Abstract: An Architecture Model for DSPL Engineering

Edilton Lima dos Santos
Faculty of Computer Science
Namur University (UNamur)
Namur, Belgium
edilton.limados@unamur.be

Ivan do Carmo Machado
Dept. Computer Science.
Federal University of Bahia (UFBA)
Salvador, Brazil
ivan.machado@ufba.br

I. Reference

This work was initially published as part of the 19th IEEE International Conference on Information Reuse and Integration (IRI 2018) in Salt Lake City, Utah, USA [1].

II. DSPL Architecture Model

Dynamic Software Product Lines (DSPL) engineering enables designing more dynamic software architectures and building more adaptable software to handle autonomous decision-making, according to varying conditions [2]. It also emphasizes variability analysis and design at development time and variability binding and reconfiguration at runtime [3]. A DSPL strategy needs to answer two key questions: *(i) When to adapt?* and *(ii) How to adapt?* [4].

In this work [1], we propose an architecture model for DSPL engineering based on the MAPE-K model [5], capable of answering such key questions. Based on the MAPE-K activities, the first question could be answered with the support of the *Monitoring* and *Analysis* activities, and the latter through the *Planning* and *Execution* ones [4]. Finally, *Knowledge* provides the necessary data to support the system reconfiguration process. The architecture model encompasses a set of features, as follows: *(i) DSPL Core*, *(ii) Feature Area*, and *(iii) Context Sensors*.

The *DSPL Core* is responsible for managing the *Adaptation Policies* and *Verification* principles, defined for each feature. The DSPL Core comprises the following features: *(a) Listener* - It is responsible for gathering and processing environmental context data that is relevant to the adaptation process by using the *Context Sensors*; *(b) Presentation Layer* - It is responsible for displaying data from the features that communicate with the *Context Sensors* or other *Features* that compose the system, according to the *Change Plan*; *(c) Manager* - It is responsible for analyzing and planning the adaptations. To execute an adaptation plan, the *Manager* uses the *Downloader*, *Installer*, and *Loader*. These allow to adapt the running system and get the desired behavior by activating/deactivating DSPL Features. They also enable building context sensor data visualizations in the *Presentation Layer* according to the active features; *(d) Loader* - It is responsible for loading and unloading a particular feature and its dependencies from the *Feature Area*, as requested by the *Manager*; *(e) Downloader* - It downloads every new feature, based on the new sensors connected to the system or the installation of user features; *(f) Installer* - It is responsible for installing new features in the *Feature Area*. The *Feature Area* is responsible for storing features and *Knowledge*. *Knowledge* contains Feature settings and associated constraint policies, which are used by the *Manager* to properly manage adaptations and reconfigurations. The *Context Sensors* is composed of *sensors* whose purpose is to get data from the environment and send them to the *Listener*.

To assess the proposed model, we implemented a DSPL in the Smart Home Systems domain, using OSGi[1] and the MQTT[2] communication broker. We aimed to understand *when* and *how* the architecture could support the identification of a given context and find the appropriate sequence of actions and the mechanisms that enable the adaptation. We observed the system's capability of recognizing new contexts and promoting the required adaptations. We also evaluated whether the proposed hardware/software infrastructure was capable of supporting the adaptation needs of each context.

III. Current Work

This work initiated the PhD research of the first author (who started in September 2019) at the University of Namur. This PhD research explores the links between MAPE-K models and testability, e.g., how to adapt the test methodology and tools to dynamically evolving features.

References

[1] E. Santos and I. Machado, "Towards an architecture model for dynamic software product lines engineering," in *IRI*. IEEE, 2018, pp. 31–38.
[2] R. Capilla, J. Bosch, P. Trinidad, A. Ruiz-Cortés, and M. Hinchey, "An overview of dynamic software product line architectures and techniques: Observations from research and industry," *JSS*, vol. 91, pp. 3–23, 2014.
[3] L. Shen, X. Peng, J. Liu, and W. Zhao, "Towards feature-oriented variability reconfiguration in dynamic software product lines," in *ICSR*. Springer, 2011, pp. 52–68.
[4] N. Bencomo, J. Lee, and S. Hallsteinsen, "How dynamic is your dynamic software product line?" in *Dynamic Software Product Lines Workshop*, 2010.
[5] IBM, "An architectural blueprint for autonomic computing," *IBM White Paper*, vol. 31, pp. 1–6, 2006.

This research was partially funded by INES 2.0, CNPq grant 465614/2014-0 and FAPESB grants JCB0060/2016 and BOL2521/2016.

[1] Open Service Gateway Initiative - https://www.osgi.org/
[2] Message Queue Telemetry Transport - http://mqtt.org/