A Multi-platform End User Software Product Line Meta-model for Smart Environments

Vasilios Tzeremes and Hassan Gomaa
Computer Science Dept., George Mason University, Fairfax, Virginia, U.S.A

Keywords: End User Development, Software Product Lines (SPL), Meta-modeling, Variability Modeling, Middleware, Smart Environments, Smart Spaces, Software Product Line Architecture.

Abstract: End User (EU) architectures for smart environments aim to enable end users to create and deploy software applications for their smart spaces. EU Software Product Lines (SPL) extend EU architectures for smart environments with product line support to promote reuse and software application portability. This paper describes a meta-modeling approach for developing EU SPLs for smart environments. We present a meta-model as the basis for developing a framework for creating EU SPLs and deriving EU applications. The meta-model is composed of platform independent and platform specific meta-models. This paper describes in detail both parts of the meta-model and discusses the relationships and mappings between them. This paper also presents the XANA EU SPL framework that was developed using the proposed platform specific meta-model and discusses XANA's product line creation and application derivation process.

1 INTRODUCTION

Smart environments, also called smart spaces, are environments equipped with visual and audio sensing systems, pervasive devices, sensors, and networks that can perceive and react to people, sense on-going human activities and respond to them (Kindberg and Fox, 2002). Several End User (EU) architectures have been proposed to assist end users to create applications for their smart environments. EU architectures act as the middleware between software applications and devices deployed in the smart space while providing friendly user interfaces for end users to create software applications. Team Computing (TeC) (Sousa et al., 2010) and Puzzle (Danado and Paternò, 2012) are examples of EU architectures for smart environments.

Even though EU architectures enable end users to create applications for their spaces, not all smart environments are configured the same way. Furthermore device capabilities vary across different smart environments. This causes end users to have to create similar applications from scratch for different environments. Software Product Line (SPL) methods address software reuse by explicitly analysing and developing the common and variable parts of a family of systems (Gomaa, 2005). However, existing SPL methods target software engineers instead of end users and their processes are rigid. In an end user environment, the process is more agile and end users are not familiar with SPL methods. Furthermore, product derivation in a traditional SPL environment is based on feature selection and products must be compliant with the SPL architecture. End user environments vary and are not guaranteed to match the SPL architecture.

EU SPLs for smart spaces provide a lightweight approach for SPL development while addressing the dynamic nature of these environments. In particular, EU SPLs extend EU architectures to create a family of applications that are then customized for different smart environments (Tzeremes and Gomaa, 2016). Figure 1 shows the EU SPL process. SPL designers create EU SPLs and end users derive applications for their smart spaces. SPL designers are technical end users or domain experts that develop software applications either for personal or commercial purposes. End users are ordinary users that want to create applications for their smart spaces. The XANA EU SPL framework provides an example of tool support for the implementation of EU SPLs.

This research investigates the extension of EU architecture meta-models for supporting the creation of EU SPLs. In detail, this paper describes a meta-modeling approach and framework for creating EU SPLs for smart environments. Our approach provides

Tzeremes, V. and Gomaa, H.
A Multi-platform End User Software Product Line Meta-model for Smart Environments.
DOI: 10.5220/0006003802900297
In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1: ICSOFT-EA*, pages 290-297
ISBN: 978-989-758-194-6

platform independent and platform specific EU SPL modeling support. In the platform independent phase, EU SPL engineers create platform independent models that can be tailored to different EU architectures through an application derivation process. In the platform specific phase, EU SPL engineers create platform specific models that are bound to specific EU platforms. Platform specific models provide an additional capability, since they have access to platform specific functionality that is not available to the platform independent models.

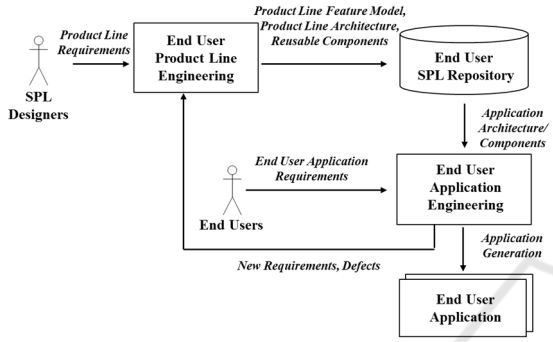

Figure 1: End User Software Product Line Process.

This paper is organized as follows. Section 2 discusses related work that this research builds on. Section 3 describes the overall EU SPL meta-modeling approach for smart environments. Sections 4 and 5 describe in detail the platform specific and platform independent meta-models respectively. Section 6 describes the XANA EU SPL process flows. Finally, section 7 provides conclusions and discusses future work.

2 RELATED WORK

Several middleware architectures have been proposed for implementing smart environments (Whitmore et al., 2015). Some of those initiatives are the ROS (Quigley et al., 2009), JCAF (Bardram, 2005) and the Smart Products (Mühlhäuser, 2008) projects. EU architectures extend middleware architectures by adding end user support. They provide user friendly interfaces for end users to be able to develop programs for their spaces. Some of the most important EU architectures are Puzzle (Danado and Paternò, 2012), PIP (Chin et al., 2010), FedNet (Kawsar et al., 2008) and TeC. This research presents an approach for extending EU architectures for smart spaces with product line concepts to promote reuse and application portability.

Model Driven Architecture (MDA) is a software development framework based on automatic transformations of models (Debnath et al., 2008). MDA separates business and application logic from underlying platform technology, distinguishing the following models: Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM) and code. The Common Variability Language (CVL) adds variability to MDA models. In particular CVL, is a Domain Specific Language (DSL) for modeling variability in models that are based on Meta Object Facility (MOF) standard defined by the Object Management Group (OMG) (Reinhartz-Berger et al., 2014). Our approach is influenced by the PSM, PIM and CVL concepts but was expanded to end user development for smart spaces.

3 OVERVIEW OF THE EU SPL META-MODEL FOR SMART ENVIRONMENTS

There are several common characteristics across EU architectures for smart spaces. For example all event driven EU architectures consist of components that are abstractions of devices, sensors, actuators, application, services etc. and connections between the components to create application logic. There is also significant variability between EU architectures. For example some EU architectures account for user-context, location, temporal relationships while others do not. There is commonality and variability across EU SPLs for smart spaces. For example, a feature implementation of one EU architecture can be significantly different from one architecture to another. To capture the commonality and variability of EU architectures and EU SPLs, we propose the EU SPL meta-model. Figure 2 shows the EU SPL meta-model for smart environments. This meta-model consists of platform independent and platform specific meta-models. The platform independent meta-model is composed of the Platform Independent Product Line (PIPL) and the Platform Independent Product (PIP) meta-models. PIPL captures product line metadata for creating software product lines for smart environments, in particular the product line features and the component architecture that implements each feature. The component architecture describes the smart environment components, connectors and other artefacts that are needed for the feature implementation. The PIP meta-model describes the structure of software applications that can be derived from the PIPL model. To derive PIP

models application engineers select product line features from the PIPL model. The selected features, combined with the application component architecture are used to create the PIP. Both PIPL and PIP models are platform independent models that can be instantiated for different platforms.

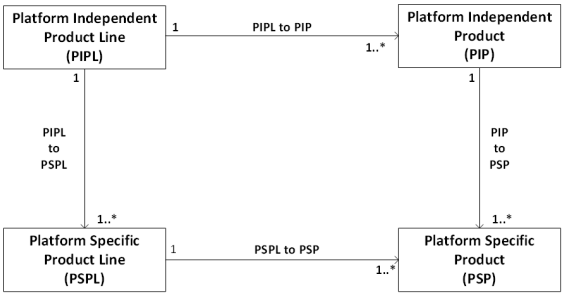

Figure 2: End User SPL Meta-model.

The platform specific meta-model consists of the Platform Specific Product Line (PSPL) and the Platform Specific Product (PSP) meta-models. The PSPL meta-model is used for creating EU SPL models for specific EU applications on specific platforms. Similar to the PIPL meta-model, the PSPL meta-model captures the product line features and their inter-dependencies, in addition to the component architecture that implements each feature. The PSPL meta-model is platform specific. PSPL models are derived from PIPL models. The PSP meta-model captures the application architecture. A shown in Figure 2, PSP models can be derived from PSPL models or alternatively from PIP models.

There is a one-to-many relationship between the platform independent and the platform specific models. For instance, multiple PSPL models for different platforms can be created from the PIPL model. Product line engineers can model platform independent EU SPLs using the PIPL meta-model that can be converted to PSPL models for different platforms. Similarly, many PSP models can be generated from the PIP model. Application engineers can generate different PIP models that can then be converted to PSP models for different platforms.

The PIPL to PIP and PSPL to PSP model relationships are also one-to-many. This implies that several PIP models can be created from one PIPL model. Although multiple PSP models can be derived from one PSPL model, the PSPL and PSP models need to be for the same target platform. For example a PSPL model designed for the TeC EU architecture can generate PSP models that apply only to the TeC platform. The following sections of this paper describe the platform specific and platform independent meta-models.

4 PLATFORM SPECIFIC META-MODELS

This section describes the platform specific meta-models, in particular the PSL and PSPL meta-models for the Team Computing (TeC) EU architecture, before describing how they can be generalized into platform independent meta-models in Section 4. In particular, section 4.1 introduces TeC and presents its application (PSP) meta-model, section 4.2 discusses how we extended the TeC application meta-model to create the TeC PSPL.

4.1 Platform Specific Product for TeC

Team Computing (TeC) is an event driven generic architectural style that enables end users to design and deploy personalized software for their spaces. It provides a diagrammatic language for application creation of a collection of activities that work together to achieve a common goal. TeC applications, also called Teams, have no central control: elements play their roles autonomously and their behavior is emergent (Sousa et al., 2010).

Figure 3 shows the application meta-model for TeC. In detail, the Team Design entity captures information about TeC teams. Team designs can be deployed to zero-or-more locations. For example one Team Design might apply to devices available to the family room of a smart home versus another one that applies to the entire house. A team design is realized by one-or-more Activity Sheets. Activities Sheets are software components, devices, and humans operating in ubiquitous computing environments. Activities Sheets have zero-or-more Inputs and Outputs. Inputs are component interfaces for receiving data. Outputs are also component interfaces but they are used for sending data. Outputs are bound by triggering conditions that when evaluated to true causes the output to be sent. In TeC, device connectivity can be achieved by having outputs from one Activity Sheet sent to inputs of another Activity Sheet. Inputs and Outputs can contain zero-or-more Payloads. Payloads capture data that are in the form of key-value pairs that are sent from Outputs to Inputs. The Activity Connector entity is responsible for capturing the Activity Sheet's connectivity within a Team Design. The Activity Connector is composed by zero-or-more Inputs and Outputs.

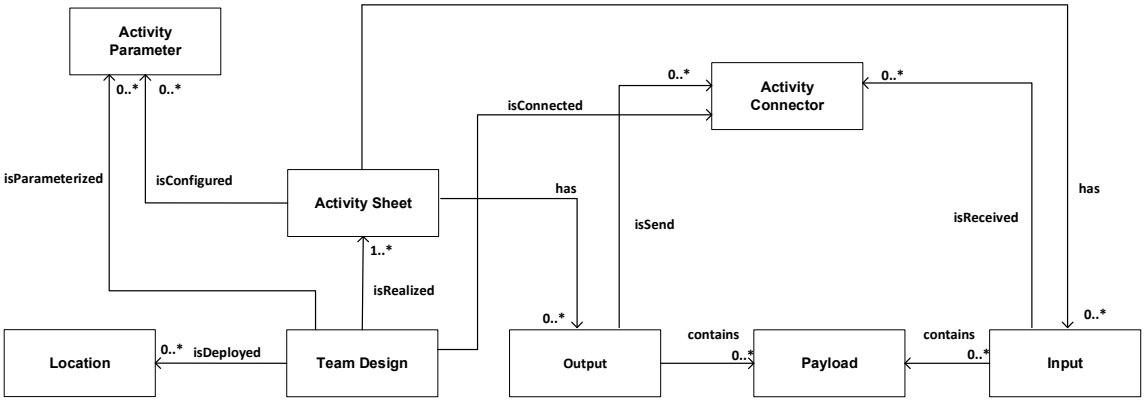

Figure 3: TeC Application Meta-model (PSP).

Figure 4 shows the Team Design of a "Flood Alert" team. The "Flood Alert" Team Design is composed of a flood detector and a Phone TeC Activity Sheets. The flood detector Activity Sheet represent moisture sensors deployed in the environment and the Phone Activity Sheet a house phone that supports landline messaging. The flood detector Activity Sheet has an Output called "alert" that sends flood notifications to the "text" Input of the Phone Activity Sheet. The Activity Connector entity for the Team Design is composed of the "alert" Output and the "text" Input. The "alert" output has a triggering condition that is evaluated to true when the flood detector detects moisture. When moisture is detected, "alert" sends two Payloads to the "text" input. The keys of the payloads are phone_number and message. The Phone Activity Sheet will interpret the phone_number payload value as the number to text and the message payload value as the contents of the message to send. An Activity Sheet can have zero-or-more Activity Parameters. Activity Parameters capture internal parameters of Activity Sheets. An example of an Activity Parameter in the "Flood Alert" example can be moisture threshold values for the flood detector Activity Sheet. When the moisture threshold values are exceeded then the sensor can report moisture.

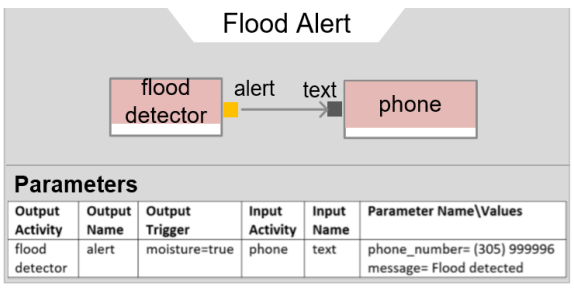

Figure 4: Flood Alert – TeC Team.

4.2 Platform Specific Product Line for TeC

We extended the TeC PSP model with product line support to create the TeC PSPL meta-model shown in Figure 5. The objective of the TeC PSPL meta-model is to be able to derive multiple TeC PSP models from one TeC PSPL model. The TeC PSPL meta-model is composed of the feature and the component meta-models. The feature meta-model is platform independent and describes the relationship of the EU SPL with features and the dependency among features. The component meta-model is specific to the TeC platform and describes the relationships between product line features and the TeC Product Line component architecture that realizes each feature.

As shown in Figure 5, an EU SPL is composed of one or more features. Each feature describes a specific functionality that the EU SPL supports. Features can be common, optional, alternative, default, or parameterized. Common features are features that exist in all products derived from the product line. Optional features are features that can be found in only some products derived from the product line. Alternative features are features that are mutually exclusive. Default features are one of a group of alternative features that the EU SPL designer has pre-selected for product derivation. Parameterized features are features that can be parameterized by end users during application derivation. Features can belong to feature groups. Feature groups can be thought as a set of features that share a common set of constraints. The Feature Dependency entity captures the dependency among features. A feature condition is used to identify whether a given feature is selected or not in a derived architecture.

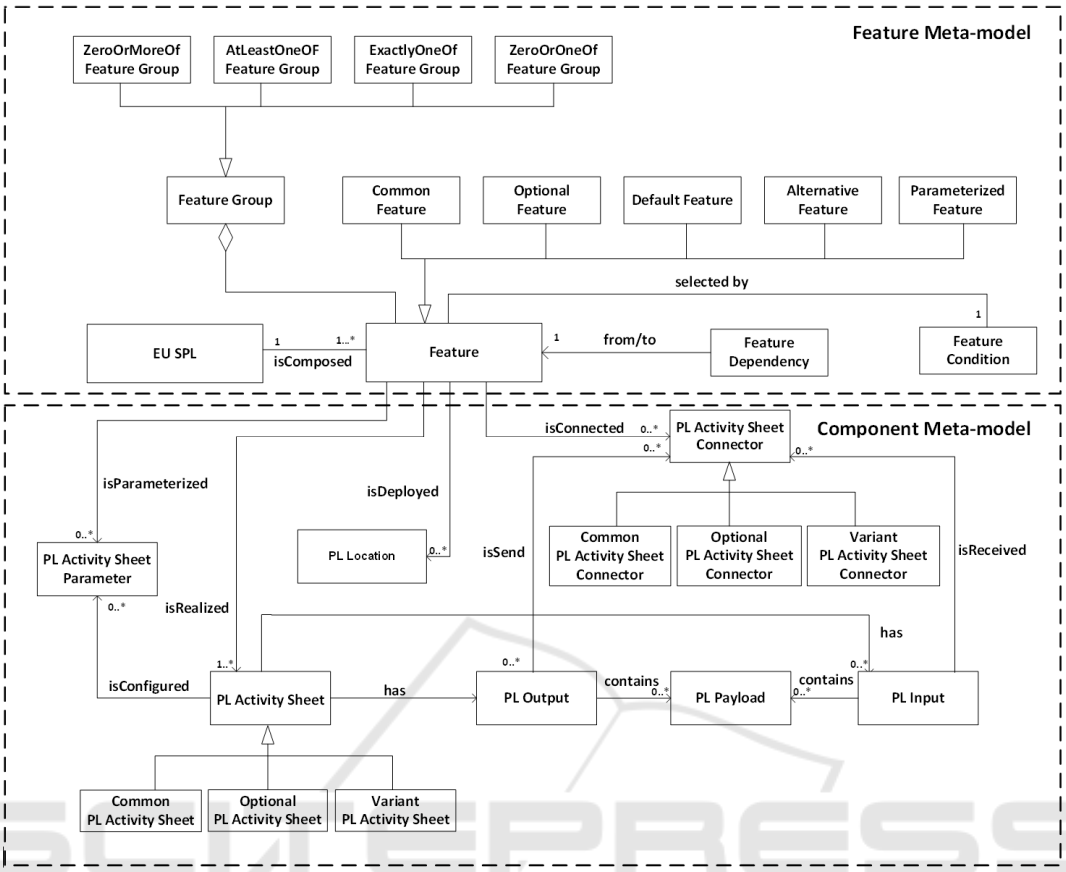

Figure 5: TeC Platform Specific Product Line (PSPL) Meta-model.

The TeC PL component architecture extends the TeC application meta-model with product line support. In particular, EU SPL features are realised by one-or-more PL Activity Sheets and are connected to zero-or-more PL Activity Connectors. PL Activity Sheets can be common, optional or variant. Common PL Activity Sheets are available to all PSPs derived from the PSPL. Optional PL Activity Sheets might be available only to some PSPs. Variant PL Activity Sheets are mutually exclusive PL Activity Sheets. PL Activity Sheets can have zero-or-more PL Inputs and PL Outputs. PL Inputs and PL Outputs can have zero-or-more PL Payloads. PL Activity Connectors can also be common, optional or variant. A feature is parameterized by zero-or-more PL Activity Parameters. Finally, features can be deployed in zero-or-more PL Locations.

The application derivation process from the TeC PSPL to the PSP model involves a set of model conversions. The PSPL component model for each derived feature is converted to the PSP component model and is added to the TeC application architecture. PL Activity Sheets in the PSPL model are converted to Activity Sheets in the PSP model. Similarly, PL Activity Connectors are converted to Activity Connectors, PL Activity Parameters are converted to Activity Parameters and PL Locations are converted to Location entities.

5 PLATFORM INDEPENDENT META-MODELS

We further extended the PSPL and PSP meta-models to create the Platform Independent Product Line (PIPL) and the Platform Independent Product (PIP) meta-models, which do not depend on any particular EU platform. The platform independent models apply to all EU architectures that support component and connector architectures.

5.1 Platform Independent Product Line (PIPL)

Similar to the PSPL, the PIPL meta-model is composed of the feature and the component meta-models. The feature meta-model is the same as the

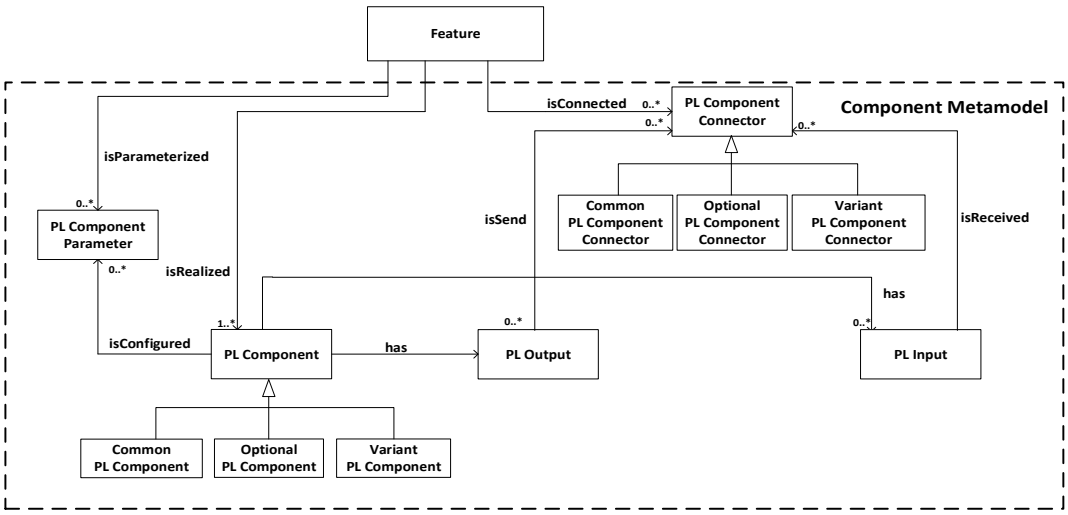

Figure 6: Platform Independent Product Line (PIPL) Meta-model.

PSPL shown on Figure 5. The component meta-model is designed to support common component connector functionality across different EU architectures. Figure 6 shows the PIPL component meta-model. In particular, each feature in the PIPL is realised by one-or-more PL Components, is connected by zero-or-more PL Component Connectors, and is parameterized by zero-or-more PL Component Parameters. PL Components are similar to PL Activity Sheets of the TeC PSPL. PL Components are generic components that represent software and hardware entities that are part of the smart environment. PL Components can be common, optional or variant and they have zero-or-more PL Inputs and PL Outputs. PL Component Connectors indicate how PL Components are connected. PL Component Connectors can be common, optional or variant. PL Component Parameters are used to parameterize PL Components.

5.2 Platform Independent Product (PIP)

The Platform Independent Product (PIP) meta-model describes the structure of models derived from PIPL models. Figure 7 shows the PIP meta-model. End user applications in the PIP meta-model are represented by the Product entity in PIP. Products are members of the product line. The Product is composed of one-or-more Components, is connected by zero-or-more Component Connectors, and is configured by zero-or-more Component Parameters. Components can have zero-or-more inputs to receive input and zero-or-more outputs to send data.

To derive a PIP model from PIPL features the following conversion must occur: 1) Each PL Component that is part of the feature realization must be converted to a Component in the PIP model 2) PL Component Connectors must be converted to Component Connectors and 3) PL Component Parameters must be converted to Component Parameters in the PIP model.

6 XANA EU SPL FRAMEWORK

To validate our approach we used the proposed EU SPL meta-models to implement the XANA EU SPL framework for smart spaces. The XANA framework enables EU SPL designers to create EU SPLs and end users to derive applications for their smart spaces. Figure 8 shows XANA's product line creation and application derivation process flows.

XANA is a platform specific EU SPL framework for TeC. The EU SPL is created using the TeC PSPL meta-model and derived applications (PSPs) are TeC application models that can be deployed to different TeC environments. To validate this research, we implemented the XANA prototype, developed EU SPLs for smart spaces, and deployed derived applications to the TeC Android simulator.

6.1 EU SPL Creation

The uppermost part of Figure 8 shows the EU SPL creation process in XANA. During product line creation, EU SPL designers define the product line features and create the product line architecture. The XANA prototype SPL creation user interface is divided into four sections: 1) Feature Model, 2) Fea-

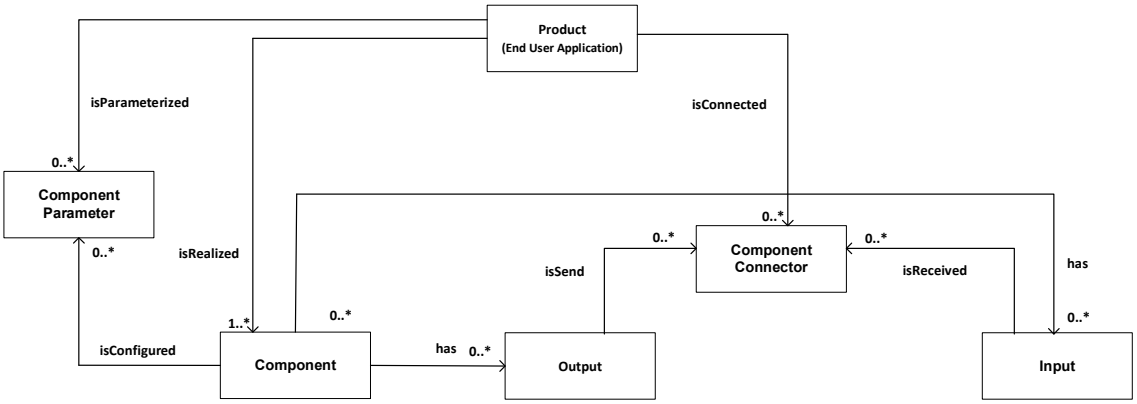

Figure 7: Platform Independent Product (PIP) Meta-model.

ture Component Architecture Editor 3) Component Selector and 4) Feature Parameter table.

The Feature Model organizes product line features in a tree structure. Each feature is decorated with a feature symbol to indicate the feature type. The Feature Component Architecture Editor captures the component and connector architecture that realizes each feature. The component selector section lists the available TeC components that can be used to realize a feature. The Parameter table captures connection details between connected TeC components.

After SPL designers complete creating the product line features they submit the EU SPL to the XANA's SPL Creation subsystem for storage. The SPL Creation subsystem stores the XANA EU SPL visual representation shown on step "1.1" in Figure 8. Then the SPL Creation subsystem transforms the EU SPL visual representation to a Java object structure representing the product line. The Java objects are serialized to JSON objects in the file system for long term storage shown on step "1.2" in Figure 8. Both the Java and JSON representations are based on the TeC PSPL meta-model shown in Figure 5.

6.2 Application Derivation

The bottom part of Figure 8 shows the EU application derivation process in XANA. During application derivation, end users are presented with the end user view of the feature model and the feature parameter table. End users, based on their requirements, select features from the feature model, configure the feature parameter table, and submit their selections to the XANA's Application Derivation subsystem as shown on step "2 Feature Selection" in Figure 8.

The Application Derivation subsystem extracts the component architecture of the selected features from the PSPL shown on step "2.1" in Figure 8 and composes the TeC App (PSP). The TeC App is serialized to JSON in the file system shown on step "2.2" in Figure 8. The Application Derivation subsystem distributes the JSON representation of the TeC App to the target TeC platform shown on step "2.3" in Figure 8. The TeC platform will store the TeC App as shown on step "2.4" in Figure 8 and deploy it to the smart space devices.

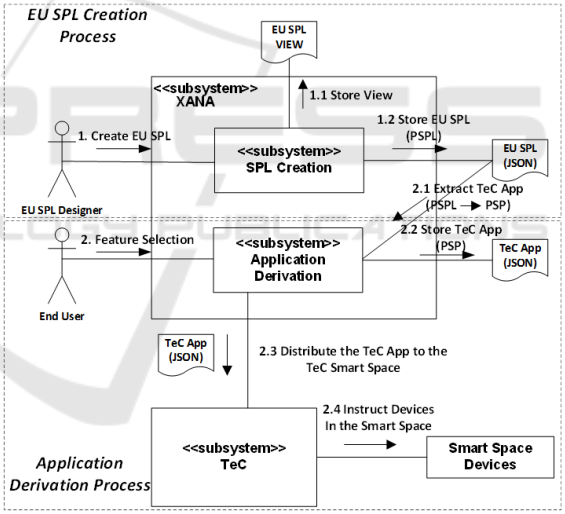

Figure 8: XANA EU SPL Creation and Application Derivation Process flow.

7 CONCLUSIONS

As EU architectures for smart spaces expand, end users will be faced with the challenge of having to develop the same types of applications for different environments. EU SPLs for smart spaces enable end users to derive and port software applications developed for individual spaces. In this paper we presented an EU SPL meta-model for creating end user product lines. The EU SPL meta-model is

composed of platform independent and platform specific meta-models. The platform specific meta-model was discussed in the context of the TeC EU architecture. The platform independent meta-model is a meta-model for creating product lines for EU applications that support component and connector architecture. The paper also presented the XANA EU SPL framework, which is based on the EU SPL platform specific meta-model, that was developed to validate this work.

The benefits of our approach are a) the EU SPL meta-model is used to add product line support to end user architectures defined for smart spaces b) product lines are designed to be platform independent and are adapted for different platforms. Currently we are investigating expanding the platform independent product line meta-model to directly derive applications for different platforms. Furthermore, additional work is needed in the evolution and validation of end user product lines. In particular, processes need to be defined to handle EU SPL evolution that involve new requirements, defect reporting, and product line versioning etc. In addition, a validation framework needs to be investigated for verifying the end user product lines.

ACKNOWLEDGEMENTS

This work is partially supported by the AFOSR award FA9550-16-1-0030.

REFERENCES

Bardram J. E., (2005) The Java Context Awareness Framework – a Service Infrastructure and Programming Framework for Context-aware Applications, *Proc. Third International Conf. on Pervasive Computing*. Springer, Berlin, pp 98–115

Chin J., Callaghan V., Clarke G., (2010) End-user Customization of Intelligent Environments. In: *Nakashima H et al (eds) Handbook of Ambient Intelligence and Smart Environments*. Springer US, Boston, MA, pp 371–407

Danado J., Paternò F., (2012) Puzzle: a visual-based environment for end user development in touch-based mobile phones. In: *Human-Centered Software Engineering. Springer*, pp 199–216.

Debnath N, Leonardi MC, Mauco MV, et al (2008) Improving Model Driven Architecture with Requirements Models. In: *ITNG 2008 5th International Conference* pp 21–26.

Kawsar F, Nakajima T, Fujinami K (2008) Deploy Spontaneously: Supporting End-Users in Building and Enhancing a Smart Home. In: *10th International Conf. in Ubiquitous Computing*. Seoul, pp 282–291

Gomaa, H., Designing Software Product Lines with UML, *Addison-Wesley*, 2005.

Kindberg J, Fox A (2002) System Software for Ubiquitous Computing. *IEEE Pervasive Comput 70–81*.

Mühlhäuser M (2008) Smart Products: An Introduction. In: Mühlhäuser et al (eds) Constructing Ambient Intelligence: *AmI 2007 Workshops*, Revised Papers. Springer Berlin, Heidelberg, pp 158–164.

Quigley M, Conley K, Gerkey BP, et al (2009) ROS: an open-source Robot Operating System. In: *ICRA Workshop on Open Source Software*.

Reinhartz-Berger I, Figl K, Haugen Ø (2014) Model-Driven Engineering Languages and Systems: *17th International Conference, MODELS*, Valencia, Spain. Proceedings. In: Dingel J, Schulte W, Ramos I, et al. (eds). Springer, Cham, pp 501–517.

Sousa JP, Tzeremes V, El Masri A (2010) Space-aware TeC: End-user Development of Safety and Control Systems for Smart Spaces. In: *Systems Man and Cybernetics, IEEE International Conference on*. Istanbul, Turkey, pp 2914–2921.

Tzeremes V, Gomaa H (2016) XANA: An End User Software Product Line Framework for Smart Spaces. In: 2016 *49th Hawaii International Conference on System Sciences*, pp 5831–5840.

Whitmore A, Agarwal A, Xu L (2015) The Internet of Things–A Survey of Topics and Trends. *Inf Syst Front* 17:261–274. doi: 10.1007/s10796-014-9489-2.