Science of Computer Programming 240 (2025) 103221

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

A Configurable Software Model of a Self-Adaptive Robotic System

Juliane Päßler [a,*], Maurice H. ter Beek [b], Ferruccio Damiani [c], Einar Broch Johnsen [a], S. Lizeth Tapia Tarifa [a]

[a] University of Oslo, Gaustadalléen 23B, Oslo, NO-0373, Norway
[b] CNR–ISTI, Via Giuseppe Moruzzi 1, Pisa, I-56124, Italy
[c] University of Turin, Corso Svizzera 185, Torino, I-10149, Italy

ARTICLE INFO

Keywords:
Self-adaptive systems
Dynamic software product lines
Probabilistic model checking
ProFeat

ABSTRACT

Self-adaptation, meant to increase reliability, is a crucial feature of cyber-physical systems operating in uncertain physical environments. Ensuring safety properties of self-adaptive systems is of utter importance, especially when operating in remote environments where communication with a human operator is limited, like under water or in space. This paper presents a software model that allows the analysis of one such self-adaptive system, a configurable underwater robot used for pipeline inspection, by means of the probabilistic model checker ProFeat. Furthermore, it shows that the configurable software model is easily extensible to further, possibly more complex use cases and analyses.

Code Metadata.

Code metadata description	
Current code version	v1.1.2
Permanent link to code/repository used for this code version	https://github.com/remaro-network/auv_profeat/releases/tag/SCP-2024
Permanent link to reproduce capsule	https://doi.org/10.5281/zenodo.13946884
Legal code license	Apache License 2.0
Coder versioning system used	git
Software code languages, tools and services used	PRISM input and property language, ProFeat, iFM23 virtual machine
Compilation requirements, operating environments and dependencies	Linux, Windows, MacOS (Intel for capsule)
If available, link to developer documentation/manual	https://github.com/remaro-network/auv_profeat/blob/scp-ifm_artifact/README.md
Support email for questions	julipas@uio.no

1. Motivation and significance

Self-Adaptive Systems (SASs) often operate in dangerous and dynamic environments where human supervision is limited or impossible, like under water or in space. Therefore, it is important to ensure that safety properties are maintained by the system throughout system operation. Once in operation, SASs are frequently reconfigured, which often means switching between different

* Corresponding author.
E-mail addresses: julipas@uio.no (J. Päßler), maurice.terbeek@isti.cnr.it (M.H. ter Beek), ferruccio.damiani@unito.it (F. Damiani), einarj@uio.no (E.B. Johnsen), sltarifa@uio.no (S.L. Tapia Tarifa).

https://doi.org/10.1016/j.scico.2024.103221
Received 29 February 2024; Received in revised form 9 August 2024; Accepted 14 October 2024

Available online 17 October 2024

system configurations during runtime. The analysis of all these different yet partially redundant configurations separately is a tedious, time-consuming, and error-prone task, especially because this ignores the changes between configurations.

In Päßler et al. [20], we showed the advantages of modelling such an SAS as a family of systems, where each family member corresponds to a possible configuration, which allows for family-based modelling and analysis as a means to combat redundancy [25]. To do so, we used formal models and tools from the field of Software Product Lines (SPL) [2]. We also used the fact that SASs can be implemented using a two-layered approach, decomposing the system into a *managed* and a *managing* subsystem [15,27], with the managed subsystem implementing the domain concerns (e.g., navigating a robot to a specific position) and the managing subsystem implementing the adaptation logic (e.g., reconfiguring due to changing environmental conditions). This separation of concerns is catered for by ProFeat [9], a tool for probabilistic family-based model checking. ProFeat provides a means to simultaneously analyse, in one single run, a family of models, each corresponding to a valid configuration.

This paper contributes a configurable software model of a self-adaptive robotic system, namely an Autonomous Underwater Vehicle (AUV) used to search for and follow a pipeline located on a seabed. Furthermore, the paper illustrates how to perform analyses of such models with ProFeat, and how to modify and extend the model. The model has been used for a case study, presented in Päßler et al. [20,21], and is inspired by the exemplar SUAVE [22]. In contrast to Päßler et al. [20], this paper does not detail the software model. Instead, it shows with the software model, how an existing framework for modelling and analysing family-based systems can be used for SAS research. Furthermore, it shows how the software model can be extended for further, possibly more complex SAS models and analyses.

2. Software description

Our configurable software model is built for analysis with the family-based model checker ProFeat. ProFeat[1] is a tool that extends the probabilistic model checker PRISM[2] [16] with functionalities such as family models, features, and feature switches, thus enabling family-based modelling and analysis of probabilistic systems in which feature configurations may dynamically change during runtime. ProFeat translates its input to the input language of PRISM to use PRISM's reasoning engine for probabilistic (family-based) model checking.

Akin to SASs, an input model of ProFeat can be seen as a two-layered model in which the behaviour of a family of systems that differ in their feature configurations, as defined by a *feature model* that specifies the features and their relations and constraints, is specified as *feature modules* (i.e., the 'managed' behavioural model) along with a *feature controller* that can activate and deactivate features at runtime (i.e., the 'managing' behavioural model), thus changing (reconfiguring) system behaviour. Furthermore, possible environments can be specified as separate modules that interact with the modules of the behavioural models.

The software model of the AUV case study in Päßler et al. [21] contains the following ProFeat modules:

- a feature model, defining the functionalities of the AUV as features and specifying their relations and constraints as well as feature-specific costs implemented as *rewards* (e.g., time and energy);
- a probabilistic, feature-guarded model of the managed subsystem (i.e., a probabilistic featured transition system [11]), defining the behaviour of the different configurations of the SAS as well as the possible switches between them (i.e., reconfigurations);
- a probabilistic model of the environment (i.e., water visibility);
- a feature controller, representing the managing subsystem of the SAS, that activates and deactivates features of the feature model during runtime (while satisfying the constraints of the feature model), based on both environmental and internal conditions, thereby enabling and disabling specific configurations and behaviour of the managed subsystem;
- properties concerning expected rewards (a.k.a. costs) and probabilities.

It is also possible to specify more modules that interact with the already existing ones, like, e.g., a model of how the hardware of the managed subsystem fails or a model of the battery consumption of the AUV. The case study's repository[3] contains one such extension by including two sensors for vision (i.e., a camera and a (side-scan) sonar) and a separate hardware module that models how these sensors can fail (permanently) or get blocked (in case of the camera, e.g., due to natural or human waste sticking to it) at runtime, causing the need to switch between vision sensors or abort the mission. During operation, the sonar is preferred for searching, because it can cover a wider area and operate at a higher altitude, whereas the camera is preferred for following and inspecting the pipeline because it is easier to detect faults in the pipeline with the camera.

3. Illustrative examples

To use the software model for analysis with ProFeat, download the iFM 2023 artefact evaluation virtual machine (VM) [18]. In the VM, first open a terminal window. Then download the artefact[4] with the command

[1] https://pchrszon.github.io/profeat/.

[2] https://www.prismmodelchecker.org/manual.

[3] https://github.com/remaro-network/auv_profeat.

[4] We will assume in the following that you saved the file in the home directory. It is also possible to save it in another directory, but then the path to this directory must be used in all commands using paths.

```
wget https://zenodo.org/records/8275533/files/auv_profeat.zip
```

and unzip it with

```
unzip auv_profeat.zip
```

Then run the following.

```
cd ~/prism
./install.sh
```

The artefact contains a file casestudy.profeat with the models of the managed subsystem, the managing subsystem, and the environment, files casestudy.fprops and casestudy_all.fprops with properties to analyse, a folder experiments with files for conducting PRISM experiments, a license, and a README file.

3.1. Running analyses

To run an analysis, navigate to auv_profeat in the terminal. Then execute the following.

```
~/profeat/bin/profeat -t casestudy.profeat casestudy.fprops
~/prism/bin/prism out.prism out.props > out.log
```

The first command translates the ProFeat model and the ProFeat property file to a PRISM model and property file, respectively. The second command uses PRISM to compute the results and saves them in the out.log file. To view the results, open the out.log file which is saved in the auv_profeat folder.

To analyse additional properties to the ones analysed here, these need to be included in the casestudy.fprops file, making sure to include any ProFeat-specific constructs like, e.g., features and variables, within ${...}.

3.2. Understanding the output

The out.log file, which contains the results of the analysis, is structured as follows. After a PRISM header, it specifies the model type, the modules, and the variables of the PRISM file that was automatically translated from the ProFeat file. It then lists the analysed properties. These are slightly different from the properties specified in casestudy.fprops because they have been translated to PRISM properties. For each of the properties, the out.log file includes a paragraph, separated by ––––, with the analysis results. The result of the analysis (Result) can be found at the bottom of the paragraph, preceded by the time that was used for model checking (Time for model checking).

3.3. Changing scenarios

The ProFeat model comes equipped with two different, predefined scenarios, each of them implementing a different behaviour of the environment. To change the scenario, open the file ~/auv_profeat/casestudy.profeat, uncomment the parameters of the desired scenario, and comment the parameters of the other scenario. To create a new scenario, change the values of the parameters min_visib, max_visib, current_prob, and inspect. It is also possible to change the influence that the thruster failures have on the path (i.e., movement) of the AUV by changing infl_tf.

3.4. PRISM experiments

It is possible to use PRISM's functionality of so-called *experiments* to perform the analysis of a parametric property for a predefined parameter range. To use this functionality with the parametric properties defined for this case study, it suffices to do as described below. The files for the PRISM experiments for the two predefined scenarios can be found in the folder ~/auv_profeat/experiments as scenario1.prism and scenario2.prism. The necessary property file, which contains the properties used for the experiments, is experiments.props in the same folder.

Open the PRISM GUI via the command

```
~/prism/bin/xprism
```

Open the model file of one of the two scenarios by going to Model –> Open model and selecting scenario1.prism or scenario2.prism. Parse and build the model by pressing F2 and F3, respectively. To load the properties, go to the Properties Tab in the lower left corner. Open the properties list by going to Properties –> Open properties list and select experiments.props. The experiments will use a variable named k for the number of time steps. Declare this variable by double-clicking in the empty Constants area, which will create an entry named C0. Change the name of the entry from C0 to k.

To run an experiment, click one of the properties and press F7. In the dialogue that opens, first decide the desired range of parameters, i.e., how many time steps to consider. For example, to create a graph from the first property, select the radio button for Range (deselecting Single Variable), and fill in 0 for Start, 80 for End and 1 for Step. Then click on Okay, give the graph a name, and either print it to an already existing graph or to a new one.

It is also possible to inspect the values that were calculated for the graph. To do so, in the Experiments pane, first select (left-click) the property whose results to inspect, then right-click and select View results from the context menu. This will enable us to determine after how many time steps the probability for the respective property to be satisfied is above a certain threshold. In this way, we for instance determined that the probability of reaching a safe state from an unsafe state is above 0.95 after 5 time steps in both scenarios.

For more information about PRISM experiments, including how to run them from the command line, consult the PRISM manual.[5]

3.5. Model extensions

The artefact can be modified and extended in different ways, some ideas are as follows.

- Explore new scenarios;
- Analyse different properties;
- Change the probabilities of the transitions;
- Introduce different environmental or internal parameters that can trigger adaptation (i.e., feature changes by the feature controller);
- Include new modules (e.g., introducing failures in the hardware or modelling battery consumption) that synchronise with the already existing ones and with the feature controller;
- Make the feature model richer by including more functionalities of the AUV that can be changed during runtime;
- Include new states in the AUV's feature module (e.g., a further task that the AUV has to perform).

The latter four suggestions require extending the AUV's feature modules, feature controller, or both.

4. Impact

The software model of the AUV case study presented in this paper is relevant for both new and existing questions in research on SASs and on dynamic SPLs as well as in industry. Below, we present the research areas in which the software model can be used and we discuss directions that highlight its relevance.

- SASs can be realised by *internal self-adaptation*, which embeds the adaptation logic in the system itself through exception handling or fault-tolerance mechanisms, or *external self-adaptation*, which separates the adaptation logic from the application logic through an external feedback loop [15,24,27]. Internal self-adaptation has been criticised for poor maintainability and scalability. Our software model implements external self-adaptation by separation of concerns between the application logic (the managed subsystem) and the adaptation logic (the managing subsystem) of the SAS as proposed in [15]. Thus, the software model provides an example of how to model and analyse use cases with this separation of concerns. The separation of concerns makes it easy to reuse the software model since it caters for modifications or extended use cases. Our software model also exemplifies how to improve scalability of the models by modelling all configurations and reconfigurations of an SAS in one modular model, enabling the analysis of all configurations and reconfigurations in a single run while maintaining the separation of concerns between the application and the adaptation logic.
- Dynamic SPL research distinguishes between *bounded adaptivity*, which models context variation that is anticipated at design time, and *open adaptivity*, which models context variation that is not planned at design time and requires model extension [7]. Our software model implements bounded adaptivity (the feature controller), for which dynamic SPLs have been advocated as a means to constrain the evolution of SASs, thus enabling the assessment of important properties of an SAS prior to its implementation [4].
- Dynamic SPLs have been proposed to manage runtime reconfiguration for self-adaptive robots [8,13]. While appealing, this is still considered an unsolved challenge [12] since managing runtime reconfiguration for SASs is in general very difficult and "there is a need to validate the proposals, either in an industrial environment or in different test cases, expanding the application areas" [1]. In fact, a recent literature review [3] on testing, validation, and verification of robotic and autonomous systems does not discuss any research that uses family-based analysis techniques for SASs as exemplified with our software model. Therefore, our software model can provide an example of how to use dynamic SPLs for modelling self-adaptive robots and for using family-based analysis techniques to analyse the robot's configurations and reconfigurations.
- Kentaro Yoshimura, chief researcher at Hitachi, recently addressed the SPL community in his keynote address entitled "The 20-year journey of SPLE in Hitachi and the next" at the 2023 SPL Conference (SPLC 2023) [17]. In his keynote, he presented the use of dynamic SPLs for autonomous robotic systems as a new industrial challenge. He said that the dynamicity is in the runtime behaviour of the autonomous robots that need to adapt and reconfigure based on input perceived from the environment without continuous human guidance. Our software model responds to this challenge by capturing the uncertainties of the environment in a separate probabilistic model that interacts with the behavioural models of the SAS.

[5] https://www.prismmodelchecker.org/manual/RunningPRISM/Experiments).

5. Conclusions

This paper contributes a configurable software model of a dynamic SPL, reflecting the self-adaptive AUV introduced in [20]. The model is part of a growing body of family-based models including, e.g., probabilistic variants of the well-known SPL benchmarks called the Body Sensor Network (BSN) SPL [23], based on the BSN from [14], and the Elevator SPL [10], based on the Lift system from [19], both of which were analysed with ProFeat by Chrszon et al. [9] and the latter also with QFLan by ter Beek et al. [5] and its original, non-probabilistic version also with the well-known mCRL2 model-checking toolset[6] in [6]. QFLan is a software tool for the modelling and analysis of highly reconfigurable systems, including dynamic SPLs. These software models are all publicly available.[7,8,9] While ProFeat provides tool support for family-based (quantitative) analysis of dynamic SPLs with probabilistic behaviour through probabilistic model checking, QFLan [26] provides tool support for statistical model checking.

6. Future plans

In the future, it would be interesting to analyse larger dynamic SPL models of SASs, which might require resorting to statistical model-checking techniques that yield statistical approximations by probabilistic simulations, thus trading 100% precision for scalability.

CRediT authorship contribution statement

Juliane Päßler: Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis. **Maurice H. ter Beek:** Writing – original draft, Writing – review & editing, Conceptualization, Visualization, Funding acquisition. **Ferruccio Damiani:** Writing – review & editing, Funding acquisition. **Einar Broch Johnsen:** Writing – review & editing, Supervision, Project administration, Funding acquisition. **S. Lizeth Tapia Tarifa:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We thank Clemens Dubslaff for explaining everything we always wanted to know about ProFeat. This work was supported by the European Union's Horizon 2020 Framework Programme through the MSCA network REMARO (Grant Agreement No 956200), by the Italian project NODES (which has received funding from the MUR – M4C2 1.5 of PNRR with grant agreement no. ECS00000036) and by the Italian MUR PRIN 2020TL3X8X project T-LADIES (Typeful Language Adaptation for Dynamic, Interacting and Evolving Systems).

References

[1] O. Aguayo, S. Sepúlveda, Variability management in dynamic software product lines for self-adaptive systems—A systematic mapping, Appl. Sci. 12 (2022), https://doi.org/10.3390/app122010240.
[2] S. Apel, D.S. Batory, C. Kästner, G. Saake, Feature-Oriented Software Product Lines: Concepts and Implementation, Springer, 2013.
[3] H. Araujo, M.R. Mousavi, M. Varshosaz, Testing, validation, and verification of robotic and autonomous systems: a systematic review, ACM Trans. Softw. Eng. Methodol. 32 (2023) 51:1–51:61, https://doi.org/10.1145/3542945.
[4] L. Baresi, Self-adaptive systems, services, and product lines, in: Proceedings of the 18th International Software Product Line Conference (SPLC 2014), ACM, 2014, pp. 2–4.
[5] M.H. ter Beek, A. Legay, A. Lluch Lafuente, A. Vandin, A framework for quantitative modeling and analysis of highly (re)configurable systems, IEEE Trans. Softw. Eng. 46 (2020) 321–345, https://doi.org/10.1109/TSE.2018.2853726.
[6] M.H. ter Beek, S. van Loo, E.P. de Vink, T.A.C. Willemse, Family-based SPL model checking using parity games with variability, in: H. Wehrheim, J. Cabot (Eds.), Proceedings of the 23rd International Conference on Fundamental Approaches to Software Engineering (FASE 2020), in: LNCS, vol. 12076, Springer, 2020, pp. 245–265.
[7] N. Bencomo, S.O. Hallsteinsen, E.S. de Almeida, A view of the dynamic software product line landscape, IEEE Comput. 45 (2012) 36–41, https://doi.org/10.1109/MC.2012.292.
[8] D. Brugali, R. Capilla, M. Hinchey, Dynamic variability meets robotics, IEEE Comput. 48 (2015) 94–97, https://doi.org/10.1109/MC.2015.354.
[9] P. Chrszon, C. Dubslaff, S. Klüppelholz, C. Baier, ProFeat: feature-oriented engineering for family-based probabilistic model checking, Form. Asp. Comput. 30 (2018) 45–75, https://doi.org/10.1007/s00165-017-0432-4.
[10] A. Classen, P. Heymans, P.Y. Schobbens, A. Legay, Symbolic model checking of software product lines, in: Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011), ACM, 2011, pp. 321–330.

[6] https://mcrl2.org/.
[7] https://wwwtcs.inf.tu-dresden.de/ALGI/PUB/ProFeat/.
[8] https://github.com/qflanTeam/QFLan/wiki.
[9] https://github.com/SjefvanLoo/VariabilityParityGames.

[11] M. Cordy, A. Classen, P. Heymans, A. Legay, P.Y. Schobbens, Model checking adaptive software with featured transition systems, in: J. Cámara, R. de Lemos, C. Ghezzi, A. Lopes (Eds.), Assurances for Self-Adaptive Systems: Principles, Models, and Techniques, in: LNCS, vol. 7740, Springer, 2013, pp. 1–29.
[12] S. García, D. Strüber, D. Brugali, A. Di Fava, P. Schillinger, P. Pelliccione, T. Berger, Variability modeling of service robots: experiences and challenges, in: Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS 2019), ACM, 2019, pp. 8:1–8:6.
[13] L. Gherardi, N. Hochgeschwender, RRA: models and tools for robotics run-time adaptation, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2015), IEEE, 2015, pp. 1777–1784.
[14] Y. Hao, R. Foster, Wireless body sensor networks for health-monitoring applications, Physiol. Meas. 29 (2008) R27–R56, https://doi.org/10.1088/0967-3334/29/11/R01.
[15] J.O. Kephart, D.M. Chess, The vision of autonomic computing, IEEE Comput. 36 (2003) 41–50, https://doi.org/10.1109/MC.2003.1160055.
[16] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: verification of probabilistic real-time systems, in: G. Gopalakrishnan, S. Qadeer (Eds.), Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011), in: LNCS, vol. 6806, Springer, 2011, pp. 585–591.
[17] H. Ogawa, K. Yoshimura, The 20-year journey of SPLE in Hitachi and the next, in: Proceedings of the 27th ACM International Systems and Software Product Line Conference (SPLC 2023), vol. 1, ACM, 2023, p. xix, https://doi.org/10.1145/3579027.
[18] M. Osama, A.L. Lamprecht, iFM 2023 artifact evaluation VM, Zenodo, https://doi.org/10.5281/zenodo.7782241, 2023.
[19] M. Plath, M. Ryan, Feature integration using a feature construct, Sci. Comput. Program. 41 (2001) 53–84, https://doi.org/10.1016/S0167-6423(00)00018-6.
[20] J. Päßler, M.H. ter Beek, F. Damiani, S.L. Tapia Tarifa, E.B. Johnsen, Formal modelling and analysis of a self-adaptive robotic system, in: P. Herber, A. Wijs (Eds.), Proceedings of the 18th International Conference on Integrated Formal Methods (iFM 2023), in: LNCS, vol. 14300, Springer, 2023, pp. 343–363.
[21] J. Päßler, M.H. ter Beek, F. Damiani, S.L. Tapia Tarifa, E.B. Johnsen, Formal Modelling and Analysis of a Self-Adaptive Robotic System (Artifact), Zenodo, https://doi.org/10.5281/zenodo.8275533, 2023.
[22] G. Rezende Silva, J. Päßler, J. Zwanepol, E. Alberts, S.L. Tapia Tarifa, I. Gerostathopoulos, E.B. Johnsen, C. Hernández Corbato, SUAVE: an exemplar for self-adaptive underwater vehicles, in: Proceedings of the 18th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2023), IEEE, 2023, pp. 181–187.
[23] G.N. Rodrigues, V. Alves, V. Nunes, A. Lanna, M. Cordy, P.Y. Schobbens, A.M. Sharifloo, A. Legay, Modeling and verification for probabilistic properties in software product lines, in: Proceedings of the 16th International Symposium on High Assurance Systems Engineering (HASE 2015), IEEE, 2015, pp. 173–180.
[24] M. Salehie, L. Tahvildari, Self-adaptive software: landscape and research challenges, ACM Trans. Auton. Adapt. Syst. 4 (2009) 14:1–14:42, https://doi.org/10.1145/1516533.1516538.
[25] T. Thüm, S. Apel, C. Kästner, I. Schaefer, G. Saake, A classification and survey of analysis strategies for software product lines, ACM Comput. Surv. 47 (2014) 6:1–6:45, https://doi.org/10.1145/2580950.
[26] A. Vandin, M.H. ter Beek, A. Legay, A. Lluch Lafuente, QFLan: a tool for the quantitative analysis of highly reconfigurable systems, in: K. Havelund, J. Peleska, B. Roscoe, E. de Vink (Eds.), Proceedings of the 22nd International Symposium on Formal Methods (FM 2018), in: LNCS, vol. 10951, Springer, 2018, pp. 329–337.
[27] D. Weyns, An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective, John Wiley & Sons, 2020.