Applying End User Software Product Line Engineering for Smart Spaces

Vasilios Tzeremes
Department of Computer Science
George Mason University
Fairfax, VA 22030, USA
vtzereme@gmu.edu

Hassan Gomaa
Department of Computer Science
George Mason University
Fairfax, VA 22030, USA
hgomaa@gmu.edu

Abstract

Smart spaces are physical environments equipped with pervasive technology that sense and react to human activities and changes in the environment. End User Development (EUD) skills vary significantly among end users who want to develop software applications for their smart spaces. This paper presents a systematic approach for adopting reuse in EUD for smart spaces by using Software Product Line (SPL) concepts. End User (EU) SPL designers develop EU SPLs for smart spaces whereas end users derive their individual smart space applications from these SPLs. In particular, this paper presents a systematic approach for EU SPL designers to develop EU SPLs and end users to derive software applications for their spaces, an EUD environment that supports EU SPL development and application derivation, and a testing approach for testing EU SPLs and derived applications.

1. Introduction

The growing adoption of ubiquitous computing and the Internet of Things (IoT) have contributed to the advancement of smart spaces. 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 [1]. In smart spaces, ubiquitous computing focuses on the interaction of end users with the environment, whereas the IoT focuses on the interconnection of devices and services. EUD environments for smart spaces aim to enable end users to take advantage of the device connectivity and end user friendly user interfaces to create applications such as scheduling tasks, convenience through automation, energy management efficiency, health and assisted living [2].

A problem with existing EUD solutions is that they either target a specific group of end users or they assume end users have a baseline technical background. In fact, end users have different computer skills, personality characteristics, professional trainings [3] etc. Technical end users and domain experts, who have the technical ability to integrate pervasive technology in smart settings, can create sophisticated software for their smart spaces. Alternatively, professional software engineers can work with domain experts and end users to design and develop EU software. Less technical end users find it difficult to create software for their smart spaces due to a lack of technical knowledge, domain expertise, and/or difficulties using EUD environments for smart spaces [4]. It would therefore be beneficial to enable end users to reuse the work of technical experts to create software applications for their spaces.

Several quality issues have been reported in applications created by end users. Some of these include errors in the logic, compatibility issues, etc. [5]. The domain of End User Software Engineering (EUSE) is derived from software engineering and provides systematic approaches for end users to create quality software. Reuse is also one of the areas that EUSE identifies as promising for improving end user software quality and promoting end user development because typical end users do not design their software applications for reuse [5]. SPL technology addresses software reuse of requirements, designs and implementations. The problem is that current SPL methods target professional software engineers rather than end users. In an end user environment, the development process is more agile. End users are not familiar with prescriptive SPL methods and therefore changes are needed to define a SPL method to target end users. By adopting reuse, end users would avoid duplicating the work of others to create similar applications. In addition, reuse of more sophisticated and stable end user applications would increase adoption of EUD for smart spaces [6].

This paper describes a systematic EUD reuse approach and environment for smart spaces by using SPL concepts. Section 2 provides the rationale for the approach. Section 3 provides an overview of the EU SPL process for smart spaces. Section 4 describes the End User Software Product Line Prototype (EUSPLP) Development Environment used to develop EU SPLs and derive applications for smart spaces. Section 5 presents a testing approach for testing EU SPLs and derived applications. Section 6 describes the evaluation approach for this work utilizing the smart home EU SPL case study created by this research. Section 7 compares this research with related

URI: http://hdl.handle.net/10125/50610
ISBN: 978-0-9981331-1-9

work. Finally, section 8 provides conclusions and discusses future work.

2. Motivation for EU SPL Development

There are several issues in developing end user applications for smart spaces that can be addressed by applying the EU SPL approach described in this paper. One issue is EUD cost. In current EUD approaches for smart spaces, development cost increases with each application since there is no reuse, and hence applications from the same domain have to be re-developed for different EUD environments and smart spaces. By utilizing the EU SPL approach, there is an initial cost to design and develop the EU SPL. However, the EU application development cost will be lower, since several applications can be derived from the EU SPL to satisfy end user requirements for individual smart spaces.

Another issue is that current EUD approaches do not address variability in end user technical backgrounds and development capabilities. Current EUD environments provide a common user interface for all end users to design and develop applications for smart spaces. They do not address non-technical end user issues in developing EU applications. The EU SPL development environment developed by this research provides different user interface and workflows for technical SPL designers to create EU SPLs, whereas it provides a simpler user interface for end users to derive applications.

Software reuse is limited in current EUD approaches. End users do not develop applications with the goal to reuse and even if they do, current EUD environments do not provide mechanisms for application reuse. Furthermore, end user applications have to be redeveloped for different EUD environments and smart spaces. On the other hand, EU SPLs promote reuse by designing and developing product line features that are realized by common, optional, and variant components and connectors. End user applications are derived by selecting EU SPL features for different EUD environments and smart spaces.

Requirements in EUD are usually unplanned and undocumented. End user requirements are too personalized to create applications that can be reused by other end users for different EUD environments. Furthermore, end users focus on implementation without taking the time to document requirements. Utilizing a systematic EU SPL approach, requirements are collected and documented through the EU SPL requirements elicitation process. Requirements are used to define the EU SPL features, feature groups and feature dependencies. Features are selected by end users to tailor the EU application to their needs.

Software design in end user applications is typically ad hoc. Non-technical end users are not familiar with software design methods and frequently develop low quality applications. Software design is an integral part of the EU SPL process. Technical EU SPL designers design product line features, feature dependencies, feature groups, software architectures, and reusable components that support different EUD environments and smart spaces. Non-technical end users reuse software designs by selecting features and components to derive applications for their smart spaces.

It can be challenging for non-technical end users to develop applications utilizing existing EUD environments for smart spaces. EUD difficulty increases with the complexity of the EU application. In EU SPLs, software development is performed by technical experts. End users derive complex applications for their spaces by selecting and configuring EU SPL features. A user study described by the authors [29] showed the feasibility of having non-technical end users select features from an EU SPL feature model and modify the feature model.

End user applications by non-technical end users are simplistic in nature. EUD environments for smart spaces provide limited user interfaces for developing complex applications. In EU SPLs, application functionalities are organized as SPL features that are realized by common and variable components and connectors. During application derivation, selected features and the corresponding software architecture are used to compose a highly configurable application.

In EUD, software testing is typically haphazard, leading to quality issues in applications developed by non-technical end users. The EU SPL process provides a systematic testing approach that can be used to test EU SPLs, derived applications, and end user application deployment in smart spaces

3. EU SPL Process for Smart Spaces

The EU SPL process provides a systematic approach for EU SPL designers, who can be technical end users and/or domain experts, working with professional software engineers, to design and develop EU SPLs for smart spaces that end users can use to derive applications for their smart spaces. Figure 1 shows the EU SPL process. Similar to conventional SPL engineering processes [7], the EU SPL process consists of two sub-processes: (a) the End User Product Line Engineering (EUPLE) process in which the end user software product line is created, and (b) the End User Application Engineering (EUAE) process in which software applications are derived.

During the EUPLE process, EU SPL designers work with end users to collect requirements, define the product line scope, and create the product line feature model using the EU SPL requirements elicitation process. The feature model captures all the features of the product line and the dependency between them. After the requirements are created, analysis modeling is performed to define the reusable compo-

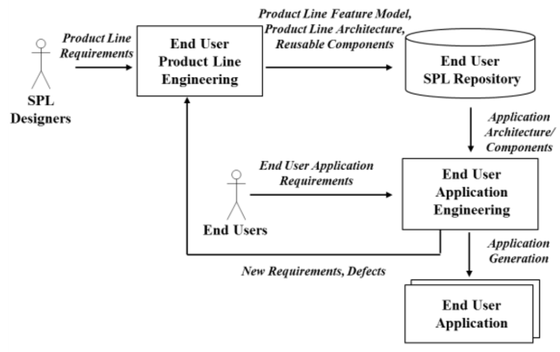

Product Line Requirements
SPL Designers
End User Product Line Engineering
Product Line Feature Model, Product Line Architecture, Reusable Components
End User SPL Repository
Application Architecture/ Components
End User Application Requirements
End Users
End User Application Engineering
Application Generation
New Requirements, Defects
End User Application

Figure 1 End User Software Product Line Process

nents and component interactions needed to realize each feature. During design modeling, the EU SPL architecture is created, feature/component dependency is determined, and component interfaces are defined. During EU SPL implementation, product line components are coded. Finally, during EU SPL testing, test cases are defined for the EU SPL features and feature combinations. There is feedback between the different phases of EUPLE. In particular, issues and software defects identified during EU SPL testing are communicated to the corresponding phases where the issue was introduced. For example, if during testing, a software defect is found that is caused by conflicting features, the issue will be communicated to the EU Analysis Modeling, EU SPL Design Modeling and EU SPL Implementation phases. All artifacts created during the EU SPL engineering are stored in the End User SPL Repository.

During (EU) Application Engineering, end users select the product line features they need from the EU SPL and derive end user applications for their smart spaces. In detail, end users utilize the End User Application Requirements Selection process, to select the product line features from the EU SPL feature model needed for their spaces. Based on the end user's selections, the end user application architecture, components and test cases are derived from the EU SPL Repository. The EU Application Testing process ensures that the test cases are executed successfully against the derived applications. Finally, the derived application is deployed to the end user smart space platform. End users communicate defects and new requirements back to EU SPL designers for future product line releases.

3.1. End User Product Line Engineering

The EUPLE process is composed of the (a) EU SPL Requirements Elicitation, (b) EU SPL Analysis Modeling, (c) EU SPL Design Modeling, (d) EU SPL Implementation, and (e) EU SPL User Application Testing sub-processes.

3.1.1. EU SPL Requirements Elicitation

EU SPL requirements elicitation helps define the overall scope of the product line. EU SPL designers with domain expertise work with end users to collect and document SPL requirements and feature model.EU SPL designers document end user requirements using Use Case modeling. Typical actors in smart spaces are humans. For instance, in a smart home SPL, depending on whether a person is a home resident or an intruder, the smart home can react in different ways. In addition to humans, smart spaces heavily depend on sensors, actuators, devices, and external systems to identify changes to the environment. For instance, a moisture sensor reading might be significant enough to notify a house resident of a possible flood. EU SPL designers document kernel use cases first followed by optional and alternative use cases.

Product line features are requirements or characteristics that are provided by one or more members of the SPL [7]. Feature modeling is used to capture feature commonality / variability and feature dependencies within the EU SPL. In addition, as part of this research, feature modeling was extended to capture feature dependencies in EUD environments (platforms) [8]. Product line features can be (a) platform independent to indicate that a feature does not depend on components or functionalities of a specific EUD environment, or (b) platform specific to indicate that a feature depends on components or functionalities of a specific EUD environment e.g., TeC, Jigsaw.

Feature models are derived by use case modeling. In a feature model, features are organized (a) as common or variable, (b) in feature groups, and (c) as parameterized features. Common features are features that exist in all products derived from the EU SPL. Variable features exist only in some SPL members. Variable features are further categorized as optional or alternative features. Optional features are noncompulsory features that depend on other common or variant features. Alternative features are used to describe mutually exclusive features.

Feature groups are used for grouping similar features. Feature groups can be classified as: (a) exactly-one-of, (b) zero-or-one-of, (c) at-least-one-of and (d) zero-or-more-of. Exactly-one-of feature groups indicate that only one feature from a feature group can be present in an end user application. Exactly-one-of feature groups are used to group alternative features, exactly one of which must be selected during application derivation. Zero-or-one-of feature groups are also used to group alternative features, one or none of which can be selected during application derivation. At-least-one-of feature groups are used to indicate that at least one feature of the feature group must be selected during application derivation. Zero-or-more-of feature groups are used to indicate that zero or more features of the feature group can be selected during application derivation.

Parameterized features are features that can be configured at application deployment time. In the feature model, features are decorated with the «platform-specific» and «platform-independent» UML stereotypes to indicate whether a feature is specific to an EUD environment.

3.1.2. EU SPL Analysis Modeling

EU SPL Analysis modeling consists of static modeling, component structuring, dynamic modeling and feature/component modeling. The EU SPL static model captures the product line components needed to realize the use cases defined and feature model. In addition, component structuring is performed to capture the component reuse stereotype, role stereotype and platform dependencies. This research used UML stereotypes to classify the EU SPL components. To capture component reuse characteristics, the following reuse stereotypes are used: «kernel», «optional», «variant», «default». This research uses the PLUS method role stereotypes to capture the application purpose of each component [7]. For example, a component can be «entity», «control», «timer», etc. Components that are only applicable to specific EUD environments are annotated with the «platform-specific» stereotype.

EU SPL designers use dynamic modeling to capture the object interactions needed to satisfy EU SPL features. UML sequence diagrams are used to model object interactions. Sequence diagrams model the message interaction of objects based on a time sequence [9]. Sequence diagrams are developed for all features defined in the feature model of the EU SPL.

Feature/component modeling is used for mapping features to the components need to realize the feature. This research utilized a table structure to capture this type of relationship.

3.1.3. EU SPL Design Modeling

EU SPL Analysis modeling focus on the analysis of the problem domain, EU SPL Design modeling maps the EU SPL Analysis model to the solution domain [10]. During EU SPL Design modeling the component inter-feature communication, component relationships and component interface models are defined.

As EU SPL designers define features and the components that implement each feature, they might determine situations where components of one feature need to communicate with components of other features to accomplish a task. This research utilized the subscription/notification design pattern for inter-feature component communication. The idea is that instead of components sending messages directly to each other, message broker components are provided as intermediaries. Components can send messages to the message broker, which then notifies subscribed components that have registered with the message broker.

UML component diagrams are used by EU SPL designers to capture (a) components available in a smart home, (b) component relationships, and (c) provided and required interfaces needed for components to communicate with each other.

The components are decorated with UML stereotypes to indicate whether a component is kernel, optional, or variant. Furthermore additional stereotypes are used to capture the role of each component. For instance, a component can be is a «message-broker» component, a «coordinator» component etc. Components can also have a multiplicity indicator to indicate the number of component instances in a smart space. For example, components can have 1…* multiplicity that indicates that there are one or more component instances in the smart space. The connections between components also indicate the required and provided interfaces between components.

EU SPL implementation is the process for implementing the code of each SPL component.

3.2. End User Application Engineering

The EUAE process is composed of the (a) End User Application Requirements Selection, (b) End User Application Derivation, (c) End User Application Deployment and (d) End User Application Testing sub-processes.

The End User Application Requirements Selection process is used by end users to specify the required SPL features for their spaces. The selected features need to be compatible with other features selected from the EU SPL. For instance, an end user cannot select two alternative features or select zero features from an at-least-one-of feature group. The outcome of the EU application requirements process is a derived feature model that captures the features selected by the end user.

The End User Application Derivation process is responsible for deriving the end user application based on the end user feature selections. In detail, the components, component connectors, and component configuration parameters that realize the selected features are derived from the EU SPL Repository to create the application architecture.

The End User Application Deployment process involves end users deploying the derived applications to their smart spaces. During application deployment, EUD environments map and deploy the derived application to a set of devices available in the smart space.

4. End User Software Product Line Prototype Development Environment

The End User Software Product Line Prototype (EUSPLP) development environment was created to validate this research. The EUSPLP environment was designed to support end users and extend EUD environments for smart spaces with SPL capability. The environment provides end user oriented interfaces to enable EU SPL designers to develop the End User SPL and end users to derive applications that can execute in a TeC EUD environment.

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 [11].

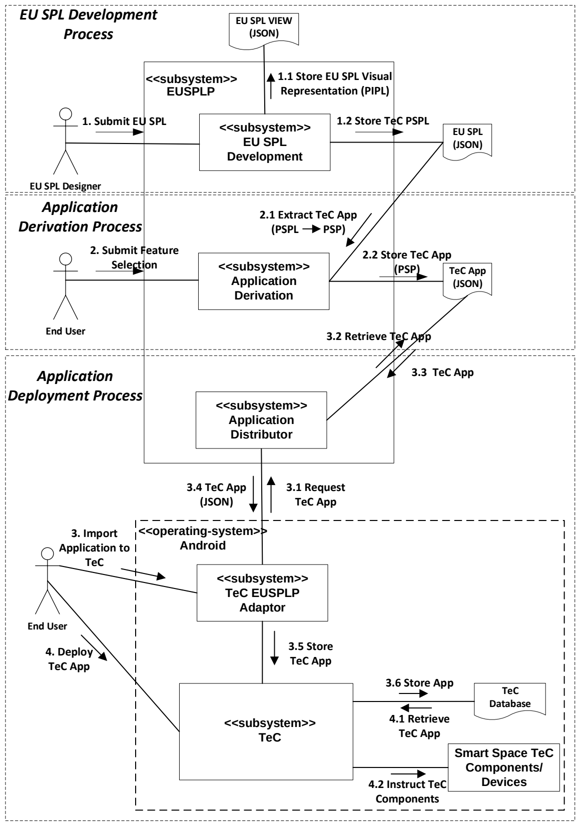
To evaluate the EUSPLP, we developed several EU SPLs for smart spaces utilizing the prototype, derived applications from the product lines created, and deployed derived applications to the TeC EUD environment Android simulator [12].

4.1. EUSPLP System Architecture

Figure 2 shows the EUSPLP subsystem architecture and processes. The EUSPLP subsystem is composed of four subsystems: (1) EU SPL Development, (2) Application Derivation, (3) Application Distributor, and (4) TeC EUSPLP Adaptor. EU SPL Development subsystem provides the user interface, services, and storage mechanisms for EU SPL designers to create and edit end user product lines. The Application Derivation subsystem provides the user interface, services and storage mechanisms for end users to derive TeC applications. The Application Distributor subsystem provides services for external systems to query and retrieve the derived application. The TeC EUSPLP Adaptor subsystem is responsible for acquiring the application derivation specification from the Application Distribution subsystem and sending it to the target TeC EUD environment to be stored in the TeC database. End users utilize the TeC EUD environment to complete the application deployment.

The EUSPLP supports three major processes shown in Figure 2: (1) EU SPL Development, (2) Application Derivation, and (3) Application Deployment. The EU SPL Development process enables end users to develop and store EU SPLs that are used for deriving EU applications. The Application Derivation process enables end users to derive applications for their smart spaces. Finally, the Application deployment process enables end users to import derived applications to the TeC environment and deploy them to their smart spaces.

4.2. EU SPL Development



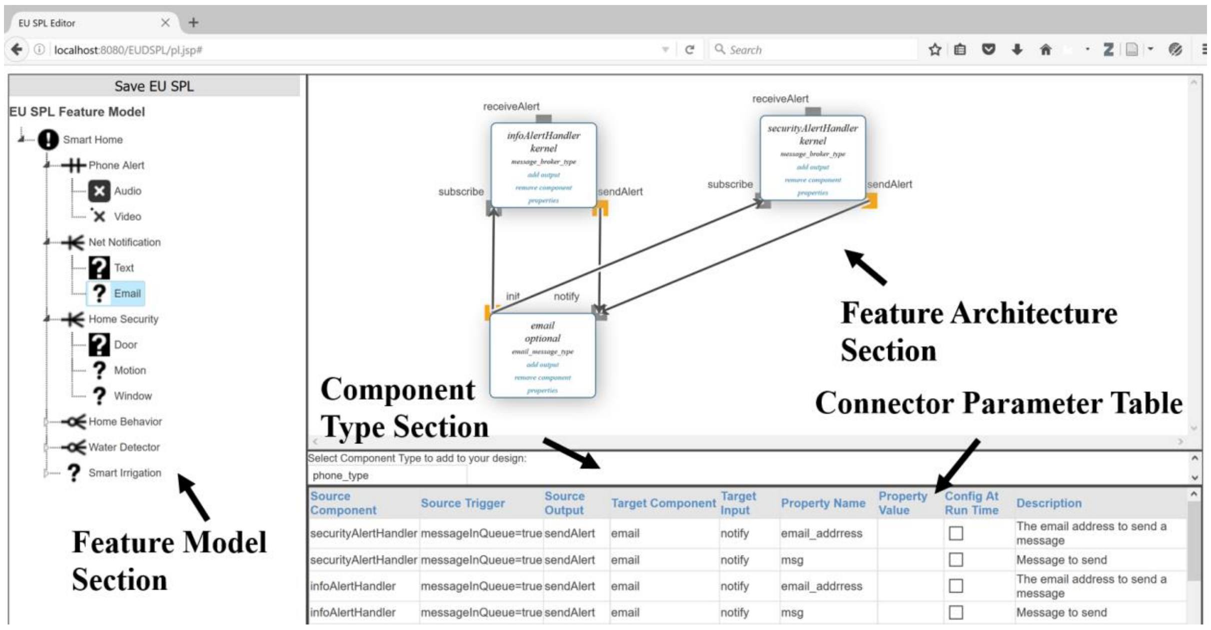
EU SPL Development Process
EU SPL VIEW (JSON)
<<subsystem>> EUSPLP
1.1 Store EU SPL Visual Representation (PIPL)
1. Submit EU SPL
<<subsystem>> EU SPL Development
1.2 Store TeC PSPL
EU SPL (JSON)
EU SPL Designer
Application Derivation Process
2.1 Extract TeC App (PSPL → PSP)
2. Submit Feature Selection
<<subsystem>> Application Derivation
2.2 Store TeC App (PSP)
TeC App (JSON)
End User
3.2 Retrieve TeC App
3.3 TeC App
Application Deployment Process
<<subsystem>> Application Distributor
3.4 TeC App (JSON)
3.1 Request TeC App
3. Import Application to TeC
<<operating-system>> Android
<<subsystem>> TeC EUSPLP Adaptor
End User
4. Deploy TeC App
3.5 Store TeC App
3.6 Store App
TeC Database
<<subsystem>> TeC
4.1 Retrieve TeC App
Smart Space TeC Components/ Devices
4.2 Instruct TeC Components

Figure 2 EUSPLP Subsystem Architecture and Processes

Figure 3 shows the user interface of the EU SPL Editor used to develop EU SPLs. The user interface utilizes an interactive tree structure for representing the EU SPL feature model and a drag and drop interface for component designs to make it easier for EU SPL designers to use. The user interface consists of: (1) The Feature Model section, (2) The Feature Architecture section, (3) The Component Types section, and (4) The Connector Parameter Table.

The Feature Model section was implemented in JavaScript by customizing and extending the jsTree [13] tree plugin of the jQuery technology. The Feature Model organizes product line features and feature groups in a tree structure. Each feature is decorated with a feature symbol to indicate the feature type. Common features are represented with the exclamation point "!" symbol. Optional features are represented with the question mark "?" symbol. Alternative and default features are represented respectively with the black "×" and white "×" symbols. The feature groups supported by the prototype are (a) zero-or-more (b) zero-or-one (c) one or more and (d) exactly-one. The EUSPLP uses the crow's foot notation [14] to capture the cardinality of a feature group. The reason that Crow's foot notation was used in the EUSPLP was because the notation is widely used to represent entity relationships in data models.

The Feature Architecture section shown in Figure 3 is used to capture the component/connector specifi-



Source Component	Source Trigger	Source Output	Target Component	Target Input	Property Name	Property Value	Config At Run Time	Description
securityAlertHandler	messageInQueue=true	sendAlert	email	notify	email_addrress		☐	The email address to send a message
securityAlertHandler	messageInQueue=true	sendAlert	email	notify	msg		☐	Message to send
infoAlertHandler	messageInQueue=true	sendAlert	email	notify	email_addrress		☐	The email address to send a message
infoAlertHandler	messageInQueue=true	sendAlert	email	notify	msg		☐	Message to send

Figure 3 EUSPLP - EU SPL Editor User Interface

cation that realizes each feature. This section utilizes a drag and drop interface, because it is widely used by end users [15]. EU SPL designers can drag and drop components to the feature architecture section and connect them together. The feature architecture section was created in this research by customizing and extending the community edition of the jsPlumb[16] JavaScript Library.

The Parameter Table section specifies all parameters that need to be configured either by the EU SPL designer or by the end users during application derivation. The parameter table user interface is created by extending the editablegrid [17] JavaScript libraries. The Parameter Table displays all component connector properties applicable to a selected feature from the feature model. The table gets auto populated with the relevant component parameters as EU SPL designers connect components in the Feature Architecture section.

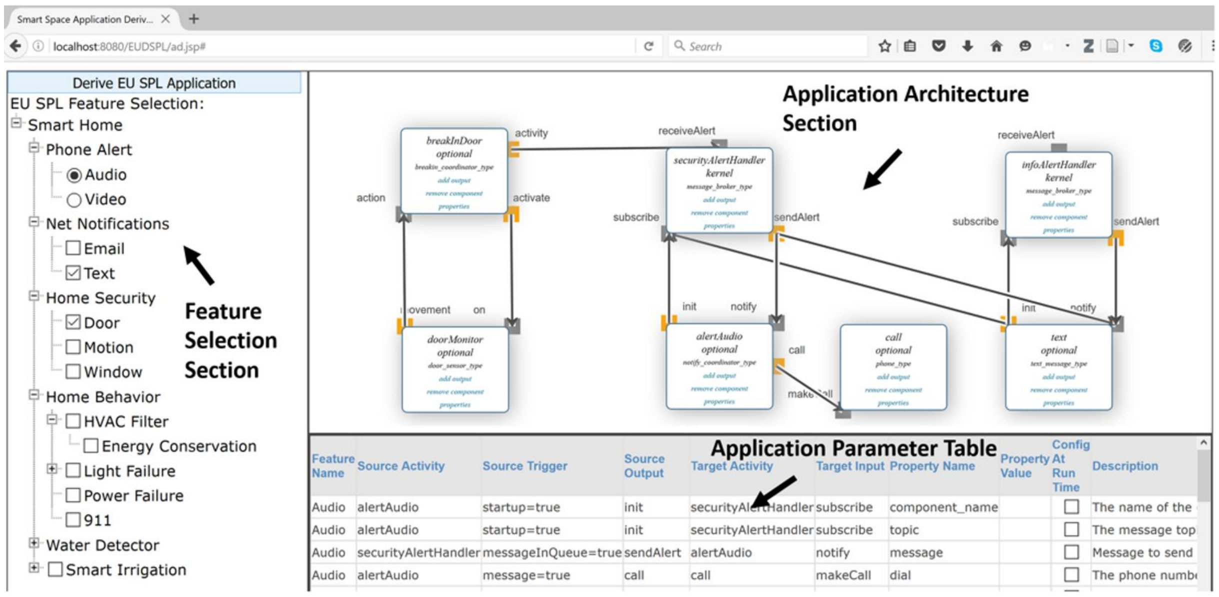
After SPL designers complete creating the product line features, they submit the EU SPL to the EU SPL Development subsystem for storage. The EU SPL Development subsystem first stores the EU SPL visual representation shown on step "1.1 Store EU SPL Visual Representation" in Figure 2. Then the EU SPL Development subsystem transforms the EU SPL visual representation to a Java object structure representing the SPL. The Java objects are serialized to JavaScript Object Notation (JSON) [18] objects in the file system for long term storage shown on step "1.2 Store TEC PSPL" in in Figure 2. JSON is a lightweight human readable data format alternative to XML.

Figure 3 shows the EU SPL for the smart home case study that was developed as part of this research in the EUSPLP. The smart home EU SPL Feature Model section consists of different features and feature groups. For instance the smart home EU SPL has one common feature called "Smart Home". The EU SPL contains the exactly-one-of feature group "Phone Alert" that depends on the "Smart Home" feature. The "Phone Alert" feature group contains two alternative features the "Audio" and "Video". Another example is the one-or-more feature group "Net Notifications" that also depends on the "Smart Home" feature and contains two features that can exist together in derived applications, the "Text" and "Email" features. The Feature Architecture section in Figure 3 shows the component architecture of "Email" feature. The component types section shows the component types that can realize each feature. Finally, the Connector Parameter table in Figure 3 shows all the configuration parameters of the "Email" feature.

4.3. End User Application Derivation

During application derivation, end users are presented with the end user view of the feature model, the Feature Selection Section, the Application Architecture section and the Application Parameter table shown in Figure 4. End users select the desired features for their EU application and the EUSPLP automatically derives the application architecture.

The nodes of the feature selection section represent common, optional and alternative features. Checkboxes represent optional features and radio boxes represent alternative features. Common features are represented as pre-selected checkboxes. End



EU SPL Feature Selection:
- Smart Home
  - Phone Alert
    - [✓ Audio]
    - Video
  - Net Notifications
    - Email
    - [✓ Text]
  - Home Security
    - [✓ Door]
    - Motion
    - Window
  - Home Behavior
    - HVAC Filter
      - Energy Conservation
    - Light Failure
    - Power Failure
    - 911
  - Water Detector
  - Smart Irrigation

Application Parameter Table

Feature Name	Source Activity	Source Trigger	Source Output	Target Activity	Target Input	Property Name	Property Value	Config At Run Time	Description
Audio	alertAudio	startup=true	init	securityAlertHandler	subscribe	component_name		☐	The name of the
Audio	alertAudio	startup=true	init	securityAlertHandler	subscribe	topic		☐	The message top
Audio	securityAlertHandler	messageInQueue=true	sendAlert	alertAudio	notify	message		☐	Message to send
Audio	alertAudio	message=true	call	call	makeCall	dial		☐	The phone numb

Figure 4 EUSPLP Feature Selection User Interface

users, based on their requirements and their smart space configurations, select a feature combination from the feature model, configure the feature parameter table and submit their selections to the EUSPLP Application Derivation subsystem as shown on step "2 Submit Feature Selection" in Figure 2. The Application Derivation subsystem extracts the component architecture of the selected features from the SPL and composes the end user application as shown on step 2.1 in Figure 2. The end user application (TeC App) is serialized to JSON in the file system shown on step 2.2 in Figure 2.

4.4. End User Application Deployment

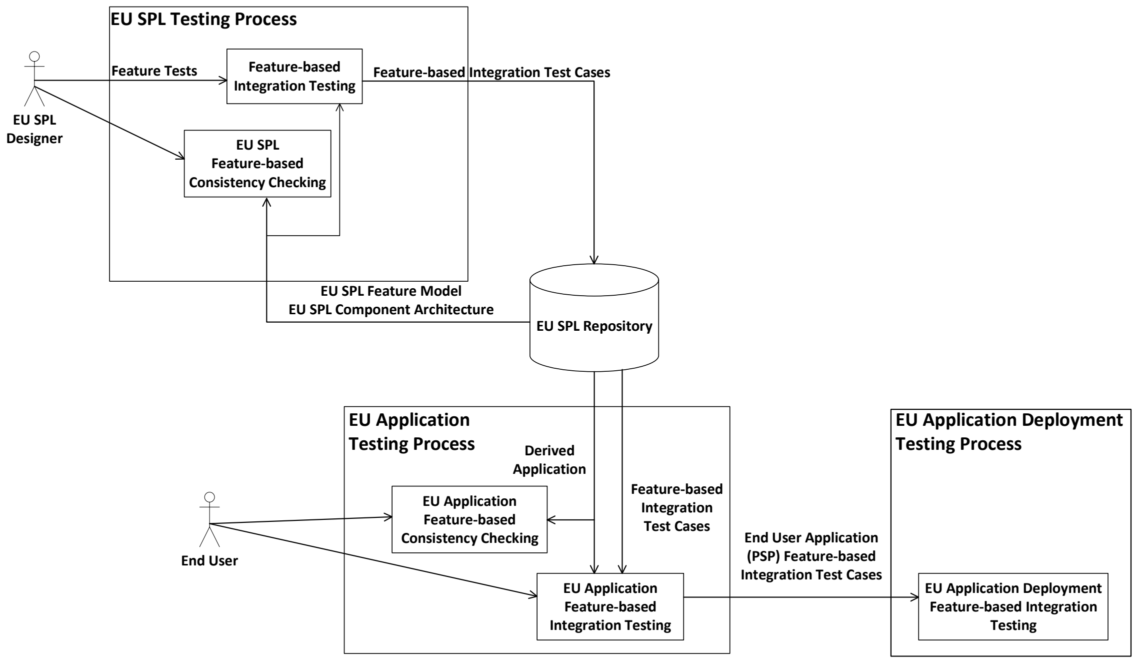
During application deployment, end users utilize the TeC EUSPLP adaptor to import the derived application to their TeC EUD environment as shown on steps 3 to 3.6 in Figure 2. Figure 4 shows the EUSPLP Feature Selection User Interface for the smart home product line. In this example three features are selected from the smart home product line: "Audio", "Text" and "Door". The left side of Figure 4 shows the application architecture of the selected features. Based on the selected features the EU application JSON representation for the TeC environment is derived. The EU application JSON is distributed to the TeC Android platform simulator when the EU application is deployed.

5. EU SPL Testing Approach

As part of this research an overall testing approach was defined to test EU SPLs and derived applications. The EU SPL Testing Approach is a hybrid approach that builds on the testing methods described by Abu-Matar [18] and Olimpiew [19]. Abu-Matar used static SPL consistency test cases to test SPLs and derived applications created in his research. Olimpiew described an approach for defining test cases for each feature that can be retrieved and executed during application derivation. Similarly, the test cases created in this research consist of: consistency test cases for testing the EU SPL and the derived applications; and test cases for each feature that can be executed during product line creation, application derivation and application deployment.

Figure 5 shows the overall EU SPL Testing Approach used to test EU SPLs and derived applications. The testing approach is composed of: (a) EU SPL Testing, (b) EU Application Testing, and (c) EU Application Deployment Testing processes. The EU SPL Testing process, which is used for testing the SPL, consists of EU SPL Feature-based Consistency Checking and Feature-based Integration Testing. EU SPL Feature-based Consistency Checking executes static test cases to verify feature and feature group dependencies. Feature-based Integration consists of integration test cases defined by EU SPL designers to test the EU SPL. In particular, integration test cases are developed for every feature and feature combination in the EU SPL to test the component interconnections. As shown in Figure 5, Feature-based Integration test cases are stored in the EU SPL Repository for usage during application derivation.

The EU Application Testing Process, which is responsible for testing applications derived from the EU SPL, consists of EU Application Feature-based Consistency Checking and EU Application Feature-



EU SPL Testing Process
Feature Tests
Feature-based Integration Testing
Feature-based Integration Test Cases
EU SPL Designer
EU SPL Feature-based Consistency Checking
EU SPL Feature Model
EU SPL Component Architecture
EU SPL Repository
EU Application Testing Process
Derived Application
Feature-based Integration Test Cases
EU Application Deployment Testing Process
EU Application Feature-based Consistency Checking
End User
End User Application (PSP) Feature-based Integration Test Cases
EU Application Feature-based Integration Testing
EU Application Deployment Feature-based Integration Testing

Figure 5 Overall EU SPL Testing Approach

based Integration Testing. EU Application Feature-based Consistency Checking contains static test cases used to verify the compatibility of features that comprise the derived application. EU Application Feature-based Integration involves executing integration test cases to test the component architecture and implementation of the derived application. The integration test cases are a subset of the EU SPL integration test cases and are based on the selected features that comprise the derived application. As shown in Figure 5, Feature-based Integration test cases to test the derived application are selected from the EU SPL Repository corresponding to the features selected by the end user.

The EU Application Deployment Testing Process shown in Figure 5, is responsible for testing the distributed deployment and execution of the TeC derived application. In detail, during the deployment testing process, EU Application Deployment Feature-based Integration Testing involves executing integration test cases to test the deployment and execution of components and their interconnections in the environment. The integration test cases are the same ones used during EU Application Feature-based Integration Testing. The integration test cases are reused to test the deployment of the derived application.

The Feature-based integration test cases provide test coverage of each feature and component during EU SPL Testing, EU Application Testing and EU Application Deployment Testing. In particular test cases are developed to: (a) test each component (b) test each feature by testing the components and connectors that realize the feature (c) If a feature depends on other features, test the feature in combination with the features it depends on.

6. Evaluation

To validate this research a smart home EU SPL case study was created with 24 common and variant features organized in different feature groups. In addition, 32 kernel and variant components were created to realize these features. The case study has features from the domains of home automation, home security, home notifications, home maintenance, resident comfort and energy conservation.

The case study was developed following the EU SPL Engineering process. In particular, the End User Product Line Engineering process was used to design and develop the case study and the End User Application Engineering process was used to derive applications. All features of the smart home EU SPL case study were implemented using the prototype's EU SPL development subsystem. In addition, several applications were derived from the smart home EU SPL using the application derivation interface of the EUSPLP. The derived applications were deployed to the TeC Android simulator.

To test the smart home EU SPL this research developed and executed 32 EU SPL feature-based consistency test cases. Examples of EU SPL consistency test cases are "Zero-or-more-of Feature Group contains Optional Feature", "Common Feature contains Kernel Component", etc. Furthermore 79 feature-based integration test cases were developed and executed to test individual component connectors, multi-component interactions of dependent features and feature interactions. To execute both consistency and feature-bases test cases, this research developed a testing framework that can simulate a TeC EUD environment. All consistency and feature-bases test cases were executed successfully in the smart home EU SPL case study using the testing framework.

For testing derived applications from the smart home case study, 13 EU application consistency test cases were developed to ensure the validity of the application feature selection. An example of a consistency test case is “All Common Features were selected”. In addition, the applicable feature-based integration test cases for the features that comprise the derived application were used to test the component architecture and implementation of the application. The testing framework was used to execute consistency and feature based test cases. For all derived applications of the smart home EU SPL, all consistency and feature based test cases were executed successfully.

Finally, to test the deployment of the derived applications, the feature based test cases from EU application testing were executed in the TeC Android simulator utilizing the simulator’s testing interface. For all the derived applications from the smart home case study that were deployed to the TeC simulator, all test cases executed successfully.

7. Related Work

Our research builds on prior work in EUD environments for smart spaces, SPL methods, and current SPL approaches for end users and smart spaces. The functionality provided by EUD environments for smart spaces can be grouped in two general areas: smart space configuration and context aware environments. Smart space configuration environments enable end users to control and combine functionality of devices. Jigsaw[20], and Puzzle [21] are some examples. Context aware environments create rules based on user context (activity, location, identity, time) and device functions. PIP [22], FedNet [4], GALLAG Strip[23], and TeC [11] are some examples. Current EUD environments for smart spaces do not address reuse. End user applications are created for specific environments and are not portable to other environments. For instance, an end user application for TeC is only applicable for the TeC EUD environment and cannot be reused for Jigsaw. In contrast, our research extended existing EUD environments for smart spaces with product line support.

SPL methods such as ISO ISO/IEC 26550 [24], PLUS [29], COPA [25], and KobrA[26] address the problem of modeling variability in SPLs and provide processes to design SPLs and derive applications from them. The research described in this paper has extended current SPL approaches to provide support for EUD development and smart spaces.

Current research on utilizing SPL concepts for end users and smart spaces includes SimPL [27] and Perez et al.[28]. As with our research, SimPL uses components, connectors and triggers to create application logic. Perez et al. utilize variability engineering for professional engineers to cooperate with end users to capture end user requirements for smart spaces. Our research extends Perez’s work beyond requirements elicitation for SPLs by utilizing visual languages and application models of EUD environments to create SPLs for smart spaces.

8. Conclusions and Future Work

This paper has described a systematic approach and development environment for designing, developing, and testing EU SPLs that end users can use to derive applications for their smart spaces. This approach offloads from the end user the task of developing the SPL software. Instead, the end user selects features from a feature model and the environment derives the application architecture and implementation. A user study [29] showed the feasibility of this approach. This research defined the EU SPL process, which provides a step by step process for designing, developing, and testing EU SPLs. The EU SPL process extended existing SPL approaches to end user development and smart spaces, as well as for deriving EU applications. The EUSPLP development environment was developed to enable the implementation of EU SPLs and application derivation for smart spaces. A testing approach was developed to test the EU SPLs and derived applications created using the EUSPLP development environment. The overall contributions of this research are the End User Product Line Engineering process, the EUSPLP development environment, and the EU SPL testing approach.

This research will continue by investigating and expanding the EUSPLP environment with smart space security models for EUSPLs. Another area for extending this research is end user visual languages for EU SPLs. This research performed a preliminary user study [29] to investigate different visual symbols for representing feature types, user interfaces for creating EU SPLs, and deriving applications for smart spaces. An extension of the original user study could be conducted to evaluate and enhance the EUSPLP visual language and user interface. EU SPL testing is another area for future research. The testing framework developed in this research could be enhanced by investigating approaches to automatically generate test cases based on feature dependencies and component relationships, in addition to test cases provided by EU SPL designers.

9. References

[1] R. Singh, P. Bhargava, and S. Kain, “State of the art smart spaces: application models and software infrastructure,” ACM Ubiquity, p. 7:2–7:9, Sep-2006.

[2] P. Rashidi and D. J. Cook, “Keeping the Resident in the Loop: Adapting the Smart Home to the User,” Trans. Sys. Man Cyber. pp. 949–959, Sep. 2009.

[3] L. Beckwith and M. Burnett, "Gender: An important factor in end-user programming environments?," in Proceeding of IEEE Symposium on Visual Languages - HCI Rome, Italy, 2004, pp. 107–114.

[4] F. Kawsar, T. Nakajima, and K. Fujinami, "Deploy Spontaneously: Supporting End-Users in Building and Enhancing a Smart Home," in Proceedings of the 10th International Conference in Ubiquitous Computing, Seoul, South Korea, 2008, pp. 282–291.

[5] M. Burnett, "What Is End-User Software Engineering and Why Does It Matter?," in Proceedings of the 2nd Intern. Symp. on EUD, 2009, pp. 15–28.

[6] V. Tzeremes and H. Gomaa, "A Software Product Line Approach for End User Development of Smart Spaces," in Proceedings of the 5th International Workshop on Product LinE Approaches in Software Engineering, NJ, USA, 2015, pp. 23–26.

[7] H. Gomaa, Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley Professional, 2005.

[8] V. Tzeremes and H. Gomaa, "A Multi-platform End User Software Product Line Meta-model for Smart Environments," in Proceedings of the 11th International Joint Conference on Software Technologies ICSOFT-EA, Lisbon, Portugal, 2016, pp. 290–297.

[9] J. Rumbaugh, I. Jacobson, and G. Booch, Unified Modeling Language Reference Manual, The (2Nd Edition). Pearson Higher Education, 2004.

[10] H. Gomaa, Real-Time Software Design for Embedded Systems. Cambridge, 2016.

[11] J. P. Sousa, "Foundations of Team Computing: Enabling End Users to Assemble Software for Ubiquitous Computing," in International Conference on Complex, Intelligent and Software Intensive Systems, Krakow, Poland, 2010, pp. 9–16.

[12] J. P. Sousa, X. Shen, V. Tzeremes, and F. Hodum, "TeC apps for smart spaces: simple, decentralized, resilient, and self-healing," ACM Conference on Ubiquitous Comp., PA, USA, 2012, pp. 637–638.

[13] J. Duckett, JavaScript and JQuery: Interactive Front-End Web Development, Wiley Publishing, 2014.

[14] R. Barker, Case Method: Entity Relationship Modelling, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[15] C. Appert, O. Chapuis, E. Pietriga, and M.-J. Lobo, "Reciprocal Drag-and-Drop," ACM Trans. Comput.-HCI., vol. 22, no. 6, p. 29:1–29:36, Sep. 2015.

[16] S. Porritt, The jsPlumb JavaScript library. 2016.

[17] P. Máca, Editablegrid. Webismymind, 2016.

[18] M. Abu-Matar and H. Gomaa, "An Automated Framework for Variability Management of Service-Oriented Software Product Lines," in Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on, 2013, pp. 260–267.

[19] E. M. Olimpiew and H. Gomaa, "Reusable Model-Based Testing," in Formal Foundations of Reuse and Domain Engineering: 11th International Conference on Software Reuse, ICSR 2009, Falls Church, VA, USA, Springer, 2009, pp. 76–85.

[20] J. Humble et al., "Playing with the Bits User-Configuration of Ubiquitous Domestic Environments," in Proceedings of the 5th Intern. Conference in Ubiq. Comp., Seattle, WA, 2003, pp. 256–263.

[21] J. Danado and F. Paternò, "Puzzle: a visual-based environment for end user development in touch-based mobile phones," in Human-Centered Software Engineering, Springer, 2012, pp. 199–216.

[22] J. Chin, V. Callaghan, and G. Clarke, "End-user Customization of Intelligent Environments," in Handbook of Ambient Intelligence and Smart Environments, Springer US, 2010, pp. 371–407.

[23] J. Lee, L. Garduño, E. Walker, and W. Burleson, "A Tangible Programming Tool for Creation of Context-Aware Applications," in Proceedings of the International Joint Conference on Pervasive and Ubiquitous Computing, 2013, p. 391.

[24] ISO/IEC 26550:2016, "Software and systems engineering – Reference model for product line engineering and management," International Organization for Standardization, Geneva, Switzerland, ISO ISO/IEC 26550, 2016.

[25] P. America, H. Obbink, J. Muller, and R. van Ommering, "COPA: A Component-Oriented Platform Architecting Method for Families of Software Intensive Electronic Products," in First Conference on SPLE, Denver, Colorado, 2000.

[26] C. Atkinson and D. Muthig, "Component-Based Product-Line Engineering with the UML," in Software Reuse: Methods, Techniques, and Tools, vol. 2319, Springer, 2002, pp. 155–182.

[27] A. Malaer and M. Lampe, "SimPL: A Simple Software Production Line for End User Development," in 15th Asia-Pacific Software Engineering Conference, Beijing, China, 2008, pp. 179–186.

[28] F. Perez and P. Valderas, "Allowing End-Users to Actively Participate within the Elicitation of Pervasive System Requirements through Immediate Visualization," in Proceedings of the 4th International Workshop on Requirements Engineering Visualization, Atlanta, Georgia, USA, 2009, pp. 31–40.

[29] V. Tzeremes and H. Gomaa, "XANA: An End User Software Product Line Framework for Smart Spaces," in 2016 49th Hawaii International Conference on System Sciences (HICSS), 2016, pp. 5831–5840.