Received May 3, 2022, accepted May 11, 2022, date of publication May 16, 2022, date of current version May 20, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3175505

RuVa: A Runtime Software Variability Algorithm

ALEJANDRO VALDEZATE[1], RAFAEL CAPILLA[1], (Senior Member, IEEE), JONATHAN CRESPO[1], AND RAMÓN BARBER[2], (Senior Member, IEEE)

[1]Department of Informatics, Rey Juan Carlos University, 28933 Madrid, Spain
[2]Department of Systems Engineering and Automation, Universidad Carlos III of Madrid, 28911 Madrid, Spain

Corresponding author: Rafael Capilla (rafael.capilla@urjc.es)

ABSTRACT Context-aware and smart systems that require runtime reconfiguration to cope with changes in the environment increasingly demand variability management mechanisms that can address runtime concerns. In recent years, we have witnessed new dynamic variability solutions using dynamic software product line (DSPL) approaches. However, while few solutions proposed so far have addressed the need to add, change and remove variants dynamically, none of them provide a way to check the constraints between features at runtime. Because all SAT solvers perform variability constraint checking in off-line mode, we suggest in this ongoing research paper the integration of RuVa, a runtime variability algorithm, with the FaMa tool suite to check feature constraints dynamically before a new feature is added or an existing feature is removed. This research suggests a novel approach to modifying the variability model of context-aware systems dynamically and check the feature constraints on the fly. We integrate our solution with a SAT solver that can be invoked at runtime by a cyber-physical system. We validate the effectiveness and performance of the proposed algorithm using simulations. We also provide a proof-of-concept for updating the configuration of a robot's variability model based on contextual changes.

INDEX TERMS Software variability, runtime variability, dynamic software product lines, feature model, context features, reconfiguration, robots.

I. INTRODUCTION

Nowadays, many systems demand runtime reconfiguration to adapt their behavior to context changes. For more than 15 years, the MAPE-K (Monitoring, Analyzing, Planning and Executing plus Knowledge) loop [1], [2] has been used for reconfiguring systems (e.g. robots [3]). Autonomous robots are a good example, as they are complex systems that need to continuously adapt their behavior and adapt to their environment in real-time [4]. For these systems it is important to guarantee robustness in the face of changing operating conditions [5]. Thus, robot navigation systems must adapt to the different situations during their navigation by changing the representation model of the environment. This leads to the software adaptation of their control, modeling, and planning algorithms including sensors and actuators [6].

However, many configurable options exhibited by systems with high variability demand additional solutions where smart systems can, not only update their system features but also add new features or remove existing ones with minimal or no human intervention. Typically, smart cities, appliances, industrial robots, and smart vehicles are examples of applications where some kind of smart reconfiguration or software update with new features is required. These domains often demand variability to configure their system options at runtime.

Software variability techniques have been used for more than three decades for representing and handling the variability of software systems [7], and use system features to reconfigure systems faster or produce different system configurations with less effort. Nevertheless, the majority of current approaches using software variability solutions configure their variants at pre-deployment time, typically used by Software Product Lines. As a consequence, the notion of Dynamic Software Product Lines (DSPLs) emerged in 2008 [8], [9] to address the runtime concerns of software variability not handled by conventional software product lines where variants are bound to their values at pre-deployment time.

A DSPL offers support for post-deployment activities when runtime reconfiguration is required. This autonomic behavior [10] provides support for product evolution and

The associate editor coordinating the review of this manuscript and approving it for publication was Mahmoud Elish.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

adaptation exploit in many cases context-aware knowledge to realize the adaptation goals and re-bind variants at runtime. Although DSPLs are not mature in the industry, they provide an extended dual-life cycle for runtime needs of systems using software variability [11].

One of the main differences between software product lines (SPLs) and DSPLs is the use of context features [12], [13] that represent that system features that could change according to varying context conditions. These approaches led to the notion of context-aware DSPLs as described in [14]. In addition, a dynamic or runtime variability mechanism [15] is central for DSPLs in order to support adding, changing, and removing variants dynamically and at post-deployment time and hence, handling the diversity of runtime scenarios. Another work [16] investigates the transition between different binding times and suggests transitions between different operational modes in a DSPL of a power plant control.

Nevertheless, there are still many open challenges not solved properly by incipient DSPL approaches. One of these challenges is how to check new feature constraints dynamically when features are added or removed at runtime. Consequently, in this ongoing research paper, we describe the integration between a dynamic variability mechanism and the feature constraint solver of the FaMa toolset in order to check the constraints at runtime when the structural variability is modified. Although we applied our solution in the robotics area, other application domains that could benefit from our approach are industrial systems using IoT devices (e.g. Industry 4.0 solutions) or for instance, smart vehicles that need to detect context-aware situations that demand a different kind of responses.

The remainder of this paper is as follows. Section II introduces the related work on runtime variability approaches and evaluating feature constraints dynamically. In Section III we outline our runtime variability algorithm and how it integrates with the FaMa tool suite, while in Section IV we evaluate the proposed efficiency and performance of the solution using simulation scenarios, and we provide a case study using a robot as proof of concept of its applicability in a real case, where the entire process of adaptation of the software to changes in its navigation models is detailed. We also describe the research questions that we address in this work. Section VI discusses the limitations of our approach and in Section VII we draw conclusions and future works.

II. RELATED WORKS

This section discusses related works on runtime variability approaches and how feature constraints can be checked at runtime.

A. RUNTIME VARIABILITY APPROACHES

Early approaches supporting basic runtime variability mechanisms for supporting automation and reconfiguration tasks of systems can be found in [17], [18]. In addition, variability transformations and the activation/deactivation of features during system execution are described for the case of smart home systems and aimed at handling different product reconfiguration [19]–[21]. Other experiences in the area of Wireless Sensor Networks use the FamiWare approach [22] to support runtime variability for reconfiguring variability models dynamically. In a similar vein, an approach in the same domain is discussed in [23], [24].

In the domain of mobile systems, a dynamic variability solution [25] is proposed to reconfigure variability at runtime and modeled using the Common Variability Language (CVL). The proposed approach used the notion of the MAPE-K loop from self-adaptive systems for adapting the application at runtime using a variability model and to optimize feature reconfiguration as well.

Nevertheless, none of the aforementioned works enable changing the structural variability at runtime and inserting new features in variability models. This facility is only suggested but they don't provide a real implementation because modifying variants and variation points dynamically is still challenging as it requires is most cases human intervention. This turns more complex when we need to modify a variation point as we need to redefine manually the logical formulas connecting the variants (i.e. features).

Some initial attempts to change variants dynamically can be found in [26], [27], while the authors in [28] use context-oriented programming (COP) solution to dynamically adapt the variations in the behavior of systems associated to specific contexts. In one recent work [29] the authors suggest a dynamic variability solution in the robotics domain and they suggest adding variants and values at runtime to the variability model of a robot as well as estimate the expected Quality-of-Service (QoS) properties of the robot during the self-adaptation task in order to find a better configuration. The approach adds variability constructs to behavior trees for modeling robot reconfiguration. Nevertheless, the proposed approach can't handle the automatic modification of the logical formulas connecting the variants nor suggest the best location to place new variants.

B. RUNTIME CONSTRAINT CHECKING

The role of dependencies between contexts is highly important as is quite similar to feature dependencies and feature interactions that appear in variability models, especially at runtime. Not many works discuss the verification of feature constraints at runtime in the context of software product lines. For instance, the work discussed in [30] highlights the importance of reconfiguration activities in DSPLs and the existence of different binding times for dynamically reconfigurable features. The configuration of a given DSPL according to the binding time selected requires modeling the binding time of the temporal constraints and the dependencies among different binding times.

In [31] the authors suggest a technique to model and verify the evolution and the dynamic behavior of a DSPL using adaptation rules aimed to activate and deactivate context features. Similarly, the proposal described in [32] explains the use of temporal constraints to model the reconfiguration

of a DSPL for Cloud systems. These constraints extend the traditional variability models to allow multiple reconfiguration alternatives adding constraints to enable the inclusion of temporal aspects and hence, control better the system's behavior. Nevertheless, the proposed solutions still don't solve those scenarios where features can be added or removed dynamically as they need to validate the constraints at runtime with the rest of the system' feature dependencies.

Additionally, the authors in [33] describe an approach for anytime diagnosis and reconfiguration capabilities. The proposed approach evaluates an algorithm in terms of performance and diagnosis quality of feature models and industrial configuration knowledge base from the automotive domain.

Related to DSPLs and its ability to perform runtime reconfiguration, the authors in [34] present an approach to model real-time constraints for DSPLs and how to analyze automatically constraint models. They apply their solution to a railways traffic scenario where several trains operate on the same track and demand reliable communication channels. Hence, the reconfiguration to solve such problems require checking real-time constraints dynamically that are modeled using a constraint language. Some recent works like the ProDSPL approach [35] suggest a DSPL able to anticipate to future variations and generate the best configuration proactively as the variability model is transformed into linear constraints that are optimized as part of the decision-making process. The approach is evaluated using mobile strategy games where some features are configured at runtime.

Finally, the authors in [36] an approach to verify the correctness and behavior of the system reconfiguration at runtime supported by context variability models that are used in dynamically adaptive systems. The approaches discussed above don't provide a way to add or remove feature constraints dynamically when new features are added on the fly. Also, handling new constraints is not easy as most of the time constraint solvers are executed in "off-line" mode separately from the runtime variability mechanism.

III. RuVa: A RUNTIME VARIABILITY ALGORITHM

In order to solve the aforementioned challenges, we evolved our runtime variability developed at Rey Juan Carlos University (URJC) ad documented for the first time in [37] into a more matured version integrated with the FaMa tool suite in 2021. The main idea is to support changes in the structural variability at runtime when features can be added, removed, or modified. As these operations are traditionally done manually by the designer, we attempt to go a step beyond previous works by automating the invocations to feature constraints solvers. Our algorithm can add and remove features dynamically based on a novel classification system that uses the so-called "super-types" and aimed to classify features in a variability model according to a set of predefined types. Hence, we support adding new features in different locations in an existing feature model by comparing the super-types of the existing features and the super-types of the new features. In case of having different possible locations or a tie where a feature can be placed in two or more locations, the algorithm makes a decision based on the number of common super-types and the logical relationship of the existing features and the new one is also compatible for a particular branch. Finally, in the case of different locations, the *Optimizer* selects the best place to include the new feature We also foresee a connection to external systems that use context-aware facilities (e.g. cyber-physical systems - CPS) where context-aware features are requested to RuVa in order to update the feature model.

Our solution foresees different scenarios to insert features automatically according to their common super-types, the possible locations of the new feature, and the combination of different logical relationships (i.e. AND, OR, XOR) where a parent feature has one or several children (i.e. subfeatures). In our solution we suggest four main scenarios, that is (i) No compatible super-type, (ii) one feature with compatible super-types without children features, (iii) one feature with compatible super-types with children features, and (iv) different features with compatible super-types. In this last scenario, it can be possible that a different location for the new feature may contain the same number of super-types, and in this case, RuVa will select the branch in the feature model less overloaded in terms of children features. In addition, the insertion will be performed accordingly to the logical relationships of the existing children's features and the logical relationship required by the new feature. Figure 1 shows an example of the fourth scenario where a new feature "Fx" is inserted in a branch that contains several super-types.

A. INTEGRATION WITH FaMa TOOL SUITE

This section describes our approach where we combined an existing runtime variability algorithm with FaMa feature constraint solver, being both elements part of our previous work and research. Consequently, we reused the FaMa solver module in order to avoid duplicating effort in programming a new solution and integrating existing ones, such as we explain in the next subsections.

The FaMa tool suite enables Automated Analysis of Feature Models [38]. Internally, each of the constraints existing in the model is translated to a logic paradigm which can be SAT or CSP among others based of. However, currently, FaMa only supports CSP-based solvers when coping with complex cross-tree constraints. The CSP model used by FaMa to prune the products that exceed the maximum budget: CSP=<F,FC,A>, where:

- F is a set of variables, f_i ϵ F, representing the selection state of each feature in the product line feature model. If the i_{th} feature is selected for a product test, then $f_i=1, $f_i = 1$ and, otherwise, $f_i = 0$.
- FC is the set of constraints that define the different relationships between different features (e.g. if the i_{th} feature is a mandatory child feature of the j_{th} feature, then $i_{th} \leftrightarrow j_{th}$) according to the mapping presented in [39].

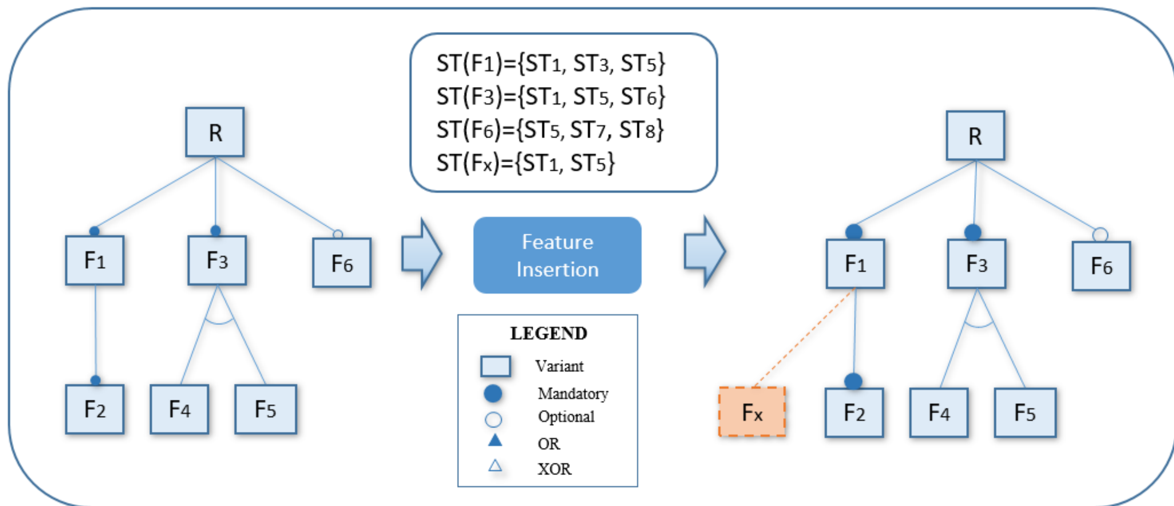

FIGURE 1. Inserting a feature which contains more than one super-type.

- A is the set of attributes describing the quality attributes of the features within the model. Note that FaMa currently only supports attributes that can be represented in the range of integers. This is, enumerations, integers, and Booleans.

Using this translation FaMa can perform questions to such models. For example, we can determine if a certain feature selection is valid or propagate the decisions within a configuration process.

B. DYNAMIC VARIABILITY CONSTRAINT CHECKING

Our solution to combine the runtime variability algorithm with FaMa's constraint feature solver is as follows. First, we adapted our runtime variability algorithm [31] to support FaMa feature models. We defined abstract features to adapt the feature models used by the runtime variability model to be readable by FaMa. This abstract feature helped us to indicate the cardinality of AND, OR, and XOR relationships.

The integration of the runtime variability algorithm and FaMa works in the following way: (i) we first upload the repository containing the list of features and the super-types, (ii) we upload our adapted feature model as a ".FM" file which can be read by FaMa (iii) we execute the runtime variability algorithm which calls FaMa solver to check the validity of the constraints list. FaMa reads the feature model and the constraints list in its own format. Previously, the runtime variability algorithm determines the best location in case a new feature is added. If the feature is removed, we only need to delete the feature and check the constraints list, and (iv) after checking the constraint model FaMa returns if the model is valid.

Figure 2 shows the calls between the runtime variability algorithm and FaMa. As we can observe in the figure, the feature model loader (*FMLoader*) uploads the target variability model for RuVa and can enact commands to add or remove features dynamically. Once an operation is selected, the *Feature Finder* locates different alternatives to add a new feature according to the comparison of super-types. Among several locations the *Optimizer* finds the most optimal location to insert a new feature and the result is returned to RuVa main control which invokes the FaMa Choco solver[1] to check the validity of the new constraints belonging to the new feature and the existing constraints in the feature model. The *Question Trader* module is in charge of sending the appropriate request to the solver to check the validity of the constraint model. If there are no incompatibilities in the new constraint model, the feature with their constraints can be added in the location selected by RuVa.

IV. VALIDATION

In order to evaluate the utility of our runtime variability solution combine with the FaMa tool suite, we came up with the following research questions:

- **RQ1.** Which is the efficacy of the RuVa algorithm in adding new features?
- **RQ2.** Which is the scalability and performance of the RuVa algorithm?
- **RQ3.** How effective are runtime variability mechanisms to be used in systems that require adaptation?

The rationale for **RQ1** is to test to what extent the RuVa algorithm can include more features dynamically in the right place in a feature model, which reduces the human effort to redesign a feature model manually. The rationale for **RQ2** is to uncover the limits to adding features dynamically in large feature models. As some systems are real-time, adding features in a few seconds might be critical while in another types of systems the time factor is not a stringent requirement. Large feature models, typically found in complex software product line approaches (e.g. the automotive domain) exist,

[1] https://choco-solver.org/

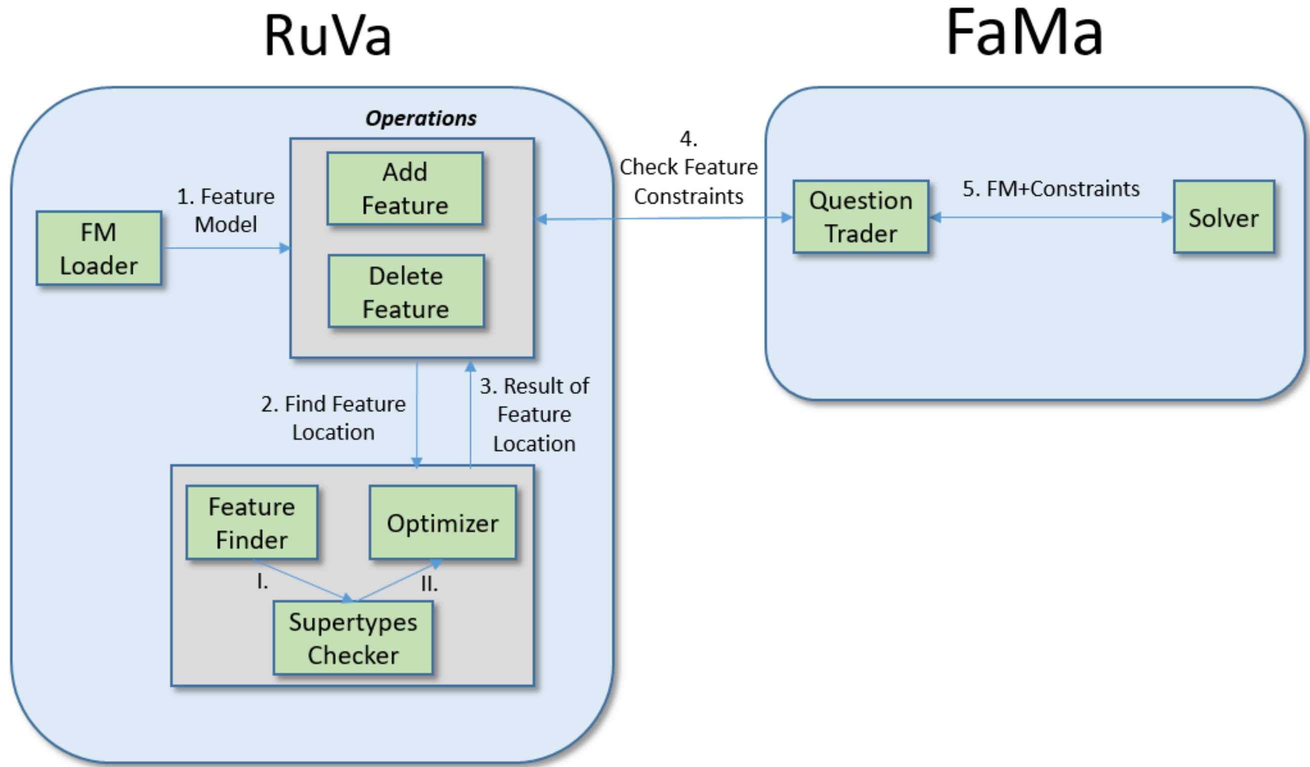

FIGURE 2. Integration and data exchange between RuVa and FaMa tool suite.

and adding new features could be a complex task to redesign the feature model, Finally, the rationale for **RQ3** is to test the effectiveness adding features "on the fly" in a real system and consuming fewer resources, which somehow validates the utility of the proposal.

Consequently, We tested our solution in two different ways. First, we run simulations to answer RQ1 and RQ2 and we used test cases generated automatically, Second, we used a robot to check if adding or updating a feature in the robot's feature model generated by a context change can automatically update the feature model. We explain the validation process in the following subsections.

TABLE 1. Two datasets containing 10 features each with different super-types and logical relationships.

SET 10-1			SET 10-2		
FX	**Logical**	**ST**	**FX**	**Logical**	**ST**
FX1	AND	ST2,ST4	FX1	OR	ST4
FX2	OR	ST1,ST3	FX2	OR	ST5,ST4
FX3	OR	ST2	FX3	OR	ST7
FX4	XOR	ST1	FX4	XOR	ST3
FX5	AND	ST2	FX5	OR	ST4,ST3
FX6	AND	ST1,ST5	FX6	AND	ST1,ST2
FX7	AND	ST4,ST5	FX7	XOR	ST3,ST2
FX8	OR	ST2,ST3	FX8	AND	ST1,ST6
FX9	OR	ST4	FX9	AND	ST10,ST7
FX10	XOR	ST5,ST4	FX10	XOR	ST9,ST3

A. DATASETS

In order to test the efficacy and scalability of the RuVa algorithm as an answer to RQ1 and RQ2, we created different datasets and used sample feature models with a mix of supertypes to test the validity of the proposed solutions. First, we used a sample feature model of 50 features generated randomly with BeTTy and we included five super-types in different branches. Features and super-types are anonymized, so they don't belong to a real feature model but are valid enough to test the efficacy of RuVa inserting features automatically. The sample initial feature model is shown in Figure 3.

Figure 1 shows an example where a new feature "fx" could be added into an existing variability model comparing the different super-types. The algorithm shows three possible locations and finally is placed in location "2" under feature "f2"

As an example of the datasets used, we show in Figure 1 two different datasets (i.e. SET 10-1 and 10-2) containing 10 features to be inserted each one and with a mix of several super-types and logical relationships for each feature. Both datasets were generated randomly. In each sub-table, the field "FX" indicates a generic feature to be inserted, the field "Logical" indicates the logical relationship the feature to be inserted requires, and the field "ST" shows the generic super-types for each feature in the dataset.

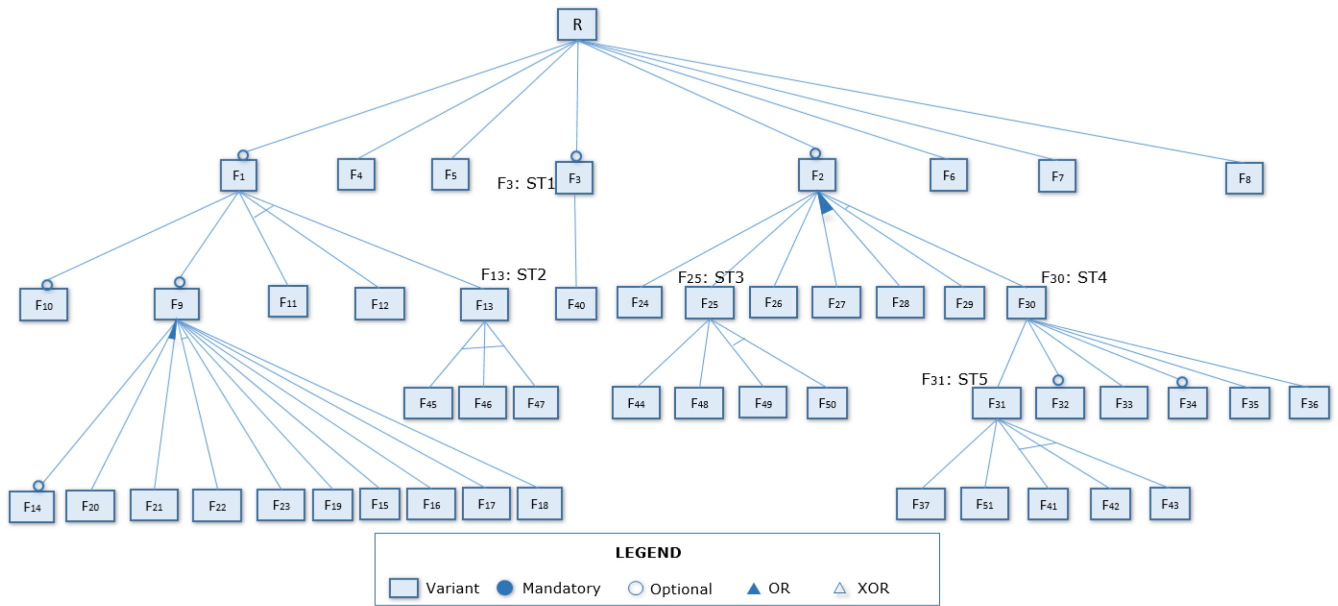

FIGURE 3. Initial feature model containing 50 features and some super-types.

TABLE 2. Time and valid insertions for different datasets.

Dataset	Time (s)	Insertions	Failures	Success rate (%)
1-1	1,03	1	0	100
1-2	0,99	1	0	100
3-1	1,98	2	1	66,7
3-2	0,96	1	2	33,3
5-1	1,80	2	3	40,0
5-2	2,65	3	2	60,0
10-1	7,63	8	2	80,0
10-2	6,89	8	2	80,0

B. EFFICACY AND SCALABILITY RESULTS

We simulated the efficacy and scalability of RuVa using sample feature models and data sets generated randomly with BeTTy and RuVa. We used a Dell Optiplex 7040 Intel Core i7-6700T with a CPU of 2.8GHz and 16 Gigabytes of RAM memory. Regarding the efficacy of the results (**RQ1**) inserting datasets of 10 ten features in the sample feature model, the results where the new features have been added according to their super-types are shown in Figure 4.

With respect to the time employed by RuVa inserting datasets of 1, 3, 5, and 10 features randomly, the results are shown in Table 2.

After we simulated 100 times the insertion of 1, 3, 5, and 10 features and the results we got are shown in Table 3. In the table, we can observe the typical ranges of time to insert the different data sets according to the success rates and the average time needed to insert 1, 3, 5, and 10 features in the sample feature model. In general, the success rates are good and we didn't observe anomalies in finding a good location for the new features, but these results may vary depending on: (i) the topology of each feature model, (ii) the distribution of the super-types, and (iii) the super-types, logical relationships and constraints of the features to be added. Also, please note that the average time to insert a new feature is almost stable, and only when we insert 10 features increased a bit.

TABLE 3. Time abd success rates inserting 1, 3, 5 and 10 features.

Dataset	Time range (s)	Avg. time (s)	Avg. success rate (%)
1 feature	[0,688-1,183]	0,621	76
3 features	[0,770-2,787]	0,678	90
5 features	[1,363-3,631]	0,628	90
10 features	[4,314-9,359]	0,741	92

Finally, the scalability results (**RQ2**) in feature models ranging between 1.000 and 5.000 features are shown in Figure 5. As we can observe, there is a reasonable increment in the time required to find a good location for the new features but the results are doable for non-critical systems, as the reconfiguration can be performed in the system in less time than that needed to redesign the feature model. Also, this time could vary if we add more super-types, the size of the constraint model is huge, or if the topology of the feature model is different.

C. A ROBOT CASE STUDY

In order to answer **RQ3** we used a real robot that can be configured at runtime to evaluate the capabilities of the runtime variability algorithm. Variability in robotics has been suggested in previous works for modeling the context features of a robot [40] and more recently in approaches managing the functional variability of robots implemented using a domain-specific language (i.e. Robot Perception Specification Language) [41] and cardinality-based feature modeling

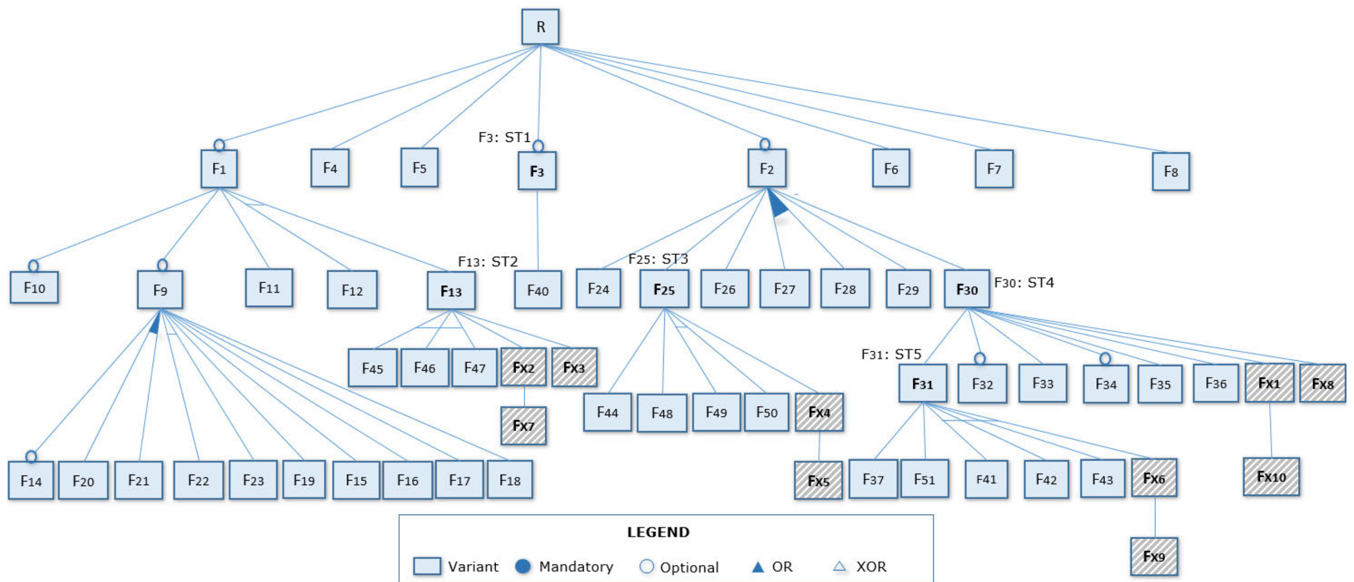

FIGURE 4. Results of new features added by RuVa.

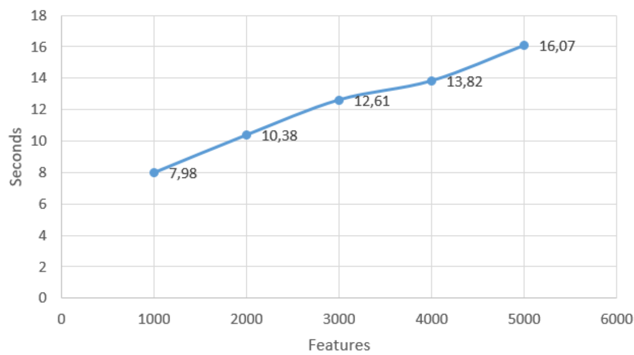

FIGURE 5. Scalability results inserting features in large feature models.

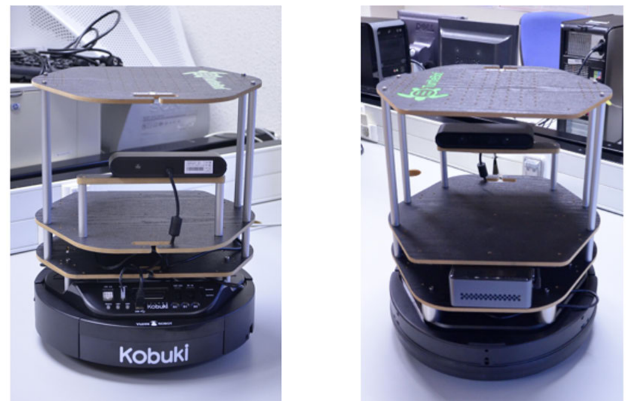

FIGURE 6. TurtleBot robot.

approaches [42]. Other works [43] model the variability of robots to simulate quality properties at runtime. Other works like [29] investigate the role of runtime variability mechanisms to design the variability of the robot behavior that changes according to contextual information and monitor the quality of service (QoS) of the robot performance. In our case, we advance previous works as we reconfigure the variability model of a robot at runtime after a context change. The description of the robot and the feature model is as follows.

1) TurtleBot ROBOT AND FEATURE MODEL

We have used a TurtleBot 2 kobuki robot developed by the Korean company Yujin Robotics in collaboration with Willow Garage. This robot is made up of an Intel NUC computer equipped with an Intel i5 processor with 8 GB of RAM and a 2D/3D Orbbec Astra distance sensor. The robot has Ubuntu 20.04 and the ROS Noetic version installed. The software of the TurtleBot encompasses several functionalities. First, the navigation of the robot can be implemented using *motion planning* algorithms that attempt to find a path without obstacles between two points on a map. Depending on the way we represent the environment to create the map, we can use different algorithms [44] such as *Geometric* based on spatial coordinates which define collision-free areas, or *Topological* which represents an area as a set of interconnected nodes. The motion planning problem resides in the selection of the most suitable graph search algorithm (e.f. AMCL based on an adaptive version of the Monte Carlo localization algorithm [45] which is used by the TurtleBot when it selects Geometric navigation). The TurtleBot has an auto-localization feature which can be also based on *Geometry* or *Topological* in a similar vein to the motion planning feature.

Another important functionality consists in the obstacle detection facility, as it uses an algorithm to follow the environment and detect the surrounding objects, TurtleBot uses a plethora of sensors such as surface detection, object detection,

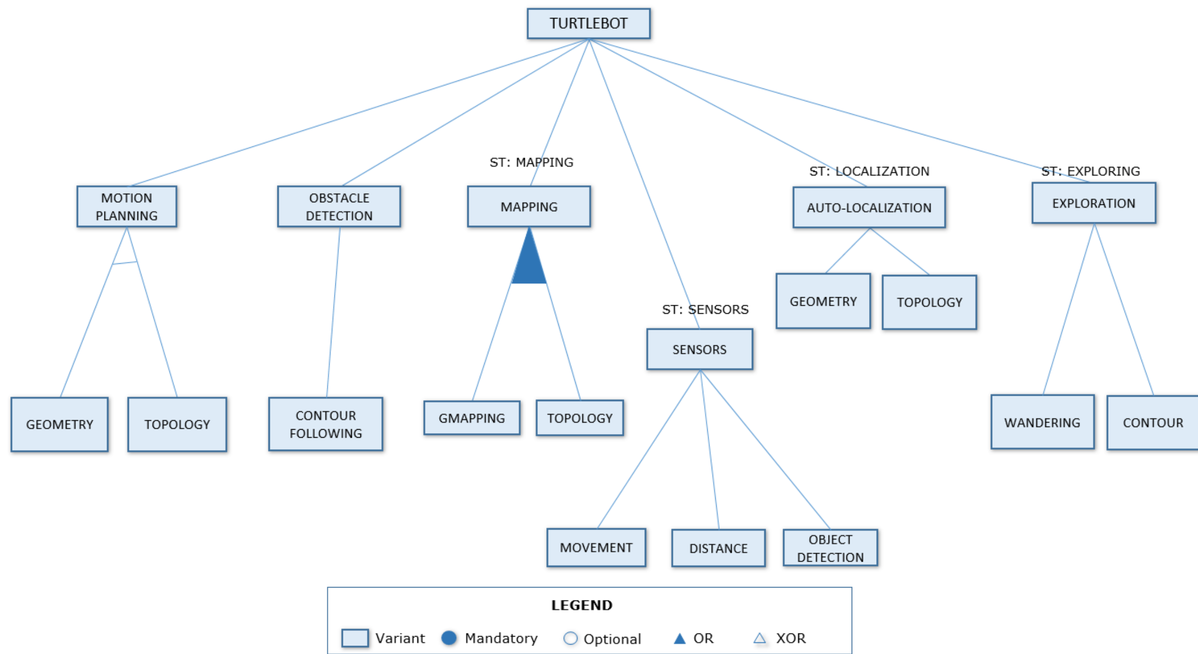

FIGURE 7. **Variability model of the Turtlebot.**

computing the distance to objects, or movement sensors to detect if there are other robots or objects moving in the scene and in which direction. Therefore, the robot can react to objects moving close to it. The robot poses a mapping functionality to create the maps and the TurtleBot implements a geometric map named *GMapping* and a *Topology* one.

Additionally, the *exploration* functionality has two different modes: a *wandering* mode where the robot builds the map according to the data sensed, while the *Contour* following behavior builds the map using the contour of the objects and walls that finds during the movement. Figure 6 shows the Turtlebot robot used in the case study.

The analysis of the TurtleBot features was performed by the last two co-authors and the first two designed the feature model based on the identification of the functional features of the Turtlebot, their properties, and their variable options, such as shown in Figure 7.

The list of cross-tree constraints (i.e. requires and excludes between features) we identified are the following ones:

Wandering REQUIRES Obstacle Detection
Mapping REQUIRES Auto-Localization
Motion Planning REQUIRES Mapping
Mapping REQUIRES Sensors
Obstacle Detection REQUIRES Sensors
Geometry EXCLUDES Topology
Geometric mapping EXCLUDES Topological mapping
Wandering EXCLUDES Contour
GMapping EXCLUDES Topology

2) INTEGRATION BETWEEN TurtleBot ROBOT RuVa

In order to perform the reconfiguration of the feature model at runtime adding new functionality, we had to integrate the TurtleBot with Ruva and agreed on a common way to exchange the data. To solve this challenge we used a communication model using the HTTP protocol and GET requests, so the robot can send the data to a PHP server that sends the data of the new feature to RuVa. The format of the GET requests is as follows:

```
http://domain/ruva.php?featureModel=<name-model> &newFeature=<n-feature>&st=<supertype>
```

The *featureModel* parameter identifies the name of the feature model used by RuVa, while *newFeature* indicates the name of the new feature being added. The *st* variable indicates the super-type of the feature to be added.

3) CASE STUDY AND MAPS

For the case study with the Turtlebot we used the ground floor of one of the buildings of Rey Juan Carlos University consisting of two different areas separated by markers readable by the robot. We used the FloorPlanner[2] software to depict the map, such as we show in Figure 8.

The Turtlebot generates the maps on the fly as long as it moves along the corridor. We used RViz,[3] a 3D visualization tool for ROS (i.e. Robot Operating System) to show the maps built by the robot. Figure 9 shows the robot moving along the corridor in Zone 1 and the map generated by the Turtlebot.

After some time, the robot changes to Zone 2 and builds the map of the new zone as shown in Figure 10.

[2]http://floorplanner.com
[3]http://wiki.ros.org/rviz

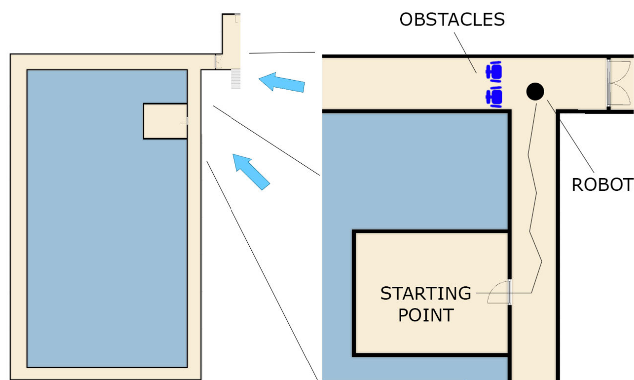

FIGURE 8. Floor map consisting in two different areas where the robot can move.

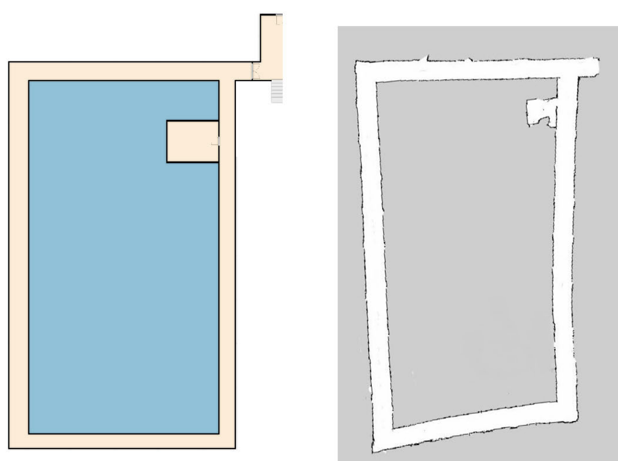

FIGURE 9. Maps built by the Turtlebot and shown using Rviz.

4) RESULTS RECONFIGURING THE TurtleBot

Once the Turtlebot changes from Zone 1 to Zone 2 and reconfigures itself building the map of the new zone, it establishes a communication with the server to RuVa sending the data of the new feature consisting in a new map. If RuVa processes the request satisfactorily, the new feature is added to the variability model according to the feature super-type. The resultant variability model shown in Figure 11 includes the

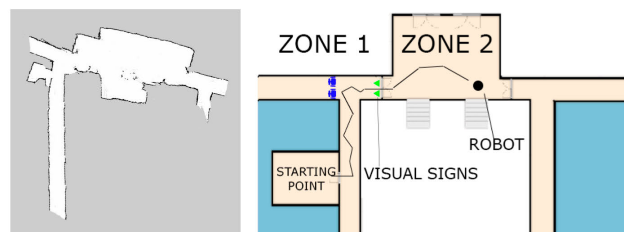

FIGURE 10. Map from the Zone 2 generated by the robot after crossing the markers.

new added feature *Octomap* which is displayed in orange. We run the case study two times and we measured the time spent by the robot to communicate the context changes to RuVa. The average time required by the TurtleBot was between 1 and 1.2 seconds, including the time to transmit the request to RuVa and the time to reconfigure the feature model. Consequently, in those cases where a new feature is added, we communicate the change to the runtime variability algorithm in order to reconfigure the variability model dynamically and without human intervention.

V. FINDINGS

Our results and lessons learned to show the following findings. First, the runtime variability algorithm behaves adequately adding new features dynamically on behalf of the super-types defined. A proper selection of super-types for representative branches in real feature models facilitates assessing software designers to place features in a suitable location. Additionally, the algorithm suggests the most suitable branch according to the number of compatible super-types and the number of children features avoiding overload in excess of the same branch. Also, checking the constraints on the fly using a solver like Choco helps to detect incompatible configurations in feature models as this task is typically performed in ''off-line'' mode and not during runtime.

Second, the algorithm proved to be highly efficient in most cases including in large feature models. The selection of feature models generated randomly minimized the bias to create ''ad-hoc'' feature models for the simulation.

Third, the use of large feature models in the performance and scalability tests demonstrated that the time required to insert different features keeps small. As redrawing the feature model is not a critical task, the values can be considered more than acceptable in all cases.

Fourth, the case study using a real system like a robot proves the applicability of the proposed solution for autonomous and context-aware systems that require some kind of reconfiguration. In this research, we integrated successfully our solution with a TurtleBot in order to redraw a feature model at runtime based on the context changes provided by the robot. As the feature model of the robot is small, the time employed to redraw the model was only

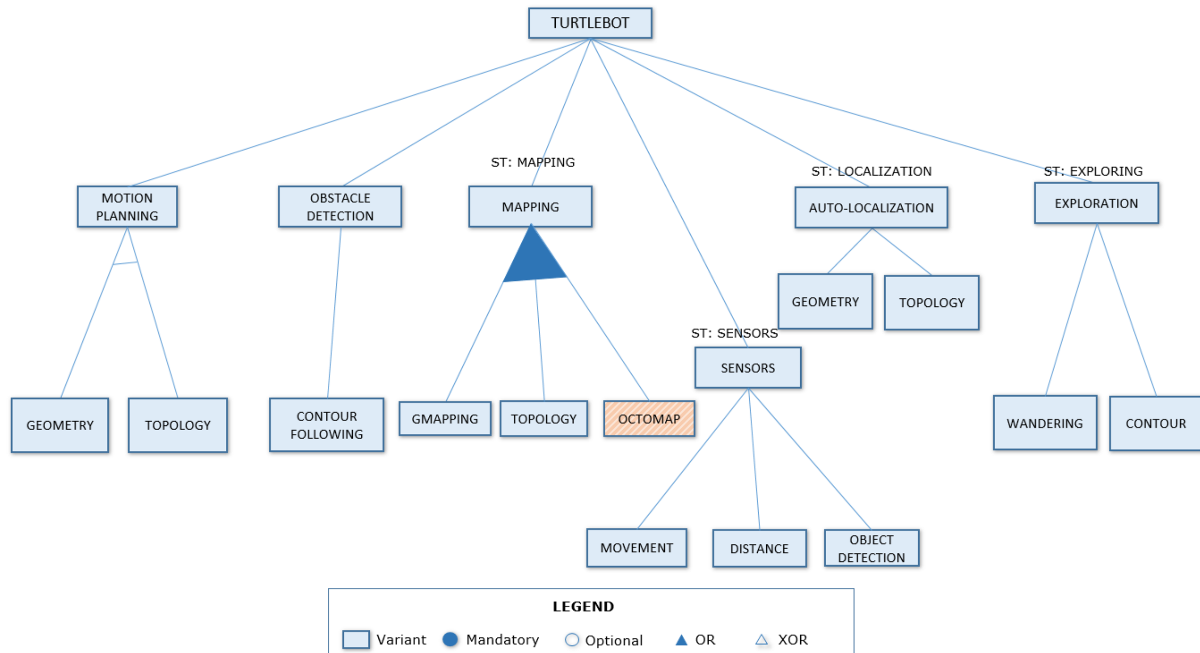

FIGURE 11. Variability model reconfigured after the context change where the new feature Octomap is added.

around 1 second including the time (i,e. 300 milliseconds) to send the data to RuVa.

Lastly, regarding the generalizability of the results to other situations settings, and measures, we can argue that based on the size of the feature models used and the time needed to insert features, our algorithm can handle larger feature models. Only for very critical domains, we should consider smaller feature models but in general terms, the time required to insert 1 feature is on average less than 1 second. This is not a problem when a context-aware system reconfigures itself and sends the data of the new feature to RuVa. However, on the other way around we need to measure the time once a new feature is inserted, and how much time a system needs to reconfigure dynamically. We can argue the proposed solution is valid for the majority of context-aware systems and maybe, only very critical systems may require additional estimations. As the proposed feature models can include context and non-context features we can say that the majority of today's systems that include variability to customize their system's variants and demand runtime reconfiguration can benefit from our solution.

VI. LIMITATIONS

Although the proposed solution was proven satisfactory, there are some limitations to our work. One limitation refers to the definition and location of the super-types. This is a human task that may influence our results as a different number and selection of these super-types may lead to different ways to add a feature or to more unsuccessful results. In those cases where a feature cannot be added automatically human intervention is necessary. In the case some intervention is needed, this doesn't mean the result provided by a human placing a new feature should be different from the automatic solution. If a branch of the variability model doesn't have a proper super-type caused by a modeling mistake or because the software designer forgot to include it, this could require a human intervention to fix that issue, but it doesn't necessarily mean the results provided by the algorithm should be different.

Another limitation is that in our case study we didn't perform the experiment the other way around, that is when a designer adds a feature and the robot reconfigures itself activating the new feature or deactivating an existing functionality. In this case, the reconfiguration time is important because a context-aware system must react in a short time. This could be exacerbated in critical systems where responses might be needed in terms of milliseconds.

VII. CONCLUSION AND FUTURE RESEARCH

The conclusions of this research highlight the importance of dynamic variability algorithms to reconfigure variability models on the fly according to varying context changes. As runtime variability solutions are central for Dynamic Software Product Line approaches, our novel approach solves the traditional gap between off-line solvers and runtime mechanisms required by dynamic variability solutions and can be used in the scope of dynamic software product line approaches for reconfiguring systems. The new optimization mechanism implemented in RuVa defines the most suitable location for new features based on their super-types and also balances the number of new features per branch.

Additionally, the adaptation of both data formats between FaMa solver and the runtime variability algorithm as well as the connection between the Robot and the algorithm was key to providing an integrated solution that can modify the structural variability at runtime. Our approach can help to restructure large feature models and reduce the burden of the human effort in redesigning variants.

The quantitative results from the simulations run exhibit good time rates when inserting new features even for large variability models. The main advantage is that we can reduce the burden of the designer by avoiding a manual design process in feature models for context-aware systems that require periodical reconfiguration. Another advantage related to context-aware systems is that systems don't need to store all the functionality in memory and can download new features on-demand autonomously or assisted by an operator. Although the main limitations have been discussed in the previous section, we have to remark that one disadvantage could come from the need to add features for which the designer doesn't have previous domain knowledge and this may require the intervention of a software modeler. We can support dynamically certain unpredictable features but not those that cannot be classified in any of the initial super-types defined for a given system.

As a recommendation for practical implementation of the solution, we suggest connecting the robot to a cloud robotics [46] system in order to download new functionality on the fly when new features must be added or activated dynamically. Therefore, those robots with memory limitations don't need to store all the functionality in the system memory. Another recommendation in case we don't want to use the choco solver to ensure the validity of the constraint model is to use a different solver that could be integrated with RuVa, such as those mentioned in [47]. Finally, the communication between CPS systems and a server supporting RuVa can be handled by standard TCP/IP and HTTP protocols but we could implement RuVa as a service stored in the same cloud robotics solutions to reduce some communication overhead. In the case of updating features in different domains (e.g. autonomous cars), we should stick to the protocols and technologies belonging to that domain.

In addition, we plan to extend our approach to other cyber-physical systems and context-aware systems (e.g. Smart Cities, IoT, smart vehicles) that demand some kind of runtime reconfiguration of their system options.

ACKNOWLEDGMENT

The authors would like to thank David Benavides and José A. Galindo University of Seville, Spain, for their support in integrating our solution with the FaMa tool suite and the sample feature models generated with BeTTy.

REFERENCES

[1] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Engineering self-adaptive systems through feedback loops," in *Software Engineering for Self-Adaptive Systems*, vol. 5525. Berlin, Germany: Springer, 2009, pp. 48–70.

[2] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and analyzing MAPE-K feedback loops for self-adaptation, in *Proc. 10th IEEE/ACM Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst. (SEAMS)*, May 2015, pp. 13–23.

[3] D. Brugali, "Runtime reconfiguration of robot control systems: A ROS-based case study," in *Proc. 4th IEEE Int. Conf. Robot. Comput. (IRC)*, Nov. 2020, pp. 256–262.

[4] C. H. Corbato, D. Bozhinoski, M. G. Oviedo, G. van der Hoorn, N. H. Garcia, H. Deshpande, J. Tjerngren, and A. Wasowski, "MROS: Runtime adaptation for robot control architectures," 2020, *arXiv:2010.09145*.

[5] A. Binch, G. P. Das, J. P. Fentanes, and M. Hanheide, "Context dependant iterative parameter optimisation for robust robot navigation," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, May 2020, pp. 3937–3943.

[6] M. M. Ng, K. M. Ko, Y. N. Park, and Y. B. Leau, "Adaptive path finding algorithm in dynamic environment for warehouse robot," *Neural Comput. Appl.*, vol. 32, pp. 13155–13171, Sep. 2020.

[7] R. Capilla, J. Bosch, and K.-C. Kang, *Systems and Software Variability Management—Concepts, Tools and Experiences*. Berlin, Germany: Springer, 2013.

[8] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic software product lines," *Computer*, vol. 41, no. 4, pp. 93–95, Apr. 2008.

[9] N. Bencomo, S. Hallsteinsen, and E. S. De Almeida, "A view of the dynamic software product line landscape," *Computer*, vol. 45, no. 10, pp. 36–41, Oct. 2012.

[10] N. Abbas and J. Andersson, "Harnessing variability in product-lines of self-adaptive software systems," in *Proc. 19th Int. Conf. Softw. Product Line (SPLC)*, Jul. 2015, pp. 191–200.

[11] R. Capilla, J. Bosch, P. Trinidad, A. R. Cortés, and M. Hinchey, "An overview of dynamic software product line architectures and techniques: Observations from research and industry," *J. Syst. Softw.*, vol. 91, pp. 3–23, May 2014.

[12] H. Hartmann and T. Trew, "Using feature diagrams with context variability to model multiple product lines for software supply chains," in *Proc. 12th Int. Softw. Product Line Conf. (SPLC)*, Sep. 2008, pp. 12–21.

[13] R. Capilla, O. Ortiz, and M. Hinchey, "Context variability for context-aware systems," *Computer*, vol. 47, no. 2, pp. 85–87, Feb. 2014.

[14] K. Saller, M. Lochau, and I. Reimund, "Context-aware DSPLs: Model-based runtime adaptation for resource-constrained systems," in *Proc. 17th Int. Softw. Product Line Conf. Co-Located Workshops (SPLC)*, 2013, pp. 106–113.

[15] R. Capilla and J. Bosch, "The promise and challenge of runtime variability," *Computer*, vol. 44, no. 12, pp. 93–95, Dec. 2011.

[16] R. Capilla and J. Bosch, "Dynamic variability management supporting operational modes of a power plant product line," in *Proc. 10th Int. Workshop Variability Modelling Softw.-Intensive Syst.*, P. I. Schaefer, V. Alves, and E. S. de Almeida, Eds., 2016, pp. 49–56.

[17] A. Helleboogh, D. Weyns, K. Schmid, T. Holvoet, K. Schelfthout, and W. Van Betsbrugge, "Adding variants on-the-fly: Modeling meta-variability in dynamic software product lines," in *Proc. 3rd Int. Workshop Dyn. Softw. Product Lines (DSPL)*, 2009, pp. 18–27.

[18] R. Froschauer, A. Zoitl, and P. Grunbacher, "Development and adaptation of IEC 61499 automation and control applications with runtime variability models," in *Proc. 7th IEEE Int. Conf. Ind. Informat.*, Jun. 2009, pp. 905–910.

[19] C. Cetina, O. Haugen, X. Zhang, F. Fleurey, and V. Pelechano, "Strategies for variability transformation at run-time," in *Proc. 13th Int. Conf. (SPLC)*, vol. 446, 2009, pp. 61–70.

[20] C. Cetina, P. Giner, J. Fons, and V. Pelechano, "Prototyping dynamic software product lines to evaluate run-time reconfigurations," *Sci. Comput. Program.*, vol. 78, no. 12, pp. 2399–2413, Dec. 2013.

[21] L. Arcega, J. Font, O. Haugen, and C. Cetina, "Achieving knowledge evolution in dynamic software product lines," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, Mar. 2016, pp. 505–516.

[22] N. Gámez and L. Fuentes, "Architectural evolution of FamiWare using cardinality-based feature models," *Inf. Softw. Technol.*, vol. 55, no. 3, pp. 563–580, Mar. 2013.

[23] O. Ortiz, A. B. García, R. Capilla, J. Bosch, and M. Hinchey, "Runtime variability for dynamic reconfiguration in wireless sensor network product lines," in *Proc. 16th Int. Softw. Product Line Conf. (SPLC)*, E. S. de Almeida, C. Schwanninger, and D. Benavides, Eds., 2012, pp. 143–150.

[24] M. L. Mouronte, O. Ortiz, A. B. García, and R. Capilla, "Using dynamic software variability to manage wireless sensor and actuator networks," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2013, pp. 1171–1174.
[25] N. Gámez, L. Fuentes, and J. M. Troya, "Creating self-adapting mobile systems with dynamic software product lines," *IEEE Softw.*, vol. 32, no. 2, pp. 105–112, Mar. 2015.
[26] L. Baresi and C. Quinton, "Dynamically evolving the structural variability of dynamic software product lines," in *Proc. IEEE/ACM 10th Int. Symp. Softw. Eng. Adapt. Self-Manage. Syst. (SEAMS)*, May 2015, pp. 57–63.
[27] N. Taing, T. Springer, N. Cardozo, and A. Schill, "A dynamic instance binding mechanism supporting run-time variability of role-based software systems," in *Proc. Companion Proc. 15th Int. Conf. Modularity*, Mar. 2016, pp. 137–142.
[28] N. Cardozo, W. De Meuter, K. Mens, S. González, and P. Y. Orban, "Features on demand," in *Proc. 8th Int. Workshop Variability Modelling Softw. Intensive Syst. (VaMoS)*, 2014, pp. 18:1–18:8.
[29] A. Romero-Garcés, R. S. De Freitas, R. Marfil, C. Vicente-Chicote, J. Martínez, J. F. Inglés-Romero, and A. Bandera, "QoS metrics-in-the-loop for endowing runtime self-adaptation to robotic software architectures," *Multimedia Tools Appl.*, vol. 81, no. 3, pp. 3603–3628, Jan. 2022.
[30] J. Bürdek, S. Lity, M. Lochau, M. Berens, U. Goltz, and A. Schürr, "Staged configuration of dynamic software product lines with complex binding time constraints," in *Proc. 8th Int. Workshop Variability Modelling Softw. Intensive Syst. (VaMoS)*, 2014, pp. 16:1–16:8.
[31] I. S. Santos, L. S. Rocha, P. A. S. Neto, and R. M. C. Andrade, "Model verification of dynamic software product lines," in *Proc. 30th Brazilian Symp. Softw. Eng. (SBES)*, 2016, pp. 113–122.
[32] G. Sousa, W. Rudametkin, and L. Duchien, "Extending dynamic software product lines with temporal constraints," in *Proc. IEEE/ACM 12th Int. Symp. Softw. Eng. Adapt. Self-Manage. Syst. (SEAMS@ICSE)*, May 2017, pp. 129–139.
[33] A. Felfernig, R. Walter, J. A. Galindo, D. Benavides, S. P. Erdeniz, M. Atas, and S. Reiterer, "Anytime diagnosis for reconfiguration," *J. Intell. Inf. Syst.*, vol. 51, no. 1, pp. 161–182, Aug. 2018.
[34] H. Göttmann, L. Luthmann, M. Lochau, and A. Schürr, "Real-time-aware reconfiguration decisions for dynamic software product lines," in *Proc. 24th ACM Int. Syst. Softw. Product Line Conf.*, 2020, pp. 13:1–13:11.
[35] I. Ayala, A. V. Papadopoulos, M. Amor, and L. Fuentes, "ProDSPL: Proactive self-adaptation based on dynamic software product lines," *J. Syst. Softw.*, vol. 175, May 2021, Art. no. 110909.
[36] E. B. D. Santos, R. M. C. Andrade, and I. D. S. Santos, "Runtime testing of context-aware variability in adaptive systems," *Inf. Softw. Technol.*, vol. 131, Mar. 2021, Art. no. 106482.
[37] R. Capilla, A. Valdezate, and F. J. Díaz, "A runtime variability mechanism based on supertypes," in *Proc. IEEE 1st Int. Workshops Found. Appl. Self Syst. (FAS*W)*, Sep. 2016, pp. 6–11.
[38] J. A. Galindo, D. Benavides, P. Trinidad, and A.-M. Gutiérrez-Fernández, and A. Ruiz-Cortés, "Automated analysis of feature models: Quo vadis?" *Computing*, vol. 101, no. 5, pp. 387–433, 2019.
[39] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Inf. Syst.*, vol. 35, no. 6, pp. 615–636, Sep. 2010.
[40] D. Brugali, R. Capilla, and M. Hinchey, "Dynamic variability meets robotics," *Computer*, vol. 48, no. 12, pp. 94–97, Dec. 2015.
[41] D. Brugali and N. Hochgeschwender, "Managing the functional variability of robotic perception systems," in *Proc. 1st IEEE Int. Conf. Robot. Comput. (IRC)*, Apr. 2017, pp. 277–283.
[42] S. García, D. Strüber, D. Brugali, A. Di Fava, P. Schillinger, P. Pelliccione, and T. Berger, "Variability modeling of service robots: Experiences and challenges," in *Proc. 13th Int. Workshop Variability Modelling Softw.-Intensive Syst. (VAMOS)*, D. Weyns and G. Perrouin, Eds., 2019, pp. 8:1–8:6.
[43] D. Brugali, R. Capilla, R. Mirandola, and C. Trubiani, "Model-based development of QoS-aware reconfigurable autonomous robotic systems," in *Proc. 2nd IEEE Int. Conf. Robot. Comput. (IRC)*, Jan. 2018, pp. 129–136.
[44] R. Barber, J. Crespo, C. Gómez, A. C. Hernámdez, and M. Galli, "Mobile robot navigation in indoor environments: Geometric, topological, and semantic navigation," in *Applications of Mobile Robots*. London, U.K.: IntechOpen, 2018. [Online]. Available: https://www.intechopen.com/chapters/63790, doi: 10.5772/intechopen.79842.
[45] W. P. N. D. Reis, G. J. D. Silva, O. M. Junior, and K. C. T. Vivaldini, "An extended analysis on tuning the parameters of adaptive Monte Carlo localization ROS package in an automated guided vehicle," *Int. J. Adv. Manuf. Technol.*, vol. 117, nos. 5–6, pp. 1975–1995, Nov. 2021.
[46] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 398–409, Jan. 2015.
[47] C. Sundermann, T. Thüm, and I. Schaefer, "Evaluating #sat solvers on industrial feature models," in *Proc. 14th Int. Work. Conf. Variability Modelling Softw.-Intensive Syst.*, 2020, pp. 3:1–3:9.

ALEJANDRO VALDEZATE received the bachelor's degree in computer science and the M.Sc. degree from Rey Juan Carlos University, where he is currently pursuing the Ph.D. degree (industrial). He has a professional experience of more than 20 years in several Spanish software companies in the areas of DevOps, software testing/QA, and cloud services. His research interests include product line engineering and dynamic variability solutions.

RAFAEL CAPILLA (Senior Member, IEEE) received the B.Sc. degree in computer science from the Universidad de Sevilla and the Ph.D. degree in computer science from the Universidad Rey Juan Carlos of Madrid, Spain, in 2004. He is currently an Associate Professor of software engineering at Universidad Rey Juan Carlos. He is a coauthor of more than 100 peer-reviewed conference and journal papers. He is a Co-Editor of the book *Systems and Software Variability Methods, Concepts and Tools* (Springer, 2013). His research interests include software architecture and architectural knowledge, software sustainability, technical debt, software product line engineering, and industry 4.0. He is a Regular Reviewer of several well-known journals, such as *Journal of Systems and Software*, *Information and Software Technology*, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, and IEEE SOFTWARE. He is a guest co-editor of more than ten special issues.

JONATHAN CRESPO received the B.S. degree in computer engineering from the Universidad Autónoma de Madrid, in 2007, and the master's degree in robotics and automation and the Ph.D. degree in electrical engineering, electronics and automation from the University Carlos III de Madrid, in 2012 and 2017, respectively. He is currently a Ph.D. Assistant Professor at University Rey Juan Carlos, Madrid. His research interests include cognitive robots, machine learning, planners, and navigation systems on mobile robots.

RAMÓN BARBER (Senior Member, IEEE) received the B.Sc. degree in industrial engineering from the Polytechnic University of Madrid, in 1995, and the Ph.D. degree in industrial technologies from University Carlos III, in 2000. He is currently an Associate Professor with the System Engineering and Automation Department, University Carlos III of Madrid, Spain. His research interests include mobile robotics, including perception of the environment, environment modeling, planning, localization and navigation tasks, considering geometrical, and topological and semantic representations. He is a member of the International Federation of Automatic Control (IFAC).