International Journal of Semantic Computing
© World Scientific Publishing Company

Software Product Line Engineering for Robotic Perception Systems

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

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

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

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.

In various application domains software product line (SPL) development has proven to be the most effective approach to face this kind of challenges.

In a previous paper [11] we have presented a Model-based approach to the development of SPL for robotic perception systems, which integrates two modeling technologies developed by the authors: the HyperFlex toolkit [19] and the Robot Perception Specification Language (RPSL) [21].

This paper extends our previous work by illustrating the entire development process of a SPL for robot perception systems with a real case study.

Keywords: Robot perception; Software variability; Model-driven Engineering.

1. Introduction

Robots are versatile machines that are increasingly been used not only to perform dirty, dangerous, and dull tasks in manufacturing industries, but also to achieve societal objectives, such as enhancing safety in transportation, reducing the use of pesticide in agriculture, and improving efficacy in fight against crime and civilians protection. Compared to the manufacturing workcell, a public road, a corn field, or a crime scene are open-end environments, which require autonomous robots to be equipped with advanced cognitive capabilities, such as perception, planning, monitoring, coordination, and control in order to cope with unexpected situations reliably and safely.

Even a simple robotic application, like moving a wheeled robot from place A to place B in an indoor environment, requires several capabilities, such as (1) sensing the environment in order to avoid unexpected obstacles (i.e. moving people), (2) planning a path from A to B taking into account several constraints (e.g. energy consumption), (3) controlling the actuators in order to execute the computed path correctly (i.e. with a given accuracy), and (4) reasoning about alternative courses of actions (e.g. waiting for a passage to get clear or plan a different path).

Robot control systems are typically designed as (logically) distributed component-based systems (see [13] 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 design and 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.

In various application domains software product line (SPL) development [41] has proven to be the most effective approach to face this kind of challenges. A SPL is a set of software-intensive systems that share a common set of features for satisfying a particular market segments needs. Each new application is built from the SPL repository of common software assets (e.g. architectural models, software components).

Few other papers dealing with software product lines for robotics can be found in the literature. They mostly refer to the mature and stable domain of industrial robotics [23, 24, 42] or illustrates specific robotic case studies [25, 4].

In a previous paper [11] we have presented a Model-based approach to the development of SPL for robotic perception systems, which integrates two modeling technologies developed by the authors: the HyperFlex toolkit [19] and the Robot Perception Specification Language (RPSL) [21].

The key characteristic of HyperFlex are the support to the symbolic representation and composition of the variability of a SPL at various levels of abstraction, from the requirements of the SPL devised during the domain analysis phase, to the functional specification of the various systems that can be generated from the SPL during the product generation phase. The HyperFlex approach builds on our experience in developing software architectures for robotic control systems in the context of the EU FP7 BRICS project [9].

RPSL is a Domain-specific Language (DSL) that enable domain experts to model the architectural variability of robot perception systems. RPSL supports the specification of robot perception systems by providing common architectural concepts and abstractions appearing in the robot perception domain.

This paper illustrates the entire development process of a SPL for robot perception systems using the modeling approach presented in our previous paper.

The paper is structured as follows. Section 2 reports on the related works. Section 3 presents the foundamental concepts of SPL Engineering and the specific issues related to the development of a SPL for robotic systems. Section 4 illustrates a case study related to semantic perception for industrial tasks with autonomous robots. Section **??** summarizes the functionality and the modeling languages supported by the HyperFlex toolkit. Section 6 summarizes the RPSL approach and the integration with HyperFlex. The relevant conclusions are presented in Section 8.

2. Related Works

Several MDE approaches and tools have been developed for Software Product Line management in a variety of application domains, such as multimedia communication [43] and enterprise information systems [34]. They all exploit the power of domain-specific modeling languages [37] to embed domain-specific knowledge in software tools. The following two subsections illustrates related works on MDE approaches for software variability management and Robotics-specific MDE approaches respectively.

2.1. *MDE for software variability management*

The common approach to model the variability of a software system, which consists in defining four models: (a) the architectural model defines the software architecture of the system in terms of implementation modules (classes, aspects, agents, components) and their interconnections; (b) the variability model describes the functional variability of the system using a symbolic representation (e.g. feature models [15] or the OMG CVL language [3]); (c) the resolution model defines the mapping between the symbols of the variability model and the implementation modules of the architectural model; (d) the configuration model consists in a specific set of variants for the variation points defined in the variability model.

In GenArch [14] the variability model and the configuration model are represented using the same meta-model, while in OMG CVL [3] the variability model and the resolution model are not explicitly separated. In our approach, the first three models are completely orthogonal, i.e. they can vary independently, while the configuration model is an instance of the variability model and can be hierarchically composed as described in the previous sections.

The *Compositional Variability* [5] 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

HyperFlex. Differently from HyperFlex, this approach defines an abstract component model and does not provide the capabilities for modeling domain-specific and heterogeneous component-based systems.

The approach described in [20] defines three modeling categories, i.e. *Commonality*, *Variability*, and *Configuration*. 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 and does not support modeling heterogeneous systems.

The *Talents* approach [35] aims at modeling and composing reusable functional features for configuring the behavior of a software system. A graphical environment simplifies feature composition. The Talents approach models functional features at the level of instances of a class in an object-oriented programming language. In contrast, HyperFlex models functional features at the level of software components and component-based systems.

In order to maximize the reuse between product families in consumer electronics, Philips' researchers defined the product population approach. The architectural variability among the products is modeled by means of the Koala component model, which provides variability mechanisms such as parameterization (for configuring components) and switches (for changing connections) [39]. Differently from HyperFlex, this approach does not model functional variability explicitely.

A model driven approach for the design of embedded component based systems is presented in [22]. The approach is based on the Flex-eWare component model, which defines an architectural model in terms of components, composites, interfaces, and connections as in HyperFlex., and additional concepts for modeling computational nodes (e.g. micro-controllers) and quality of service properties. The Flex-eWare component model allows the generation of source code for three different target platforms (eC3M, Fractal and OASIS). Differently from HyperFlex, Flex-eWare models software variability only at architectural level.

Feature models usually do not scale up when the number of variation points and variants becomes substantial, because a single and huge feature model is too complex to maintain and to be understood by humans. In [6] software variability is represented in several feature models, which focus on different aspects. The feature models can be then composed by means of two operators (insert and merge) that produce a larger feature model and preserve some properties (e.g. valid configurations).

The *staged configuration* approach described in [16] consists in a stepwise specialization of feature models, where separate models define the configuration choices available in each stage.

The HyperFlex approach differs from the above approaches as it allows modeling the software variability of each functional system with a separate feature model, which represents the provided functionalities. When several architectural models are composed into a more complex architectural model, a new feature model is defined, which represents its higher level composite functionalities. A selection of features at a higher-level triggers a selection of features in the features models at the level below.

2.2. *Robotics-specific MDE approaches*

In recent years, several model-driven approaches and tools for the development of robotic systems have been proposed, such as OpenRTM [8], Proteus [17], and SmartSoft [28]. An extensive survey about MDE approaches in robotics is given in the work of Nordmann *et al.* [29].

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.

The Proteus approach [17] introduces a modeling language called RobotML which is based on UML profiles and targeted to ease the design, simulation and deployment of robotic applications. To this end, RobotML is structured around four main packages, namely (a) the architecture package providing concepts to model software and hardware components, (b) the behavior package providing state machine concepts to model the behavior of software components, (c) the communication package providing concepts relevant for data and control flow among components, and (d) the deployment package providing concepts to define an assignment of a robotic system to a target platform, for example, a middleware or simulator. Although RobotML provides an impressive tooling for example for the sake of code generation concepts for modeling the functional architecture as proposed by HyperFlex are missing.

In [31] the authors propose the SHAGE (Self-Healing, Adaptive, and Growing softwarE) framework to generate automatically the software architecture of a service robot based on a task plan. The approach aims at optimizing the use of robot resources and at reducing the effort of the system integrator.

It is worth noting that the vast amount of MDE approaches in robotics mainly focus on providing abstractions and notations to enable developers to specify and implement architectural structures such as components, modules and their composition (e.g. [40], [29]). In this connection, domain-specific MDE approaches to represent the architectural structures are proposed by several authors. In [30] the authors propose a language workbench to engineer complex movement control architectures based on motion primitives and their coordination. In [33] the authors propose the SafeRobots Eclipse-based toolchain integrating textual and graphical DSLs for specifying and developing robotic systems. SafeRobots is mainly concerned with modeling the component architecture not only from a structural, but also from a non-functional point of view by attaching and declaring non-functional properties to components and sub-systems. All in common of those approaches is that they focus mainly on architectural concerns and not on the features which are resolved by the architectures.

Modeling the architectures resolving features by domain-specific approaches 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 [36] 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 this connection the research presented in this article aims to reuse and compose already existing tools, languages and technologies rather than implementing a framework from scratch.

3. Software Product Line Development for Robotics

Typically, a SPL is a strategic investment for organizations, which wants to achieve customer value through large commercial diversity of their proucts with a minimum of technical diversity at minimal cost. This business strategy is called *software mass customization* in [27] and can be adopted according to the following three models.

The *proactive* approach consists in designing a complete SPL to support the full scope of products. This is possible, when the organization knows precisely the requirements of its products and can define their common software infrastructure and overall architecture. In Robotics, the *proactive* approach can be reasonably applied to very specific domain, such as robotic planetary exploration, where big organizations like NASA or the European Space Agency (ESA) develop all the software for their robot control systems in-house, because of the uniqueness of their equipments.

The *extractive* approach consists in reengineering a pre-existing set of systems that have a significan amount of similarities in order to eliminate duplicated code and enhance interoperability of common functionalities. In Robotics, the *extractive* approach has been applied in very mature domains, such as Flexible Manufacturing Systems (FMS), where a strong committment of robotics and industry stakeholder can be ensured [42].

The *reactive* approach consisits in revising the design of the software infrastructure and overall architecture of the SPL in order to accomodate the unpredictable requirements of new products. In Robotics, the *reactive* approach is approapriate to the development of relatively simple systems, such as robotic lawn mowers, robotic floor cleaners, which are characterized by a limited sets of functionalities. New requirements may emerge, for example, from the need to integrate the robotic equipment with other devices in a smart environment.

Our investigation focuses on a fourth class of robotics systems, i.e. autonomous robots that require advanced capabilities, such as mobile manipulation, to performe complex tasks (e.g. logistics) in everyday environments (e.g. a hospital) Today, only proof of concepts of these kind of systems exist, which are typically developed for sake of research in the context of international challenges, such as robocup@home [2] and robocup@work [26]. These robots show a high degree of autonomy in tasks such as cleaning up a room by collecting objects and placing them in the right bins, reacting to an emergency situation that may occur in a normal house hold (i.e. fire in the kitchen), transporting and assembling mechanical parts in an industrial setting.

Adopting the SPL approach in this context is challenging due to the current practice in software development for autonomous robots: it is mostly based on community efforts, which are not coordinated by key stakeholders, such as robot manufacturers or software foundations. This situation is due to the fact that most complex autonomous robotic systems are still in the research phase [32] and have not yet found commercial exploitation. As a consequence, a huge corpus of software systems, which implement the entire spectrum of robot functionalities, algorithms, and control paradigms, is potentially available as open source libraries [10]. Unfortunately, their reuse even in slightly different application scenarios requires significant effort, because the assumptions about the computational and robotic hardware, the software infrastructure, and the robot operational environments are hidden and hard coded in the software implementation.

In this paper, we propose a highly decentralized approach to the development of robotic SPL, which does not assume that the developed assets (e.g. resuable components, stable architectures) and the product derivation are under control of a single developing organization as it is typical for SPLs. The approach envisages three main stakeholders:

- The community of researchers, who keep implementing new algorithms for common robot functionalities as open source libraries, increasing the variability of robotic control systems.
- Specialized software houses, who design SPL for specific robotic subdomains (e.g. robot navigation) and SPLs for robot application domains (e.g. logistics).
- System integrators, who develop robot applications with incresingly chal-

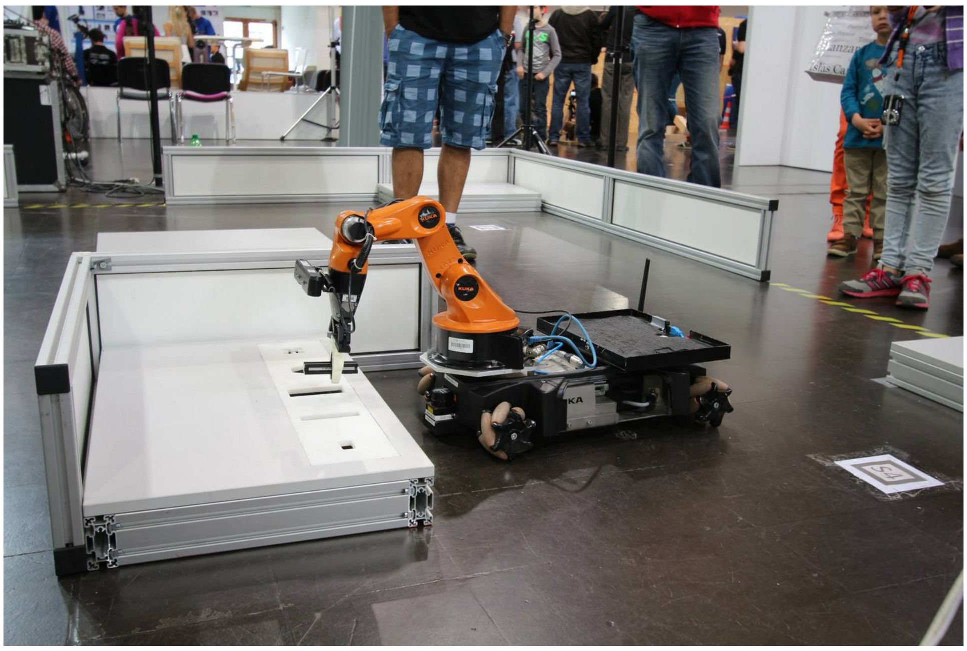

Fig. 1: A youBot mobile manipulation robot performing a precision placement task.

lenging requirements by customizing functional subsystems and applications to be deployed on specific robotic systems.

The proposed approach to the development of SPLs for autonomous robots takes into account the specific issues of software development for autonomous robots and is a mix of the three approaches discussed above. It is *proactive* as it is applied to the development of SPLs for functional sub-domains (i.e. motion planning, robust navigation, 3D perception), for which experts in specific functionalities can define stable software architectures. It is *extractive* as it promotes the development of core assets for the SPL (i.e. reusable components) through the refactoring and harmonization of existing open source libraries. It is *reactive* as it supports the development of higher-level SPLs for specific application domains (e.g. hospital logistics) as composition of SPLs for functional sub-domains.

4. 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 [26] and RoCKIn [7].

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

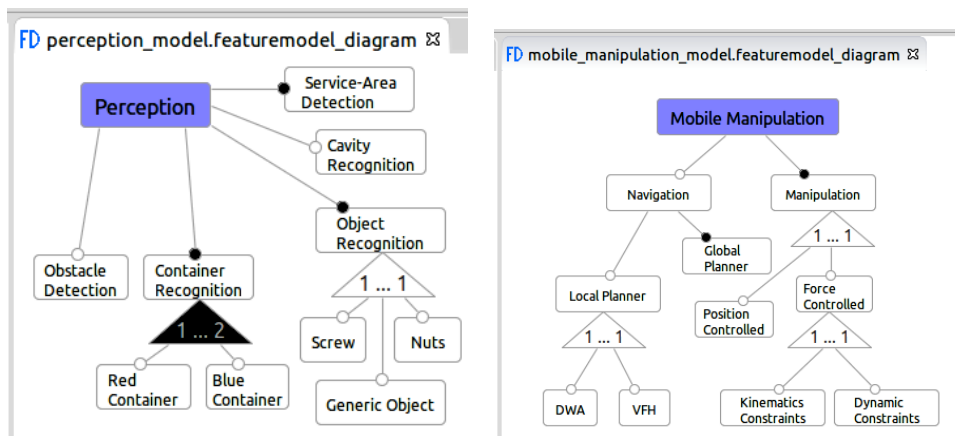

Fig. 2: Perception capabilities (left) and Manipulation and navigation capabilities (right) of the RoboCup@Work scenario.

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.

5. Domain Analysis with Feature Models

Feature Models symbolically represent the variant features [38] 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.

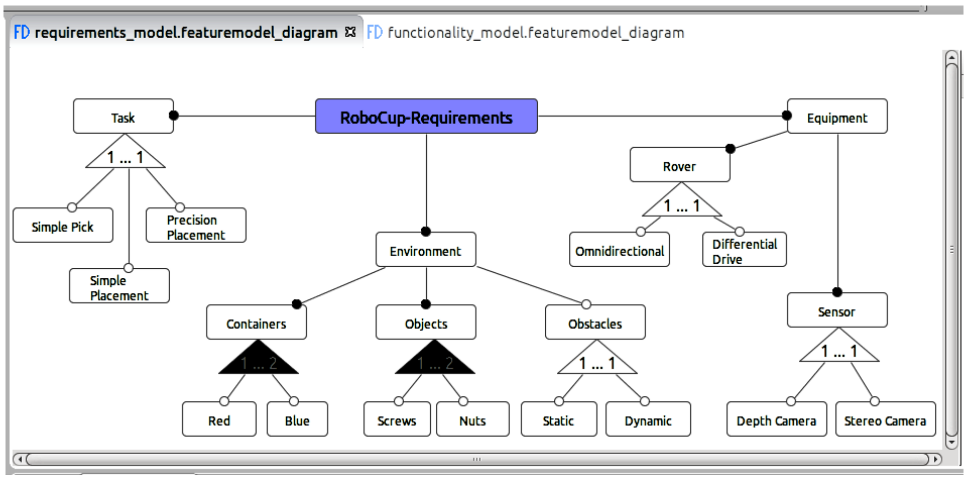

Fig. 3: Requirements of the RoboCup@Work scenario

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.

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 [12] 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).

Figure 2 shows two screenshots of the HyperFlex Toolkit that represents the Feature Models of the system capabilities for the RoboCup@Work scenarios. The left-hand side figure represents the perception capabilities and the right-hand side figure represents the manipulation and navigation capabilities.

Each 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 3 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.

5.1. *Feature Model Composition*

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

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.

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 4 shows an example, where *FM_A* is a parent Feature Model and *FM_B* and *FM_C* are child

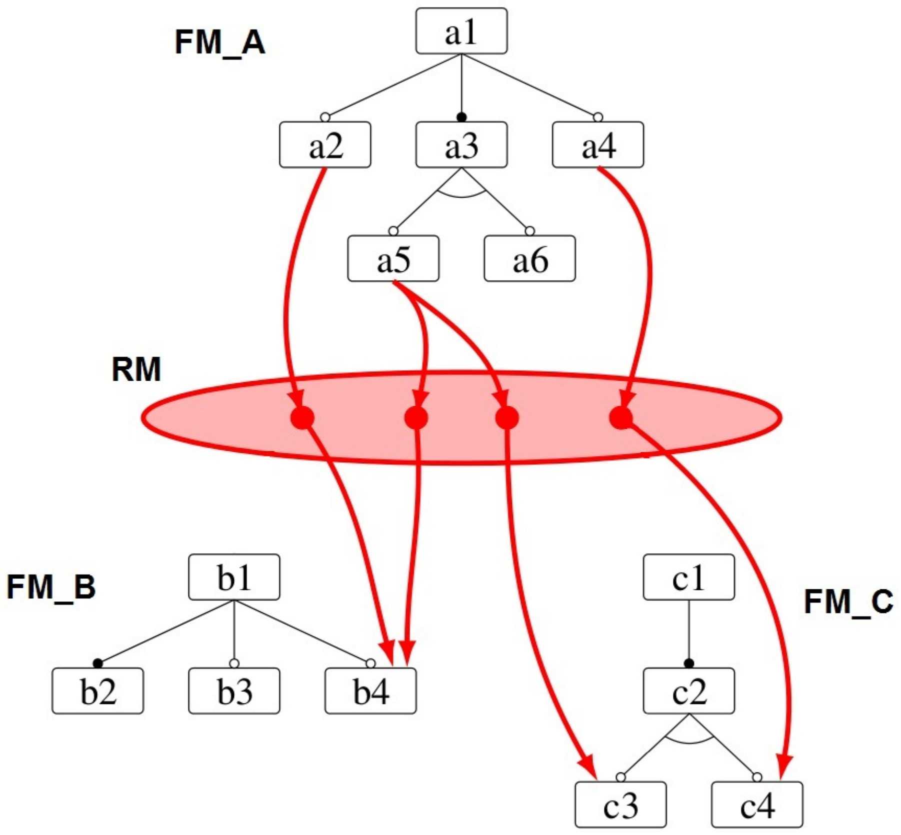

Fig. 4: A generalized visualization of the different feature model composition approaches available in HyperFlex. In the context of the case study Feature Model *FM_A* represents the requirements Feature Model (see Fig. 3) and *FM_B* and *FM_C* expresses the perception Feature Model (see Fig. 2) respectively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

Feature Models.

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

Basically, three different feature model composition situations can be distinguished.

Firstly, a feature of the parent FM (e.g. feature a5 in Fig. 4) can be linked to several features of different child FMs (e.g. features b4 and c3). In the context of the case study the feature *Simple Placement* in Fig. 3 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 FM (e.g. features a2 and a5 in Fig. 4) can be linked to the same feature of a child FM (e.g. feature b4). 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 (see feature *a6* in Fig. 4) and vice versa (see features *b2* and *b3* in Fig. 4).

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 4 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.

6. Architecture Design with RPSL

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).

The Robot Perception Specification Language (RPSL) [21] 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

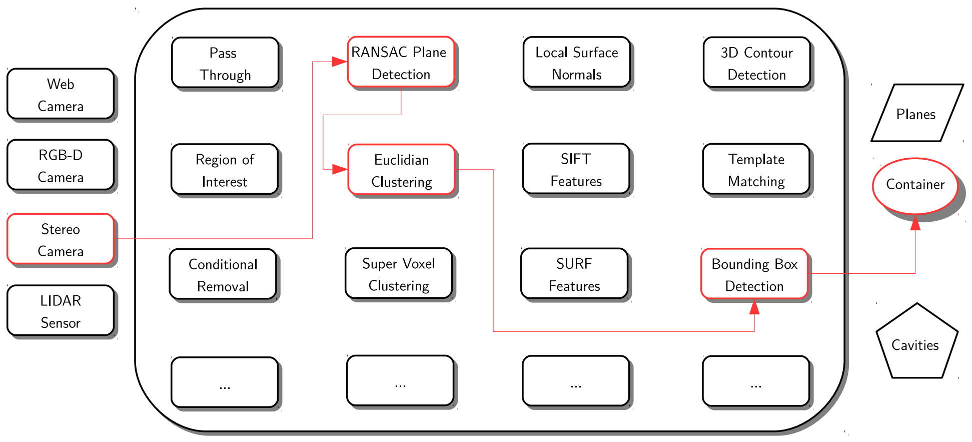

Fig. 5: Perception graph for the RoboCup@Work scenario

composing sensing and processing components in a perception graph which yields a directed, acyclic graph (DAG) where sensor and processing components are nodes. For example, Figure 5 visualizes the architectural variability of the perception system for the RoboCup@Work scenario. Here, sensing components on the left-hand side represent sensors which are typically available on mobile robots such as cameras, depth cameras and laser scanners. Those sensors produce a large amount of data which is processed by many and diverse processing components. Those 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. After performing several processing steps some output is provided as shown on the right-hand side of Fig. 5. Having RPSL at their disposal developers are enabled not only to specify domain models of sensing and processing components, but also to compose concrete processing graphs as the one highlighted in red (see Fig. 5). Those perception graphs represents a concrete architecture resolving a functional feature. For example, the graph shown in Fig. 5 represents the container recognition feature shown in Fig. 3.

In our previous work [18] 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.

```
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. 6: 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.

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

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

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

7. Resolution Models for Product Derivation

The applicability of feature models in the domain of robot perception [18] 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 the context of the case study (see Sec. 4) 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).

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).

In the next step, each feature belonging to the perception capability (see Fig. **??**) is resolved in terms of one or more perceptual components or even perception graphs modeled with RPSL (see Fig. 6).

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.

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.

For example, the feature *Cavity Recognition* is resolved by two components (see Fig. 7), 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. 6) 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. 3) and the mandatory perception capability *Service Area Detection.*

8. 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 [29] 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. Such a development process is often decomposed in several phases ranging from requirements analysis and functional design to implementation and deployment of robot architectures. The work presented in this article already covers, enhanced by MDE-based tools, several development phases and in our future work we will work on supporting often neglected, yet important development phases such as deployment and maintenance.

References

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

[2] Robocup @ Home League. http://www.robocup.org/robocup-home/, 2014.
[3] Common Variability Language (CVL), August 13, 2012. OMG Revised Submission. Oystein Haugen et al. - IBM, Fraunhofer FOKUS, Thales and Tata Consultancy Services.
[4] S. Abd Halim, N. A. J. Dayang, I. Noraini, and D. Safaai. An approach for representing domain requirements and domain architecture in software product line. In *Software Product Line - Advanced Topic*, pages 23–42. InTech, 2012.
[5] 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.
[6] 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.
[7] 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.
[8] 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.
[9] 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.
[10] J. Boren and S. Cousins. Exponential Growth of ROS [ROS Topics]. *IEEE Robotics & Automation Magazine*, 18:19–20, 2011.
[11] D. Brugali and N. Hochgeschwender. Managing the functional variability of robotic perception systems. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 277–283, April 2017.
[12] 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.
[13] D. Brugali and P. Scandurra. Component-based robotic engineering (part i)[tutorial]. *Robotics & Automation Magazine, IEEE*, 16(4):84–96, 2009.
[14] 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.
[15] E. Cirilo, I. Nunes, U. Kulesza, and C. Lucena. Automating the product derivation process of multi-agent systems product lines. *Journal of Systems and Software*, 85(2):258–276, 2012.
[16] K. Czarnecki, S. Helsen, and E. Ulrich. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10:143 – 169, 04/2005 2005.
[17] 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.
[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] M. Jan, C. Jouvray, F. Kordon, A. Kung, J. Lalande, F. Loiret, J. F. Navas, L. Pautet, J. Pulou, A. Radermacher, and L. Seinturier. Flex-eware: a flexible model driven solution for designing and implementing embedded distributed systems. *Softw., Pract. Exper.*, 42(12):1467–1494, 2012.

[23] B. Jørgensen and W. Joosen. Coping with variability in product-line architectures using component technology. In T. D'Hondt, editor, *Technology of Object-Oriented Languages, Systems and Architectures*, volume 732 of *The Kluwer International Series in Engineering and Computer Science*, pages 208–219. Springer US, 2003.

[24] E. Jung, C. Kapoor, and D. Batory. Automatic code generation for actuator interfacing from a declarative specification. In *Proc. of the 2005 IEEE/RSJ International Conference on*, pages 2839–2844, 2005.

[25] K. C. Kang, M. Kim, J. Lee, and B. Kim. Feature-oriented re-engineering of legacy systems into product line assets: a case study. In *Proceedings of the 9th international conference on Software Product Lines*, SPLC'05, pages 45–56, Berlin, Heidelberg, 2005. Springer-Verlag.

[26] 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.

[27] C. W. Krueger. Easing the transition to software mass customization. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 282–293, London, UK, UK, 2002. Springer-Verlag.

[28] 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.

[29] 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.

[30] A. Nordmann, S. Wrede, and J. Steil. Modeling of movement control architectures based on motion primitives using domain-specific languages. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5032–5039, May 2015.

[31] Y.-S. Park, H.-M. Koo, and I.-Y. Ko. A task-based and resource-aware approach to dynamically generate optimal software architecture for intelligent service robots. *Softw., Pract. Exper.*, 42(5):519–541, 2012.

[32] E. Prassler. *Service robots in everyday environments: Where are we? Where are they?* Number 76 in Springer Tracts in Advanced Robotics. Springer, 2012.

[33] A. Ramaswamy, B. Monsuez, and A. Tapus. Saferobots: A model-driven approach for designing robotic software architectures. In *2014 International Conference on Collaboration Technologies and Systems (CTS)*, pages 131–134, May 2014.

[34] J. Recker, J. Mendling, W. M. P. van der Aalst, and M. Rosemann. Model-driven enterprise systems configuration. In E. D. 0001 and K. Pohl, editors, *CAiSE*, volume 4001 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2006.

[35] J. Ressia, T. Grba, O. Nierstrasz, F. Perin, and L. Renggli. Talents: an environment for dynamically composing units of reuse. *Software: Practice and Experience*, 44(4):413–432, 2014.

[36] 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.

[37] M. Strembeck and U. Zdun. An approach for the systematic development of domain-specific languages. *Softw., Pract. Exper.*, 39(15):1253–1292, 2009.

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

[39] R. van Ommering. Mechanisms for handling diversity in a product population. In *Fourth International Software Architecture Workshop.* Citeseer, 2000.

[40] D. Vanthienen and H. Bruyninckx. The 5C -based architectural Composition Pattern : lessons learned from re-developing the iTaSC framework for constraint-based robot programming. *Journal of Software Engineering for Robotics*, 5(May):17–35, 2014.

[41] D. M. Weiss and Chi. *Software Product-Line Engineering: A Family-Based Software Development Process.* Addison-Wesley Professional; Har/Cdr edition, 1999.

[42] R. Weiss, J. Doppelhamer, and H. Koziolek. Bottom-Up Software Product Line Design: A Case Study Emphasizing the Need for Stakeholder Commitment. In *Proc. 5th SEI Architecture Technology User Network Conference (SATURN'09)*, 2009.

[43] Y. Wu, A. A. Allen, F. Hernandez, R. B. France, and P. J. Clarke. A domain-specific modeling approach to realizing user-centric communication. *Softw., Pract. Exper.*, 42(3):357–390, 2012.