Variability Modeling of Service Robots: Experiences and Challenges

Sergio García[1], Daniel Strüber[1], Davide Brugali[2], Alessandro Di Fava[4], Philipp Schillinger[5], Patrizio Pelliccione[1,3], Thorsten Berger[1]
[1]Chalmers | University of Gothenburg, Sweden, [2]University of Bergamo, Italy, [3]University of L'Aquila, Italy, [4]PAL Robotics, Spain, [5]Bosch Center for Artificial Intelligence, Germany

ABSTRACT

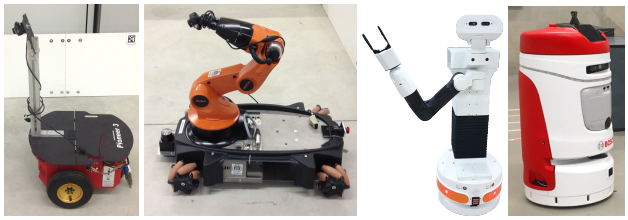
Sensing, planning, controlling, and reasoning, are human-like capabilities that can be artificially replicated in an autonomous robot. Such a robot implements data structures and algorithms devised on a large spectrum of theories, from probability theory, mechanics, and control theory to ethology, economy, and cognitive sciences. Software plays a key role in the development of robotic systems, as it is the medium to embody intelligence in the machine. During the last years, however, software development is increasingly becoming the bottleneck of robotic systems engineering due to three factors: (a) the software development is mostly based on community efforts and it is not coordinated by key stakeholders; (b) robotic technologies are characterized by a high variability that makes reuse of software a challenging practice; and (c) robotics developers are usually not specifically trained in software engineering. In this paper, we illustrate our experiences from EU, academic, and industrial projects in identifying, modeling, and managing variability in the domain of service robots. We hope to raise awareness for the specific variability challenges in robotics software engineering and to inspire other researchers to advance this field.

1 INTRODUCTION

As robots become increasingly important for our everyday lives, there is a growing need for software engineering practices for the robotic domain. Unlike in other embedded systems domains, which are strongly moving toward the definition of reference architectures (e.g., AUTOSAR [8] for automotive), defining a general reference architecture for robotic applications is elusive: robots can have many purposes, forms, and functions, can operate to accomplish various missions, and often operate in an open-ended environment. As a consequence, each robotic system has to be equipped with a specific mix of functionalities that strongly depend on several factors, such as the robot mechanical structure (a rover with zero or multiple arms), the tasks to be performed (cleaning a floor, rescuing people after a disaster), and the environmental conditions (indoor, outdoor, underground). These factors give rise to a multitude of variability dimensions that call for appropriate variability mechanisms and management methodologies.

VAMOS '19, Leuven, Belgium

DOI: 10.1145/3302333.3302350



(a) Pioneer (b) youBot (c) TIAGo (d) ITA

Figure 1: Mobile robots used in our projects

We report on experiences with variability engineering in various EU, academic, and industrial robotics software engineering projects. These projects focus on *service robots* [15], that is, mobile, ground, and intelligent robots that typically assist humans. In addition to our own experiences as five researchers and two industrial practitioners, for a richer perspective, we interviewed two colleagues who are experts at PAL Robotics (www.pal-robotics.com): a product manager with ten years of experience (I1) and a software engineer with nine years of experience (I2).

Our experience stems from the following projects. BRICS [5] aimed to provide researchers and developers with software tools and methods that simplify the configuration of a robot control software system. A key outcome of BRICS is the HyperFlex toolchain, which uses feature models [3] to support an automated robotic product generation process. The project Co4Robots (www.co4robots.eu) aims to produce robotic applications able to perform complex missions in a collaborative way. We provide the experience of our industrial partners and our role as the researchers software engineers in the project. PAL Robotics is involved in several EU projects on, among others, service robots and industrial robotics, benchmarking robotic frameworks, model-driven methodology, multiple-robot collaboration, and home-assisting robots. The experience at the Bosch Center for Artificial Intelligence (BCAI, www.bosch-ai.com) stems from a research project on coordinating multiple robots [29].

Figure 1 shows robots used in these projects: a Pioneer P3-DX robot equipped with a camera, a Kuka youBot equipped with a robotic arm, a TIAGo robot from PAL Robotics, and an Intelligent Transport Assistant (ITA). Pioneer P3-DX and Kuka youBot are widely used in robotics research. TIAGo is a service robot designed for indoor environments, combining mobility, perception, manipulation, and human-robot interaction capabilities. ITA is a research prototype used at BCAI that can be deployed in human environments, such as office buildings or factory shop floors, to transport small objects. It adapts to changes in its environment and to new tasks, and it interacts with humans by signaling the actions it is about to take. These robots are used in the remainder to illustrate our discussion.

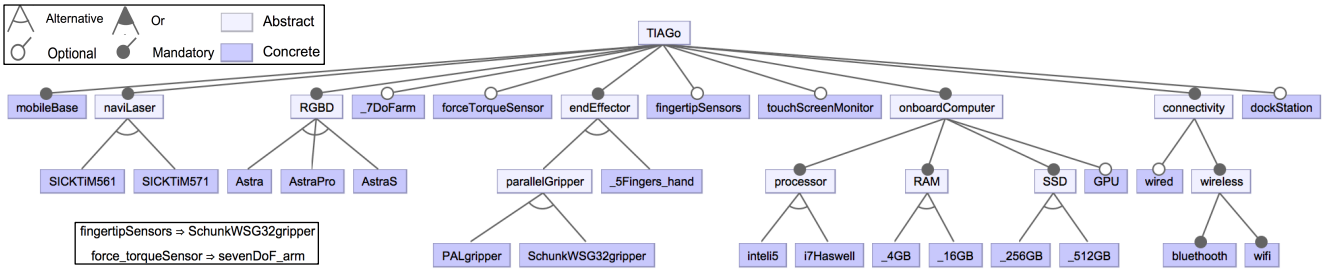


Figure 2: Feature model of the TIAGo robot (excerpt)

2 DRIVERS OF VARIABILITY

We identified four drivers of variability from own experience and practitioner interviews at PAL Robotics and BCAI. For each driver, we discuss the type of variability arising from it. We relate the drivers to an existing taxonomy from the literature, specifically Gherardi's [9] taxonomy of factors influencing the variability that needs to be handled when engineering robot systems.

Customer Requirements (CR). At PAL Robotics, customers can choose from a variety of options regarding hardware and software of the robot. Hardware options include the type of sensors, actuators, and accessories. The feature model in Fig. 2 illustrates the hardware variability for the TIAGo robot. Software options include both customizations (which are directly related to the hardware selection) and premium software. Examples of premium software packages are facial perception and automatic speech recognition.

Differences in requirements generally lead to static variability: depending on the customer's needs, different hardware and software components are deployed to the robot. Hardware choices of sensors and actuators define the required interfaces and controllers. On the same robotic model, different software might be deployed based on the customization that is required by a specific customer. This type of variability leads to having different variants of the same robot model. Like in other domains, varying customer requirements is a core driver of variability in robotics, despite being neglected in the state of the art, such as in the taxonomy of Gherardi [9].

Environment (E). Advanced scenarios require a robot working in a rich environment on a complex task. To illustrate: PAL uses its robot Tiago to detect, select, and pick objects from a table and place them in some other location. Such a scenario requires the robot to deal with an enormous variability of environmental conditions, such as the design of the objects, the table, and possible obstacles.

Some of the environmental variability can be resolved at design time. For example, visual markers used by a robot for orientation may be placed only on the floor. Then, the camera can be mounted in a fixed position on the robot (Fig. 1a). On the contrary, if markers are placed also on the walls or on individual items, the camera should be mounted on the robot arm (Fig. 1b). In the latter case, the arm motion should be adequately coordinated and specific functionalities need to be included in the control system to orientate the camera appropriately. However, a substantial degree of variability can only be resolved at runtime (e.g., obstacles of various kinds may require some flexibility in the behavior of the robot). Runtime variability is still a largely unsolved issue in the projects considered in this paper. We elaborate on this challenge shortly (Sec. 5).

In Gherardi's taxonomy [9], this variability driver is related to *robot situatedness*, or *context*—that is, robots operate in a dynamic and complex environment. To this end, robots must be aware of their own state and surroundings, which is typically achieved using a variety of sensors as well as navigation and perception algorithms.

Robot Hardware (RHW). PAL Robotics offers a scale of different hardware solutions to accommodate requirements and environmental conditions as identified above. Hardware variability also drives the variability of the associated software. For instance, at PAL Robotics, most robots share the same perception packages as they use the same type of visual sensors LIDAR and RGBD cameras. However, other robots do not include an RGBD camera and consequently require a dedicated perception package, which is developed by another company. Note that customer requirements and robot hardware drivers are strongly related in products offered by manufacturers as PAL. On the other hand, at BCAI, the slight differences in the specifics of the robots also imply a difference in certain configuration artifacts, in particular, calibration files.

Another driver of variability we experienced is robot *embodiment* [9]—how a robot body interacts with the environment through on-board sensors which provide stimuli that influence actions according to the hardware and software architecture. Embodiment directly affects the software, since, for instance, different navigation algorithms are used based on the kinematics of the robot (e.g., differential versus omni-directional drives).

Middleware (M). Different middlewares (or versions thereof) drive software variability, as middleware variability requires developers to cater for different APIs. Many frameworks and middlewares were released in the last years to ease the development of robotic applications, including ROS (Robot Operating System), Orocos, OpenRTM, and Smartsoft. While companies may try to stick to one framework to ease the development and reduce the variability, the same middleware may have different versions with different requirements, as we experienced first hand. For instance, a key challenge at PAL Robotics is to deal with different versions of the underlying middleware, ROS. PAL's goal is to reuse the robot application code across different ROS versions. In the BRICS project, two different middlewares were used to exploit their special capabilities. Gherardi's taxonomy [9] also mentions different frameworks as a driver (factor).

In the next two sections, we relate these variability drivers to the mechanisms—classified into *ad hoc* and *systematic*—we observed for addressing them, as shown in Table 1. Mechanisms either partially address drivers (i.e., some aspects of the driver is not managed) or fully address them (i.e., driver is fully and correctly managed).

Table 1: Variability drivers and mechanisms

	driver addressed by mechanism:	fully	partially
ad hoc	clone&own	M, E, CR	
	conf. facilities of rob. frameworks		RHW, E
	home-grown configurators	CR	RHW
systematic	feature models	RHW	E
	SPL architecture	M	
	framework heterogeneity		M
	SPL configuration	RHW, E	

3 AD HOC VARIABILITY MANAGEMENT

Experience shows that practitioners often rely on ad hoc strategies to manage variability. In our experience, the robotics domain is not an exception, typically using clone&own, configuration facilities of robotic frameworks, and home-grown configurators.

Clone&own. This readily available strategy [7] relies on copying and adapting existing software variants to new (customer or environmental) requirements. A prime example for using clone&own is to support different ROS distributions—a common practice at PAL. The company developed a framework that is based and built on top of ROS, which integrates PAL's own packages. According to interviewee 2, PAL's branching policy is the same as the one used in ROS (i.e., developers create a new branch for each new ROS version). PAL tries to have a single branch "master" to simplify maintenance. However, when a backward is incompatible (i.e. interfaces or data of earlier version cannot be successfully used by newer versions) the new code must be adapted, so PAL creates a new development branch. An example of such adaptation is some new code to adapt from ROS Indigo to ROS Kinetic. Then, the branch major version is changed (i.e., from 1.2.3 to 2.0.0). Finally, PAL leaves the old default branch with the name of the old development branch (e.g., "indigo-devel" branch), leaving it in the state of backwards compatible with the newer branch. Branches are maintained forever and backwards-compatible bug fixes are cherrypicked for all the branches if possible and necessary.

For the considered research projects at BCAI, a single long-term branch is usually sufficient. An exception to this are demo branches, which are created to support the specifics of a given demonstration scenario setup. However, development beyond the demo is usually not merged into these branches.

Configuration Facilities of Robotic Frameworks. A variety of software frameworks exists for developing robot control systems that are designed as (logically) distributed component-based systems (see Brugali et al. [6] for a survey). They offer mechanisms for real-time execution, synchronous and asynchronous communication, data flow, and control flow management. These frameworks are supported by a runtime infrastructure, which is in charge of instantiating, connecting, configuring, and activating the components of the system. The runtime executes these operations according to a set of instructions that are defined in a textual configuration file.

The BCAI research group is currently using ROS as a framework and utilizing some of the configuration mechanisms provided by it. In the most recent project of this group, environment, robots, and their behavior are modeled (e.g., the robots' behavior is modeled in terms of temporal logic). The expected behavior from the team of robots varies based on the context—the environment and the desired mission. To manage the variability of those models (variability related with RHW and E at runtime), the BCAI group makes use of ROS launch files at two levels. A general launch file manages the robotic team as a whole (e.g., number of robots, model of each robot) and specific ones define internal parameters of each robot (e.g., calibration files). In the same way, the definitions of the environment and the mission to be achieved are managed by launch files (e.g., selecting a certain transition system for the mission).

In the BRICS project, both ROS and Orocos were used for encapsulating robotic functionalities into software components due to their benefits (e.g., the ability of Orocos of real-time working or the huge number of open source resources available for ROS).

Home-Grown Configurators. As a manufacturer, PAL Robotics provides a customized product to clients (see Fig. 2). In the process from inferring the requirements from clients to the delivery of a full-functioning robot, PAL produces a number of artifacts. First, the customization of such product is summarized by the company in "requirements documents." PAL then ensembles the robot, deploys the required software to control the different hardware parts of the robot and validates it. Internally, the company uses an in-house GUI to select the implemented modules of the robot (e.g., which end-effector or which navigation laser), which automatically generates configuration files and ROS packages to be deployed in the robot. Those configuration files tell the robot which controllers (source code) must be launched and which parameters must be used for each module when the robot is started. Hence, the variability of each specific robot is mostly managed by that GUI. However, if the hardware architecture of a robot is changed (e.g., the gripper is replaced by another model) the configuration file has to be re-executed and deployed within the robot by using the GUI.

4 SYSTEMATIC VARIABILITY MANAGEMENT

In various application domains, software product line (SPL) development has proven to be effective for developing software control systems that are flexible enough to easily accommodate diverse requirements. We report on our experiences of developing LogisticSPL, a robotic software product line for robot-based logistic scenarios in the context of the BRICS project.

In BRICS, we aimed to develop methods and tools for the design, configuration, and composition of stable software architectures for specific functional sub-systems of an autonomous robot control system. Each functional sub-system is designed as a SPL, whose software architecture explicitly models software variation points and variants; variability is represented by a feature diagram [10]. The BRICS approach is supported by a set of meta-models and tools (the HyperFlex toolchain [11]) for (1) modeling the software architecture of component-based functional subsystems and its variability, (2) composing software architectures and variability models hierarchically, (3) deriving specific applications by configuring each subsystem at deployment time in a distributed environment.

We consider an application to modern logistic systems, which in a typical industrial setting exploits autonomous mobile manipulator robots for handling and moving objects from a source location to a target location. Typical tasks are, for example: transporting, loading, unloading, grasping, and placing objects. These tasks require the

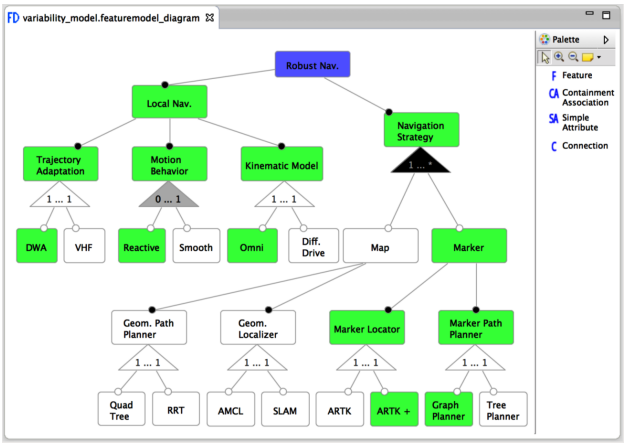


Figure 3: HyperFlex with a feature model of the Robust Navigation system (green features represent a configuration)

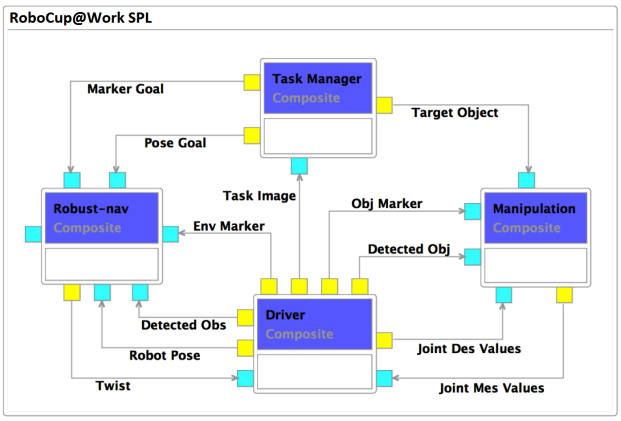


Figure 4: Architecture of the Logistic SPL

robot to have navigation, manipulation, and perception capabilities, where perception is typically tightly coupled with the specific approach adopted for implementing navigation and manipulation. We assume that the functionalities for navigation and manipulation are provided by functional systems developed by third-party research teams. The goal is to support the generation of specific control applications as products of the SPL by configuring the functional systems and application-specific components.

Feature Models. The feature model shown in Fig. 3 captures the functional variability of the navigation system, called *Robust Navigation*. HyperFlex uses a cardinality-based feature modeling [27]. The functionality *Robust Navigation* is realized by the set of functionalities represented by the mandatory features *NavigationStrategy*, *Robot Kinematic Model*, *Trajectory Adaptation*, and *Motion Behaviour*. Two navigation strategies are supported, i.e. the variants map-based and marker-based represented by the features *Map* and *Marker* respectively, which in turn are realized by a set of subfeatures associated with specific algorithms. For example, for the *Geometric Path Planning* variation point two variants are available: the algorithms Rapidly-exploring Random Trees [19] and Quad-tree [13]. Two robot kinematic models are supported: the *Omnidirectional* and the *Differential Drive*. Selecting one model rather than the other means using different implementations for the components *TrajectoryGenerator*, *TrajectoryAdapter*, and *TrajectoryFollower* in the *LocalNavigation* subsystem. The variation point *Motion Behavior* refers to the parameters that specify the acceleration limits considered when a trajectory is generated or adapted. Three variants are represented: if none of the children features is selected the default values are used; the *Reactive* feature corresponds to the maximum values of the limits; the *Smooth* feature corresponds to the minimum values of the limits. In conclusion, feature models may be used to manage variability coming from RHW and E (only at design-time).

SPL Architectures. The architecture of a SPL plays the role of reference architecture [25] for a family of products. While refactoring effort can never be completely avoided, it should be a goal of any product line development to define an architecture that will be mostly stable for future products [32]. Figure 4 depicts the architecture of the Logistic SPL, which is made of four subsystems. The *Robust Navigation* system is implemented in ROS. The *Manipulation* system, implemented in Orocos, provides the functionalities for object handling and recognition. It is also used for the marker based navigation when the markers are in variable positions and the camera is mounted on the arm. At least one of these two systems has to be included in the application architecture. The *Driver* system, also implemented in Orocos due to its real-time requirements, aggregates the hardware drivers. Finally, the *Task Manager* system is implemented in ROS and defines the coordination policies and some application specific components. The *Driver* and the *Task Manager* are mandatory systems. Since the *Robust Navigation* and *Manipulation* systems are optional (some logistic tasks require only one or the other functionality); also, all the connections are optional. This approach is used to manage variability in terms of M.

Dealing with Framework Heterogeneity. An open challenge in BRICS is to model the architecture of heterogeneous robotic control systems whose components are implemented using multiple software frameworks. A typical solution may involve an abstract component meta-model that can be specialized for different software frameworks, thus modeling the system architecture in a framework-independent way and specifying the target framework of each component at implementation time. This approach has two main drawbacks. First, some software frameworks (e.g., Smartsoft) already have their own component meta-model. Harmonizing these meta-models to be compliant with a common abstract component meta-model is complicated since existing model-driven tools depend on the framework-specific meta-models. Second, robotic frameworks support significantly different component models. For example, Orocos components use data-flow ports for asynchronous point-to-point communication, while ROS nodes exchange data and events through registers called topics according to the publisher/subscriber interaction paradigm. A common abstract meta-model would be highly generic and the tools developed on it would provide little support to the designer, partially addressing M.

SPL Configuration. The criteria for configuring the robot control system according to the desired functionalities, the environment characteristics, and the relevant constraints are not explicitly represented in the feature model, but embedded in a model-to-model transformation applied by the HyperFlex toolchain to automatically generate the corresponding architectural model that manages RHW. For example, the selection of a specific sensor triggers a transformation that includes the corresponding software driver in the control

system architecture, the selection of a given robot behavior triggers a transformation that set specific values for the robot dynamics parameters. The relevant constraints and the model-to-model transformations are defined by the team of domain experts and software engineering experts, who design and implement the robotic SPL.

The most desirable feature selection may depend on the environment (managing variability from E). For example, in the path planning algorithm, the most adequate strategy for obstacle avoidance depends on the type of obstacles present: no obstacles, only static ones, or also moving ones. The HyperFlex toolchain is currently geared to static variability, i.e., situations in which the environment is fixed so that a most desirable configuration can be specified at design time. Configuration updates at runtime are not supported.

5 DYNAMIC VARIABILITY

According to our experience, dynamic variability is an unsolved and ongoing common challenge. However, different groups have different approaches and, therefore, tackle it in different ways.

Robot Configuration. To deal with variability, robotic hardware and software must be built in a modular way. As stated previously, the robot configuration at PAL is managed at design-time by using in-house tools. While this is a proper solution for managing static variability, it prevents robot (re)configuration at run-time since the configuration files must be re-generated each time the robot structure is changed. There is no clear solution for this issue and they plan to keep on working with their current approach.

At BCAI, the configuration is managed at launch-time by loading different configuration files (see Sec. 3) by using launch files at different levels (general and robot-specific ones). They assume a rather well-defined scenario where robots execute different missions as dynamically planned at run-time given the loaded configuration. However, splitting the configuration into different files might lead to non-deterministic situations with existing ROS tools. In particular, it is not clear what happens when multiple configuration files that specify the same parameter values are launched. The proposed solution is to load the configuration files in a deterministic way by extending the functionality of standard tools such as `roslaunch`.

Adaptation. The intelligence of each robot or the capacity of reacting and adapting to the context is closely related to runtime variability. For instance, robots must be able to adapt to a variety of environments or tasks. At PAL, adaptation is managed by means of state machines, which describe the behavior of each robot in specific scenarios and are created at design-time. This method is also used to react to failures. For instance, in the scenario where a TIAGo robot must detect, pick, and place specific objects, the robot may fail when grasping one object. The high-level controller implements specific states that trigger recovering tasks if a failure is detected (e.g., the position of the arm is no longer known). However, the complexity of their run-time adaptation algorithms is growing due to the addition of parallel execution states. Also, there are other challenges related with run-time variability, such as how to model the environment such that the robots are able to adapt to it. Another challenge is how to cope with a context where the robot must interact with humans, since human behavior is hard to model.

For BCAI, the high-level adaptation is rather static (normally defined in temporal logics), while the low-level components must adapt dynamically (e.g., obstacle avoidance). To cope with environmental and task variability, the practice was to encode rules to specify the behaviors of robot teams. However, BCAI realized that the more rules exist, the more complex it is to dynamically plan and react. A possible solution is to adopt a software tool that allows developers to create and manage temporal logic rules. Furthermore, it seems to be useful a systematic evaluation of which rules should be incorporated in the task and which ones instead in the system model.

To tackle challenges based on adaptation, both BCAI and PAL are investigating solutions based on machine learning.

6 CHALLENGES

From our experience and the experience of our interviewees, we identify a list of challenges related to variability modeling and discuss possible solutions. Some challenges are common to other domains (e.g. automotive) while others are specific to robotics.

Multi-Purpose Robots. A major challenge is to design robots that are conceived to be generic so to be used for various purposes. Users will be expected to be able to specify in a precise way the mission that a robot or a set of robots should accomplish. The challenge is exacerbated by the fact that in the near future robots will be used for tasks of everyday life and the users will lack expertise in robotics and ICT [14]. This calls for domain-specific languages (DSLs) for effectively specifying missions [23, 24, 26, 28]

Dynamic Variability. Challenges related to dynamic variability management are the most common ones in our joint experiences: dynamic reconfigurations of the system are often made impossible by static code generation. Run-time adaptation is complicated by combinatorial explosions and by the difficulty of modeling human behaviors. These challenges are discussed in more detail in Sec. 5.

Maintenance Cost of Configuration Tools. At PAL, the in-house configuration tools sometimes create duplicated code and configuration files, thus increasing the maintenance cost. Their planned solution is to *"streamline their process by creating a more surgical approach"* (I2). Another solutions could be using a commercial configurator (e.g., pure::variants) or one from systems software [4].

Software Variability from Hardware Variability. As robot manufacturers, PAL offers a list of robotic hardware platforms with different features that must be controlled separately. To cope with this type of variability they strive to produce common software compatible for all their platforms. This increases the development time and complexity, but it also tends to lead to more structured code configurable via parameters or plugins, I2: *"That means that for a grasping task we cannot make assumptions regarding the number of arms of the robot, whether it's biped or not, or the sensors it has."*

Framework Heterogeneity. Dealing with multiple robotic software frameworks as target platforms is another challenge. In Sec. 4, we considered a solution based on a generic meta-model that can be specialized for each framework. However, the solution had two main drawbacks: it does not deal well with existing tools that rely on available framework-specific meta-models, and these meta-models are too heterogeneous to allow a convenient abstraction.

Lack of Resources. The lack of resources employed to manage variability is a common issue at several entities. A possible reason is that variability management is often considered a secondary

goal. Possible solutions are new variability modeling approaches focusing on automation, and a better education for team members.

7 RELATED WORK

There have been a number of initial efforts to manage variability in the robotics domain. A first group of works deals with the use of software product lines for robotics. Jørgensen et al. [16] introduce a systematic approach to handle variability in SPL architectures by the identification and implementation of well-designed software components. Likewise, another work [17] shows initial results of how a GUI is used for the automatic generation of robot hardware interfacing software, consisting of 41 features. However, they both refer to the mature and stable domain of industrial robotics. Other works illustrate specific robotic case studies [1, 12, 18].

Another line of research focuses on the use of domain-specific languages (DSLs) for variability management in robotics. Lotz et al. [21, 22] propose an approach for managing two variability dimensions. *Variability of operation* refers to the flexible planning of sequences of actions to achieve a higher-level task. *Variability of QoS* refers to the retention of non-functional properties, such as safety or task efficiency, under changing conditions. The authors propose two DSLs, one for each dimension, and a solution for integrating them in a consistent way during runtime. Tewfik et al. introduce a DSL for specifying the structure and behavior of a robot team and generating code to several execution platforms [31]. Our experiences as reported in this paper emphasize the need to address variability of operation, QoS, and execution platforms. However, the integration of all these variability dimensions is an open challenge. Existing work on managing several variability dimensions offers solutions based on the explicit modeling of multiple viewpoints [2, 20]. However, these works do not address the distinct challenges identified in this paper, arising from interactions of different variability dimensions that specifically appear in the robotics domain.

Model Driven Engineering has been applied recently to the field of robotics [30]. This methodology is required to compose systems out of exchangeable components and move towards well-engineered system development processes instead of the current craftsmanship practices. Steck et al. propose a system that exploits models used at design- and run-time to build robotic systems.

8 CONCLUSION

We reported on our experience with variability engineering in various EU, academic and industrial projects. In the industrial projects, we found that variability is still managed mostly ad hoc, using clone-and-own or some initial configuration mechanisms. The considered academic project included a full-fledged application of software product line engineering. In all cases, we found a need to better support dynamic variability. We also identified a number of challenges to be addressed in the future. While many of the identified challenges are well-known from general variability research, some of them are unique to the robotics domain, where the overarching mission specification plays a key role. An important challenge is the integration of the multiple variability dimensions, which calls for an integrated unified modeling and management approach.

ACKNOWLEDGEMENTS

Supported by EU H2020, Vinnova Sweden, and the Swedish Research Council. We thank Jordi Pages, Luca Marchionni, and Francesco Ferro for support and assistance with technical information.

REFERENCES

[1] S. Abd Halim, N. A. J. Dayang, I. Noraini, and D. Safaai. 2012. An Approach for Representing Domain Requirements and Domain Architecture in Software Product Line. In *Software Product Line - Advanced Topic*.
[2] R. Bashroush, I. Spence, P. Kilpatrick, J. Brown, and C. Gillan. 2008. A multiple views model for variability management in software product lines. (2008).
[3] T. Berger, R. Rublack, D. Nair, J. Atlee, M. Becker, K. Czarnecki, and A. Wąsowski. 2013. A Survey of Variability Modeling in Industrial Practice. In *VaMoS*.
[4] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. 2013. A Study of Variability Models and Languages in the Systems Software Domain. *IEEE Transactions on Software Engineering* 39, 12 (2013), 1611–1640.
[5] R. Bischoff, T. Guhl, E. Prassler, W. Nowak, G. Kraetzschmar, H. Bruyninckx, P. Soetens, M. Haegele, A. Pott, P. Breedveld, J. Broenink, D. Brugali, and N. Tomatis. 2010. BRICS – Best practice in robotics. In *ROBOTIK*.
[6] D. Brugali and P. Scandurra. 2009. Component-based robotic engineering (part i)[tutorial]. *Robotics & Automation Magazine, IEEE* 16, 4 (2009), 84–96.
[7] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki. 2013. An Exploratory Study of Cloning in Industrial Software Product Lines. In *CSMR*.
[8] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, and K. Lange. 2009. AUTOSAR–A Worldwide Standard is on the Road. In *14th International VDI Congress Electronic Systems for Vehicles*.
[9] L. Gherardi. 2013. Variability modeling and resolution in component-based robotics systems. (2013).
[10] L. Gherardi and D. Brugali. 2011. An eclipse-based Feature Models toolchain. In *6th Italian Workshop on Eclipse Technologies (EclipseIT 2011)*.
[11] L. Gherardi and D. Brugali. 2014. Modeling and Reusing Robotic Software Architectures: the HyperFlex toolchain. In *ICRA*. Hong Kong, China.
[12] L. Gherardi, D. Hunziker, and G. Mohanarajah. 2014. A software product line approach for configuring cloud robotics applications. In *CLOUD*.
[13] Y. Han, J. Jeong, and J. Kim. 2012. Quadtree based path planning for Unmanned Ground Vehicle in unknown environments. In *ICCAS*. IEEE, 992–997.
[14] IFR. 2016. World Robotic Survey. https://ifr.org/ifr-press-releases/news/world-robotics-survey-service-robots-are-conquering-the-world-.
[15] IFR. 2018. Service robots. https://ifr.org/service-robots/.
[16] B. Jørgensen and W. Joosen. 2003. Coping with variability in product-line architectures using component technology. In *Technology of Object-Oriented Languages, Systems and Architectures*.
[17] E. Jung, C. Kapoor, and D. Batory. 2005. Automatic code generation for actuator interfacing from a declarative specification. In *IROS*.
[18] K. Kang, M. Kim, J. Lee, and B. Kim. 2005. Feature-oriented re-engineering of legacy systems into product line assets: a case study. In *SPLC*.
[19] S.M. LaValle. 2006. *Planning algorithms*. Cambridge university press.
[20] S. Liaskos, L. Jiang, A. Lapouchnian, Y. Wang, Y. Yu, J. do Prado Leite, and J. Mylopoulos. 2007. Exploring the Dimensions of Variability: a Requirements Engineering Perspective. *VaMoS* 7 (2007), 17–26.
[21] A. Lotz, J. Inglés-Romero, C. Vicente-Chicote, and C. Schlegel. 2013. Managing Run-Time Variability in Robotics Software by Modeling Functional and Non-functional Behavior. In *EMMSAD*.
[22] A. Lotz, J. F. Inglés-Romero, D. Stampfer, M. Lutz, C. Vicente-Chicote, and C. Schlegel. 2014. Towards a Stepwise Variability Management Process for Complex Systems: A Robotics Perspective. *Int. J. Inf. Syst. Model. Des.* (2014), 55–74.
[23] C. Menghi, C. Tsigkanos, T. Berger, and P. Pelliccione. 2019. PsALM: Specification of Dependable Robotic Missions. In *ICSE*.
[24] C. Menghi, C. Tsigkanos, T. Berger, P. Pelliccione, and C. Ghezzi. 2018. Property Specification Patterns for Robotic Missions. In *ICSE*.
[25] E. Nakagawa, P. Antonino, and M. Becker. 2011. Reference Architecture and Product Line Architecture: A Subtle but Critical Difference. In *ECSA*.
[26] A. Nordmann, N. Hochgeschwender, and S. Wrede. 2014. A Survey on Domain-Specific Languages in Robotics. In *SIMPAR*. Springer.
[27] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. 2002. Extending feature diagrams with UML multiplicities. In *IDPT*. Citeseer.
[28] D. Di Ruscio, I. Malavolta, P. Pelliccione, and M. Tivoli. 2016. Automatic Generation of Detailed Flight Plans from High-level Mission Descriptions. In *MODELS*.
[29] P. Schillinger, M. Bürger, and D. V. Dimarogonas. 2018. Simultaneous Task Allocation and Planning for Temporal Logic Goals in Heterogeneous Multi-Robot Systems. *The International Journal of Robotics Research* 37, 7 (2018), 818–838.
[30] A. Steck, A. Lotz, and C. Schlegel. 2011. Model-driven Engineering and Run-time Model-usage in Service Robotics. In *GPCE*.
[31] Z. Tewfik, F. Jean-Loup, S. Serge, Z. Mikal, D. Saadia, M. Francois, M. Nicolas, N. Cyril, K. Selma, and P. Bruno. 2016. A Toolset to Address Variability in Mobile Robotics. *Journal of Software Engineering in Robotics* (2016).
[32] C. Tischer, B. Boss, A. Müller, A. Thums, R. Acharya, and K. Schmid. 2012. Developing Long-Term Stable Product Line Architectures. In *SPLC*.