XANA: An End User Software Product Line Framework for Smart Spaces

Vasilios Tzeremes
George Mason University
vtzereme@gmu.edu

Hassan Gomaa
George Mason University
hgomaa@gmu.edu

Abstract

This paper describes XANA, an end user development framework that extends existing end user development tools for smart spaces with software product line concepts. The framework targets two types of users: the Software Product Line (SPL) designers and the end users. SPL designers use XANA’s product line creation interface to create the software product line for end users. The product line creation interface allows SPL designers to capture the product line’s feature model, the component design that implements each feature and the product line architecture. End users using XANA’s application derivation interface select the features needed for their smart spaces and generate software applications from the product line previously created. In this paper we present the phases and operational semantics of XANA. We illustrate its use with a case study. The paper presents a preliminary user study that was conducted to assess the usability of the framework.

1. Introduction

As computing becomes ubiquitous software demands are rapidly increasing. Software requirements for end users are becoming personalized and often fluctuate. Professional engineers do not have the capacity to satisfy all software needs. End users are expected to contribute more and more to software development [1]. End User Development (EUD) involves a set of methods, techniques, and tools that enable users of software systems, who are acting as non-professional software developers, to create, modify, or extend a software artifact [3]. Examples of EUD are spreadsheet programming, visual programming, email rule filters, web site creation tools, etc. Another prominent area for end user development is smart spaces. Smart spaces are physical environments equipped with pervasive technology that sense and react to human activities and changes in the environment. Examples of smart spaces are homes, hospitals, offices, farms, bridges, and highway tolls that can be instructed to act proactively and/or reactively to different activities. As connectivity across devices increase, network speeds surge, pervasive interfaces to control devices are created and Human Computer Interaction (HCI) improvements makes it possible for end users to program their smart spaces. For example end users might want to configure their own solutions for home security, building automation, space notifications, elderly care, energy conservation and office ergonomics.

Having end users creating software applications has several benefits. Some of the benefits are that it empowers end users to create software applications, the applications are built to the end user specs and there is better adaptation of the software applications by end users. Having end users creating software applications also helps. End users have different technological backgrounds. Thus not all end users have the same development abilities. Furthermore EUD is more opportunistic than systematic, requirements are usually unplanned and undocumented, reuse is ad-hoc and software testing is typically haphazard, leading to quality issues [4]. End User Software Engineering (EUSE) focuses on approaches, techniques and tools to improve the quality of end user software [5]. Software Product Line (SPL) methods can also help end users to reuse work of others and improve the software quality. The issue with SPL methods is that they target professional software engineers and can be hard for end users to understand. This research investigates how SPL concepts can be applied to end user development for smart spaces so end users can reuse software applications for their spaces instead of having to develop them from scratch.

This paper describes a lightweight end user development framework for smart spaces called XANA that is based on SPL concepts. Section 2 discusses problems and challenges in end user development for smart spaces. Section 3 introduces the end user software product line process. Section 4 describes Team Computing, an EUD tool for smart spaces and XANA, its phases, language operational semantics, and its application to a surveillance case study. Section 5 presents the design and results of a preliminary user study we conducted to validate XANA’s usability. Section 6 explains our evaluation approach. Section 7 discusses benefits and limitations of XANA. Section 8 compares XANA with related work. Finally, section 9 provides conclusions and discusses future work.
2. EUD Challenges for Smart Spaces

Several EUD tools have been proposed to enable end users to customize their smart spaces. One of the problems with existing 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, ages, gender [6] etc. Technical end users and domain experts have the ability to create sophisticated software for their smart spaces. However, less technical end users find it difficult to create software for their smart spaces due to a lack of technical knowledge, domain expertise, and difficulties using EUD tools. For example FEDNet, an EUD architecture that enables end users to deploy devices and custom EUD applications to smart spaces that the FEDNet architecture is deployed. In a usability study for FEDNet, a participant said that she would be interested in having the type of applications created by FEDNet in her home but she would rather have a more technical end user develop it [7] for her. 35% of the participants in the study provided similar feedback. The focus of our work is to enable end users to reuse work of more technical end users and domain experts to create software for their spaces. Several quality issues have been reported by applications created by end users. Some of them include errors in the logic, compatibility issues etc. One of the reasons for this is lack of reuse [5]. The domain of EUSE is derived from software engineering and provides systematic approaches for end users to create quality software. Reuse is one of the areas that EUSE identifies as a promising for improving end user software quality and promoting end user development. Some of the issues of reuse in EUD is that end users don't design their software applications for reuse and even if they do, other end users have difficulties finding and reusing the software applications to address their needs [5]. SPL technology addresses software reuse of requirements, designs and implementations, and could assist with EUSE. The problem is that SPL methods target professional software engineers rather than end users. SPL creation involves requirements gathering, commonality /variability analysis, feature modeling, variable architecture definition, component design and implementation. In an end user environment, the development process is more agile. End users are not familiar with prescriptive SPL methods and therefore a more light-weight SPL method is needed to target end users.

One approach for applying SPL technology to end user smart spaces would be to have SPL designers (who are technical end users and domain experts) apply product line methods to create end user SPLs for smart spaces and have end users derive their individual smart space applications from these SPLs. There are several challenges in applying SPL to end user development. One challenge is creating the end user SPL architecture. The end user development process needs to address issues of how SPL features are created, how EUD components relate to features, how the commonality and variability of features, components and connectors are captured, how EUD applications are derived and deployed to smart spaces, and how EUD SPLs interact with existing EUD tools and smart space architectures. Another challenge is to define a language that SPL designers can use to create the EUD SPLs and end users to derive applications. The language needs to be expressive enough to capture SPL concepts while targeting end users. Furthermore the development approach needs to distinguish between end user SPL creation by technical developers/end users and application derivation by less technical end users. Feature documentation is also another challenge. As indicated previously in the FEDNet user study, end users have difficulties understanding how EUD architectures work. By abstracting EUD applications as SPL features that end users can select for their spaces is a step forward because end users do not have to deal with the complexity of creating these applications from scratch but the features need to be documented at the end user level. SPL designers need to document the SPL appropriately so end users can understand what each feature does and if it is appropriate for their environment. Integrating end user SPL architectures with existing EUD tools is also a challenge. Different EUD tools for smart spaces have different characteristics. The EUD SPL architecture needs to be capable of using and extending these tools. The idea is for technical end users and domain experts to use existing EUD tools to design the EUD SPL architecture. End users should also be able to use EUD tools to deploy EUD SPL applications to their smart spaces. Finally end user SPLs need to be capable of evolving and this presents other challenges. Some of the evolution issues are how does the EUD SPL approach support updates to the smart space feature model and architecture, how do end users make changes to derived products, how are defects communicated to SPL designers, how is the product line updated, and how are product line updates communicated back to end users.

3. End User Software Product Line Process

A software product line is “a set of software intensive systems sharing a common, managed set of features that specify the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way” [8]. A SPL consists of a family of systems that share common and variable functionality. Common functionality reuses features and components among all products derived from the product line. Variable functionality is what differentiates each of the product members of the SPL. Product lines are widely used and can be found in many commercial software applications that are offered in different versions. An example is the Windows operating system, which is distributed in different ver-
End user development technologies for smart spaces provide end users with device and software abstractions to simplify end user application development. Prior to EUD technologies, end users had to install devices in their spaces and use the device programming APIs to create personalized applications for their spaces. EUD technologies act as the middleware between devices and applications. Devices are deployed to the EUD architecture. The EUD architecture provides device abstractions for end users to create software applications and during application deployment the EUD architecture translates those abstractions to executable code for interfacing with the devices. Our proposed SPL EUD Framework (XANA) is an extension to EUD technologies that enables SPL developers to utilize EUD technologies to create SPLs and end users derive and deploy software applications for their spaces. By integrating SPL concepts to EUD, end users can reuse and customize for their spaces applications created by domain experts without having to develop the software themselves. This section describes the end user development technology for smart spaces used in this research, Team Computing and the integration with our proposed EUD SPL solution XANA. The first section describes Team Computing. The second section describes XANA and provides examples of its integration with Team Computing.

4.1. Team Computing

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. Activities are software components, devices, and humans operating in ubiquitous computing environments. Teams have no central control: elements play their roles autonomously and their behavior is emergent [11]. Team designs define teams and are created in the TeC editor. A team design is a collection of activities connected together. Additional
logic, conditions and output events can be added by end users to activities for customization. Output events are connected to input events of other activities and get activated when their holding condition is true.

In the TeC editor activities are shown as rectangle boxes. Output and input events are shown with boxes attached to activities. Arrows used to connect activities in team designs are directed from output to input events. TeC also supports data streams generated by activities. An example of a data stream is video feeds. Data streams are shown as triangles attached to activities. Arrows are directed from output to input streams. After a team is designed it can be deployed to a smart space that is configured with TeC. Figure 2 shows an example of a surveillance team in TeC. The goal of the team is to notify the house owners when a break-in is detected while they are away. The team has three activities “track”, “presence detector” and “notify by phone”. The “track” activity is implemented with two RFID readers installed in the property, one installed for entry and one for exit, and connectivity to the TeC database that contains the identity of each provisioned user, the RFID card information for each user and user status. The “presence detector” activity is implemented by a group of motion sensors and a software application that collects and aggregates the status of the sensors and makes the determination if movement is detected. Finally the “notify by phone” activity is implemented as a software application that is connected to the phone network and makes automated phone calls. The “track” activity has an output event called “away”. The “away” event has one parameter called “track_res” to indicate the identity of the residents the “track” activity should monitor. In this case Frank and Dora are the residents of the house. The “away” event gets triggered when Frank and Dora are not in the house and is connected to the “on” input event of the “presence detector” activity. The “on” event activates the presence detector while the residents are away. The “presence detector” activity has an output event called “action” to indicate that movement was detected in the house. The “action” event is connected to the “send” input event of the “notify by phone” activity. The “notify by phone” activity has one parameter called “notify_res” that captures residents that need to be notified.

A change in the TeC activity physical implementation does not have to impact the team designs as long as the activity interface is preserved but team designs would have to be redeployed in the space. For example if the implementation of “track” activity changes and instead of the RFID technology there is location tracking by cell phone mechanism. The surveillance team interface would still be valid but the team would have to be redeployed so the location tracking technology gets new computing and messaging instructions by the TeC architecture. Devices can also be added, removed or fail functioning while a team operates. For example if a motion sensor is added or removed from the “presence detector” activity the team will readjust. In the event that all motion sensors are removed or fail to operate due to power failure then the TeC editor will show errors for all teams that use “presence detector” activities.

4.2. XANA SPL EUD Framework

XANA is an end user development framework for smart spaces that promotes reuse by adopting software product line concepts. To illustrate XANA’s capabilities, we extended Team Computing (TeC), with a design language that enables SPL designers to create EUD SPLs and end users to customize and deploy applications to their smart spaces [11], [12]. We selected TeC as the EUD technology to integrate with XANA due to TeC’s expressiveness for end user development. XANA has two phases: product line creation and application derivation. During product line creation, SPL designers define the product line features and create the product line architecture. SPL designers are technical end users or domain experts that develop software applications either for personal or commercial purposes. The SPL is designed in XANA using the TEC EUD technology’s component or device specifications. During application derivation, end users select features for their spaces and XANA deploys the features in the smart space. In particular XANA derives the component architecture specification from the SPL that implement the selected features. Then XANA maps the component architecture specification to component instances available in the smart space and deploys them to the EUD architecture.

The following parts of section 4 discuss the product line creation and the application derivation phases of XANA using a surveillance product line as a case study. The surveillance product line has three main features “phone alert”, “video alerts” and “911”. The “phone alert” feature notifies the house owners with text messages when a break in is detected. The “video alert” feature notifies the house owners with a video feed of what is happening in the house during a break in. The “911” feature notifies the police in the event the house owners cannot be reached. The “phone alert” and “video alert” features are alternative features. The “911” feature is an optional feature.

4.2.1. Product Line Creation

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
<th>Activity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>track_res</td>
<td>Frank, Dora</td>
<td>track</td>
<td>Tracked Residents</td>
</tr>
<tr>
<td>notify_res</td>
<td>Frank</td>
<td>notify by phone</td>
<td>Notify Residents</td>
</tr>
</tbody>
</table>
During product line creation, SPL designers create the product line’s feature model and the product line architecture in XANA. The feature model contains all features of the product line organized in a tree structure. To create a feature, SPL designers create the component design using the end user development tool’s components and connectors. The component design is tagged with the appropriate XANA feature symbol and is added to the feature model. The product line features and component designs comprise the product line architecture. A feature symbol in XANA indicates the feature type. XANA supports common, optional, alternative and default alternative feature symbols to enable SPL designers to create feature models. The common feature symbol is used to denote mandatory features. The optional feature symbol represents optional features that end users may want to select. The alternative feature symbol represents replacement features. The alternative default feature symbol is used to indicate the default selected feature among a set of alternatives. The common feature symbol is represented with the question mark “?””. The optional feature symbol is represented with the question mark “?””. Alternative and alternative default features are represented respectively with the “×” and “⊗” symbols. XANA utilizes a tree structure to capture dependency among features in the feature model. For example consider feature {B} → {A}, feature {B} that depends on feature {A}. In the feature model feature {A} will be the parent node of feature {B}. Alternative features are grouped under a parent node. For example consider two alternative features {D} V {E}. In the feature model the application designer would have to create a parent node {C} to group features {D} and {E}. SPL designers don’t have to create SPL features from scratch. They can use features in the feature model as templates update the component design and add it as a new feature to the feature model. SPL designers are also responsible during product line creation for documenting features so end users know what each feature does and what components need to be available in the space during feature deployment.

During product line creation the feature parameter table is also created. Feature parameters are SPL parameters that are defined by SPL designers during product line creation and are configured by end users during application derivation or at application run time.

Examples of feature parameters are user identities, timers, phone numbers etc. The feature parameter table captures parameterized feature metadata. Feature parameter metadata are data that describe a parameter in detail for example parameter data type, name, etc.

Figure 3 – 5 show the design of the surveillance product line. Figure 3 shows the design of the “alert” feature in TeC. The “alert” feature has a “track” activity that tracks the house residents. When the residents are away the “track” activity sends a message to activate the “presence detector”. While the residents are away if the “presence detector” detects any activity in the house it will send a message to the “notify” activity. The “notify” activity once it receives the message will send an intrusion message to the residents. The “notify” activity is outlined with a dotted line to indicate that this is an abstract activity. Abstract activities imply that the concrete activity will be provided by a child feature. The bottom part of Figure 3 describes the feature model and is divided in two sections. The left side contains a toolbar with the product line common (“!”), optional (“?”), alternative (“×”) and default alternative (“⊗”) feature symbols and the feature model of the product line. As the application designer completes designing each feature he drags the desired feature symbol to the feature model and names the feature. In this case the feature is named “alert”. Since the main goal of the surveillance team is to notify residents the application designer selected the common feature symbol to tag this feature. This means that no application can be derived without the alert feature. The right side of Figure 3 contains the feature parameter table. The table has four columns: name, parent, type, description. The name column corresponds to the name of the parameter. The parent column indicates the activity that the parameter belongs to. The type column specifies the parameter data structure. The description column documents the parameter. The allowable and default value columns were omitted from this table because they were not used in this example. The parameter table in Figure 3 has two parameterized features. The first one is the “track_res”. The parameter belongs to the “track” activity and its data type is a list of residents’ ids. This parameter indicates which residents

Figure 3. Surveillance PL common feature design

Figure 4. Surveillance PL optional feature design
need to be tracked. Similarly the “notify_res” parameter belongs to the “notify” activity and indicates which residents need to be notified.

Figure 4 shows the component design of the “911” feature. The “911” feature is decorated with the optional (“?”) feature symbol in the feature model. The “911” feature has 4 activities: “track”, “presence detector”, “call 911” and “notify by phone”. As before, there is a connector between the “away” output of the “track” activity and the “on” input of the “presence detector” to indicate that when the residents are away, the presence detector is activated. When the “action” event gets generated by the “presence detector”, a “bridge” event gets triggered in the “call 911” activity. In addition, an alert is sent to the “notify by phone” activity which represents the residents’ smart phone. If this is a false alarm, the resident can press “ok” and cause the “handled” event to get triggered. If within five minutes of a “bridge” event the “handled” event does not get generated, the “call 911” activity will issue a call to the police. The “911” feature is created independently of the other features, which is why the “911” and the “alert” feature are peers on the feature model.

Figure 5 shows the design of the “video alert” alternative feature. The “action” output event of the “presence detector” activity is connected to the “on” and the “send” inputs of the “film” and the “notify with video” activities. The “film” activity encapsulates cameras in the house. The “on” input of the “film” activity triggers the cameras in the house to capture video. The “send” input of the “notify with video” activity triggers a notification of a breaking to be send to the residents’ smartphone. At the same time, the “film” activity sends a video feed of what is happening in the house to the “notify with video” activity to display it to the residents’ smart phone. The residents can stop the video feed by pressing ok on their phone. The feature model in Figure 5 shows how the “phone alert” and “video alert” features are organized. Both features are grouped under the “alert” feature. The default alternative feature symbol was used for the “phone alert” feature and the alternative feature symbol for the “video alert” feature. The “video alert” feature has a mandatory dependency to the “film” feature. After the product line is completed, XANA generates the feature selection interface for end users to derive products for their spaces. The feature selection model is another view of the feature model that is more end user oriented. Optional and alternative feature symbols in the feature model are converted to check boxes and radio buttons at the feature selection view.

4.2.2. Application Derivation

During application derivation, end users are presented with the end user view of the feature model and the feature parameter table. The feature selection model represents the product line containing all available features that the end user can select for the smart space. The nodes of the feature selection model represent common, optional and alternative features. We selected checkboxes to represent optional features and radio buttons to represent alternative features. Because common features are always required, they are not selectable and are only shown in the feature model for information purposes. The feature parameter table captures each feature variables.

End users, based on their requirements and the devices and components installed in their spaces, select a feature combination from the feature model and configure the feature parameter table. As end users traverse the feature model a description of each feature is displayed along with the components that need to be available in the EUD architecture in order for the feature to be deployed. During feature deployment XANA will display an error if some of the components that are required to implement the feature are not available in the space. Based on the end user selections, XANA composes the application architecture, maps the application components to the EUD architecture component instances and deploys the product line instance to the EUD architecture.

Figure 6 shows how the surveillance team is configured for Frank’s environment. Frank selected the “911” optional feature and the “phone alert” alternative feature. The features require that TeC activities that support the “track”, “presence detector”, “notify by phone” and “call 911” activity instances and the hardware to support them is installed on Frank’s space. In the parameter section, the “track_res” parameter is
configured with Frank and Dora as the residents that need to be tracked. The “alert_res” parameter is configured with Frank as the person to be notified in case of a break in. When Frank finishes with the product line selections, he can select the “Generate Team” button so XANA can generate and deploy the customized surveillance application to the TeC architecture deployed in his space.

5. User Study

We conducted a preliminary user study to evaluate the usability of XANA. We contacted 9 individuals to take part in our user study. The participants were between 28 to 46 years old. 4 of the participants had Computer Science (CS) related background. The rest of the participants had moderate computer skills and were from different backgrounds (e.g. insurance agents, accountants, restaurant workers, hairdressers etc.). We refer to them as CS participants and Non CS participants in the rest of this paper. Each participant selection was targeted based on age and technical background in order to evaluate if XANA is generic enough to be adopted by participants or participants with different ages and technical backgrounds.

Each individual user study took around 30 minutes to complete and included a feature selection and a feature model modification session. The study started with a 5 minute overview of software product lines. This was followed by the feature selection session. The session began with a 5 minute presentation of an automobile product line where we demonstrated how feature selection works. After the participant was comfortable with the idea we presented him/her with the smart home product line. We asked the participant to perform four tasks. The first three tasks were feature selection related. The user had to select pre-specified features in the smart home product line. For the last task users had to complete a feature parameter table. If the participant did not have any difficulties with feature selection we proceeded to the second session of our study.

We started the feature model modification session with a 5 minute tutorial of how features are created. For the tutorial we used the automobile product line that we used in the feature selection session. After the tutorial we presented the participants with the smart home product line from the previous session in the design view. We asked the participants to complete three tasks. The participants had to add a pre-specified new feature to the product line for each of the tasks.

At the end of the user study, we prepared a set of open ended questions asking participants to provide feedback on the feature selection and feature model modification process. In order to assess the usability of XANA, we measured the time it took each participant to complete each of the tasks and the errors made. Based on our experience with the framework and other similar studies [7], [13] we established a baseline of expected completion times per task. Overall, everyone successfully completed both sessions of the user study with minor errors and within our expected completion times.

5.1. Feature Selection Results

On average it took each participant 89 seconds to complete the first three tasks. Even though CS participants performed better, there were no major differences between the two groups. During the feature selection session, 3 errors where made out of 27 tasks. 2 of the errors where related to the participants not selecting the parent dependency. Both participants selected the correct feature in the model but did not select the parent dependencies. The same participants selected the correct dependences on the other tasks. We asked the participants why they made the error and they said that they only focused on the child node. This issue can be easily resolved by enhancing XANA to not allow end users to select a child feature if the parent feature is not selected first. The last error was from one participant in the second task not paying attention and started selecting all the features that she would like her house to have. After we explained again the goal of the task which was to only select the task specified features the participant was able to correctly finish the task. All participants were able to complete the feature parameter table without errors. We found that users were entering the same values with different formats. In the user questionnaire all users found the tree structure easy to understand and a good grouping feature mechanism. One comment was that when a node in the feature selection model is selected, it would be easier to use if the node parent dependences in the tree where automatically selected. All of the participants found the checkbox and radio box notation easy to understand.

5.2. Feature Model Modification Results

On average it took each participant 113 seconds to complete all tasks during feature model modification. Overall there were 3 errors from a total of 27 tasks in the second session. All three errors were on the third task. On two of the errors even though the participants selected the correct feature symbol they added it in the wrong place in the feature model. When we asked them why they added the feature in the wrong place, they responded that it makes better sense for them there. After we explained the purpose of the task again the users added the feature to the right place. On the third error the participant added the feature to the right place but selected the incorrect feature symbol. When we brought it up to his attention he told us that didn’t pay attention to it. Concerning the error related to the participant choosing the wrong feature symbol, the same participant selected the correct feature symbol in the previous task. The other errors happened on the
same task and were the same type of error. This makes us believe that there might be an issue with the specific task. This makes us believe that there might be an issue with the specific task. In future versions of our experiment we will revise this task in order to verify that there are no issues with our language.

The questionnaire showed that all users found the symbol notation intuitive. Three Non CS participants suggested replacing symbols with letters. One of the participant suggested “O” for optional “S” for common, “D” for default and “A” for alternative. On the other hand, CS participants found that symbols were a good way of communicating the feature model concept and distinguishing it from feature selection.

Overall everyone successfully completed the feature model modification tasks with a low number of errors. Participants with no programming experience had more difficulty in remembering the symbols. All of the subjects found the tree structure easy to group features and a good way to represent the feature model. One of the participants suggested that it would be easier if there was a guided approach to create features and add them to the feature model.

5.3. User Study Extensions

This was a targeted user study to ensure the feasibility of our approach. One of the limitations of the user study was the participant selection. Even though there was diversity in age and professional background, the participant sample was small. Also the user study tasks were intentionally low in complexity so the participants could focus on the XANA feature semantics. At this stage, our primary goal of the study was to examine if XANA concepts for feature model creation, feature selection, feature symbols and parameter tables could be understood by end users. We plan to use this study as a baseline for future user studies. In particular we would like to expand our initial user study and measure the effort required to create the EUD SPL and feature selection versus the effort of having end users create the end user application using the EUD tool. In order to obtain statistically significant results we plan to increase the number of participants, utilize a random process for participant selection and expand the age range, gender ratio, technical and professional background of the participants. In addition we plan to have two groups of participants. The first group will consist of technical end users or domain experts that will be tasked with developing the SPL. The second group will consist of end users that will derive product line applications. Both groups will be tasked to create the same applications using the EUD tool by itself. Also we plan to increase the complexity of the tasks by having end users interact with the component designs that implement each feature.

6. Evaluation

To evaluate our approach we implemented several end user applications using the TeC end user development technology in the smart home domain. The applications dealt with home security, automation, owner notifications and energy conservation [11]. We designed and deployed the applications using TeC’s editor in both desktop and mobile platforms. X10 [14] hardware was used to control different devices in the smart home. We created different TeC wrappers for the X10 hardware and simulated devices that were not available. TeC wrappers were used to provide a common interface between the TeC architecture and the devices.

We extended TeC with XANA and used this to create an EUD SPL for smart homes with different common, optional and alternative features. The smart home SPL features we created relate to energy conservation, home security, water detection, emergency notification, home module replacement and home automation. Then we derived different SPL instances for the smart home. At this point in our evaluation phase, the application development in TeC, the TeC extension with XANA, and product line development were done by expert software developers and not end users.

7. Discussion

Our approach of creating an end user development product line for smart spaces has several benefits. First it offers an agile SPL approach that can be applied to existing EUD tools for smart spaces. Technical end users and domain experts are familiar with EUD tools for smart spaces. By augmenting these tools with product line constructs, as we did with TeC and XANA, end users can reuse components developed by technical end users and domain experts for their spaces. Second it provides the means for reducing the end user software development effort since end users can reuse software applications that others have created and configure them for their spaces. Third the framework takes into account the end user background and provides different user interfaces for SPL designers to create SPLs and end users to derive customized applications for their smart spaces. Fourth our approach can reduce component compatibility issues. SPL designers create the product line using abstract components. At deployment time, XANA coordinates with the smart home architecture to select the appropriate concrete components to realize the end user application. Finally our approach can improve end user application software quality through reuse of previously developed components.

Our approach also has some limitations. One limitation is that as the EUD SPLs grow in size there is potential for conflicts between features. For instance components and connectors required by one feature might contradict with what is required by another feature. To overcome this issue are investigating in incorporating SPL validation techniques [15] that SPL de-
signers can use to identify conflicts between features. Another limitation of our approach is that end users are restricted by the EUD SPL architecture. If end users want changes to the SPL, they need to involve the SPL designers. As a possible solution, we are exploring extending the XANA framework to permit variation points in the EUD SPL that allow end users the flexibility to add or modify components and devices in their smart spaces in a controlled way. Finally another limitation is that end users have to redepoly SPL applications as device and component changes occur to their spaces. We plan to incorporate dynamic product line techniques [22] so that the SPL can adapt to space changes.

8. Related Work

Several end user development tools have been proposed for smart spaces. The tools can be grouped in two general areas: Space reconfiguration and context aware tools. Space reconfiguration tools enable end users to control and combine functionality of devices. Puzzle [16], PIP [17], and FedNet [7] are some examples. Puzzle presents devices as jigsaw pieces for users to put together. PIP creates virtual appliances as a collection of devices. FedNet provides a tangible interface for device interconnection. Context aware tools create rules based on user context (activity, location, identity, time) and device functions. iCAP [13], GALLAG Strip [18], and TeC [11] are some representatives in this category. iCAP uses a rule-building interface for building applications. GALLAG Strip enables users to create context aware applications through a sequence of screens in a mobile device. XANA is as an extension of these tools to further simplify them and promote reuse. This paper presented a case study of XANA integrated with TeC. TeC is used to create the features and components in the product line. After the product line is created, end users derive applications for their spaces.

Software product line methods such as PLUS [19], KobrA [20], QADA [21], DSPLs [22] address the problem of modeling variability in product lines and provide processes to create them and derive applications from them. XANA builds on the ideas of software product lines but it targets end users instead of professional engineers. XANA can be thought as a lightweight product line framework for end users. For instance UML is one modeling language for product line design. XANA’s visual language is more restrictive than UML but it contains all necessary constructs for end user product line creation.

Current research on utilizing product lines for end users includes Perez et al. [23] and SimPL [24]. In SimPL domain engineers set up product line instantiation environments for end users to create software. It allows parametric creation of applications. The difference of SimPL with our framework is that XANA provides a visual language with SPL feature symbols to distinguish between common and variable components. Also SimPL according to the authors does not apply to the smart home domain. Perez et al. utilize variability engineering for professional engineers to cooperate with end users to capture end user requirements for smart spaces [23]. This is the closest to our work we found. Perez provides examples using jigsaw and programming by demonstration. We provided a case study in TeC. The differences of our work with Perez are: a) XANA is used to create product lines that end users can use to derive applications for their spaces than capturing end user requirements. b) We provide a consistent structure for product line creation and application derivation independently of the framework used.

9. Conclusions

As computing becomes ubiquitous and smart spaces become more complex, end users are expected to produce more software to suit their needs. Even though there are several end user development tools, not all end users have the technical skills to use these types of tools. Moreover, there have been several software quality issues with applications created by end users. One of the problems attributed to poor quality is lack of reuse. End users create their own applications without testing them properly. By adopting SPL concepts, software quality could be improved since software would be created once and be reused and tested by several end users. Thus, the same software components would be reused in different application configurations and would then be tested in different ways. Defects and software releases could be centralized instead of being managed by different end users.

The contribution of this paper is a lightweight end user development framework (XANA) that promotes reuse, its operational semantics and a visual language that supports different types of tools. XANA is based on software product line concepts while targeting end users. XANA provides different interfaces for product line creation and application derivation. For product line creation, the framework provides a visual language that allows SPL designers to specify functionality in common, optional, and alternative features. During application derivation, XANA presents end users with a simplified view of the software features that they can select for their smart spaces. Some of the benefits of XANA are that it targets both technical and non-technical end users, improves quality through the adoption of reuse, and enhances existing end user development tools for smart spaces. To demonstrate this, we extended the TeC development tool with XANA. XANA’s application derivation interface provided a view of the feature model and enabled end users to derive TeC applications for their smart spaces. In the preliminary user study we conducted, all participants were able to successfully complete feature selection.
and feature model modification tasks without any difficulties. The user background did not appear to have a big impact in the study results.

Additional work is needed to verify that XANA captures the necessary feature symbols that SPL designers need to create end user product lines. Additional work is also needed in the evolution of end user product lines. Some of the product line evolution issues that need consideration are: how defects are communicated to SPL designers, how SPL designers update the product line and how product line updates are communicated back to end users. Additional work is needed in an SPL validation framework to verify feature models and derived applications.

10. References


