

[13] L. Gammaitoni and N. Hochgeschwender. Rpsl meets lightning: A model-based approach to design space exploration of robot perception systems. In *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2016. to appear.

[14] L. Gherardi and D. Brugali. Modeling and Reusing Robotic Software Architectures: the HyperFlex toolchain. In *IEEE International Conference on Robotics and Automation (ICRA 2014)*, Hong Kong, China, May 31 - June 5 2014. IEEE.

[15] A. Haber, H. Rendel, B. Rumpe, I. Schaefer, and F. Van Der Linden. Hierarchical variability modeling for software architectures. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 150–159. IEEE, 2011.

[16] O. Haugen, A. Wasowski, and K. Czarnecki. Cvl: Common variability language. In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, pages 277–277, New York, NY, USA, 2013. ACM.

[17] N. Hochgeschwender, S. Schneider, H. Voos, and G. K. Kraetzschmar. Declarative specification of robot perception architectures. In *4th International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2014.

[18] G. K. Kraetzschmar, N. Hochgeschwender, W. Nowak, F. Hegger, S. Schneider, R. Dwiputra, J. Berghofer, and R. Bischoff. Robocup@work: Competing for the factory of the future. In *RoboCup 2014: Robot World Cup XVIII [papers from the 18th Annual RoboCup International Symposium, João Pessoa, Brazil, July 15*, pages 171–182, 2014.

[19] A. Lotz, J. F. Inglés-Romero, C. Vicente-Chicote, and C. Schlegel. Managing run-time variability in robotics software by modeling functional and non-functional behavior. In *Enterprise, Business-Process and Information Systems Modeling*, pages 441–455. Springer, 2013.

[20] A. Nordmann, N. Hochgeschwender, D. Wigand, and S. Wrede. A survey on domain-specific modeling and languages in robotics. *Journal of Software Engineering in Robotics (JOSER)*, 7(1), 2016.

[21] T. Saxena and G. Karsai. Mde-based approach for generalizing design space exploration. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Part I*, MODELS'10, pages 46–60, 2010.

[22] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques. *Softw., Pract. Exper.*, 35(8):705–754, 2005.