Feature-oriented Re-engineering of Legacy Systems into Product Line Assets – *a Case Study*

Kyo Chul Kang, Moonzoo Kim, Jaejoon Lee, and Byungkil Kim

Software Engineering Lab. Computer Science and Engineering Dept.
Pohang University of Science and Technology, South Korea
{kck,moonzoo,gibman,dayfly}@postech.ac.kr
http://selab.postech.ac.kr/

Abstract. Home service robots have a wide range of potential applications, such as home security, patient caring, cleaning, etc. The services provided by the robots in each application area are being defined as markets are formed and, therefore, they change constantly. Thus, robot applications need to evolve both quickly and flexibly adopting frequently changing requirements. This makes software product line framework ideal for the domain of home service robots. Unfortunately, however, robot manufacturers often focus on developing technical components (e.g., vision recognizer and speech processor) and then attempt to develop robots by integrating these components in an ad-hoc way. This practice produces robot applications that are hard to re-use and evolve when requirements change. We believe that re-engineering legacy robot applications into product line assets can significantly enhance reusability and evolvability.

In this paper, we present our experience of re-engineering legacy home service robot applications into product line assets through feature modeling and analysis. First, through reverse engineering, we recovered architectures and components of the legacy applications. Second, based on the recovered information and domain knowledge, we reconstructed a feature model for the legacy applications. Anticipating changes in business opportunities or technologies, we restructured and refined the feature model to produce a feature model for the product line. Finally, based on the refined feature model and engineering principles we adopted for asset development, we designed a new architecture and components for robot applications.

1 Introduction

Home service robots utilize various technology-intensive components such as speech recognizers and vision processors to offer services. As markets for home service robots are still being formed, however, these technical components undergo frequent changes and new services are added and/or existing services are often removed or updated to address changing needs of the users. To compete in this rapidly changing market, robot manufacturers should be able to evolve robot products quickly with a minimal cost. The home service robot industry has strong needs for software devel-

opment framework with which applications can be evolved easily. This situation makes software product line ideal for the home service robot industry.

Due to limited development resources, robot developers focused on technology intensive components at an early stage of product development without careful consideration of how software applications would evolve with changing requirements. Without a fore-thought architectural consideration, initial products have often been developed by integrating technology components in an ad-hoc way. Consequently, products suffered from feature interaction problems and maintenance of applications became costly. Re-engineering legacy robot applications into product line assets can enhance the competitive power of robot products by both decreasing development cost and increasing flexibility of robot applications. Jean-Marc at al [1][2][6] suggest an architecture-centric re-engineering process for initial product line asset recovery. This approach emphasizes a software architecture as a key to recovery of domain concept and relations. Bosch at al [3][4] consider a feature model as a core for creating product line assets from legacy products. These studies, however, do not suggest concrete design principles or guidelines for creating product line assets with adaptability.

In this paper, we describe our experience of re-engineering home service robot applications into product line assets via a feature-oriented methodology that is based on concrete principles and guidelines [5]. First, we extracted components and architectural information from legacy robot applications [7]. Second, based on the recovered information and domain knowledge, we discovered and modeled features of the robot applications. Anticipating future evolution of applications by considering potential business opportunities and technology changes, we refined the feature model adding additional features and variability information [8]. Finally, based on the refined feature model and three engineering principles we adopted to develop evolvable assets [9], we designed a new architecture and components for the product line. This re-engineering approach is depicted in Fig.1.

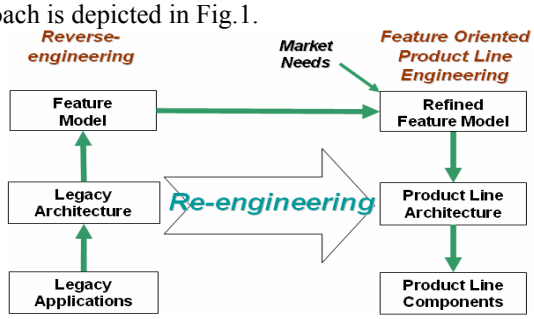


Fig. 1. Overview of re-engineering process

Sect. 2 gives an overview of home service robots. Sect. 3 explains the process of recovering architectural information from legacy applications. Sect. 4 describes recovery and refinement of a feature model from the legacy applications. Sect. 5 illustrates redesign of an architecture and asset components based on the refined feature model using the engineering principles we adopted. Sect. 6 validates the re-engineered product line assets. Finally, Sect. 7 describes the lessons learned from this project and Sect. 8 summarizes the paper and suggests future works.

2. Background on the Home Service Robot (HSR)

In this section, we briefly overview services of the home service robot (HSR) whose applications we re-engineered into product line assets. HSR is developed for daily home services such as home surveillance, cleaning, etc. From the HSR manufacturer, we received high level specifications of required HSR services such as “Call and Come” (locate and come to the user), “User Following” (continuously follow the user), “Security Monitoring” (home surveillance), and “Tele-presence” (control HSR remotely), etc. [1] In addition, we received two separate HSR applications each of which implements the “Call and Come” service and “User Following” service respectively. Of these primary services of HSR, we explain “Call and Come” and “User Following" services in detail.

*** Call and Come (CC)**

This service first analyzes audio data sampled from microphones attached to the surface of the robot and detects predefined sound patterns (e.g., hand clap or voice command). Currently, there are two commands “come” and “stop”. Once a “come” command is recognized, the robot detects the direction of a sound source. Then, the robot rotates to the direction of a sound source and tries to recognize a human face by analyzing video data captured through the front camera. If the caller's face is detected, the robot moves forward until it reaches within one meter from the caller (distance from the caller is measured by a structured light sensor). A “Stop” command simply makes the robot stop. If the following operation such as command recognition, sound source detection, or face recognition fails, CC resets to an initial state and waits for a new command.

*** User Following (UF)**

The robot uses a front camera and a structured light sensor to locate the user. Once UF is triggered, the robot constantly checks the vision data and sensor data from the structured light sensor to locate the user. The robot keeps following the user within one meter range. If the robot misses the user, the robot notifies the user by generating an audio message and UF terminates. The user may give a “come” command to let the robot recognize the user and restart UF.

Based on the given specifications and information extracted from the two legacy applications, we recovered a preliminary feature model covering both applications. The legacy HSR applications hard-coded most features without considering variation points for future extension or refinement. For example, the legacy HSR application has features such as “Face Detection Method” and “Object Recognition with SL” for user detection and user tracking. These features, however, do not have variations but have fixed implementations. For example, “Face Detection Method” is implemented based on “Color-based” method, not allowing other detection techniques to be adopted. For more detailed of features supported by the legacy HSR applications, see Fig. 5.

[1] For more information on HSR services and hardware, see [9]

3. Information Extraction from Legacy HSR Application

In this section, we explain how architectural information was extracted from the legacy applications and what potential problems were with the architecture.

3.1 Reverse Engineering Process

Fig. 2 describes the process of recovering a conceptual architecture as well as a process architecture from legacy applications.

1. From legacy applications, we obtain object relationship diagrams (see Fig. 3) mechanically, i.e., using the Rational-Rose[2] tool.
2. Based on the extracted object relationship diagram, we determine objects which constitute services (e.g., CC and UF services). This step needs heuristics based on domain knowledge and additional data flow analysis. Then, we identify *operational units* that the service consists of, by analyzing method invocations and data flows. By assigning operational units into architectural components, we recover a conceptual architecture.
3. From the object relationship diagram and identified service/operational units, we determine which objects (i.e. active objects) take initiative of invoking other objects' operations by creating processes/threads. Then, we identify interactions between active objects via a control flow analysis. By capturing these interactions between active objects, we recover a process architecture which shows assignment of software components to processes or thread synchronization relations.

How this process was applied to CC is explained in the following subsections.

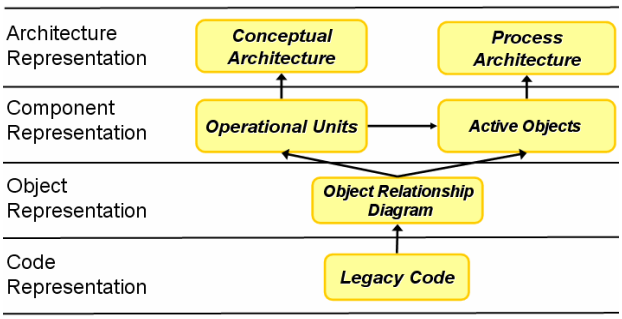


Fig. 2. Recovery of conceptual architecture and process architecture

3.2 Recovery of Operational Units

Fig. 3 illustrates recovery of operational units from the object relationship diagram for CC. Using functional cohesion as a criterion, we classified operational units into

[2] Rational-Rose is a trademark of IBM corporation.

three categories – *sensor (input), controller (coordination), and actuator (output).* Using these categories as a guide, we identified five operational units as follows.

- sensor units: “Face Detection”, “Clap Recognition”, and “SL Sensing”
- a controller unit: “CC Command Controller”
- an actuator unit: “Actuator Controller”

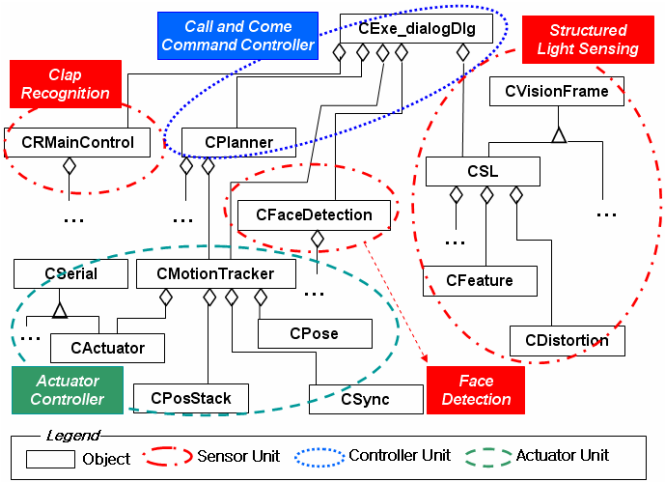
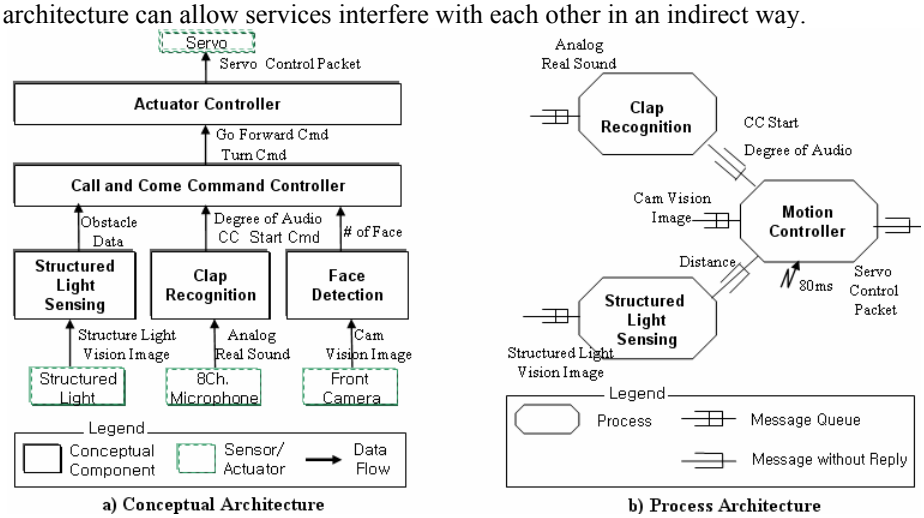


Fig. 3. Recovery of operational units for CC

3.3 Recovery of Conceptual Architecture and Process Architecture

Through an additional data flow analysis, the identified operational units are configured into the conceptual architecture depicted in the Fig. 4.a). This conceptual architecture is hardly adequate for multi-service robots because all service units (e.g. CC Command Controller) can access and control “Actuator Controller” directly. This architecture can allow services interfere with each other in an indirect way.



a) Conceptual Architecture **b) Process Architecture**

Fig. 4. Recovered conceptual architecture and process architecture

To recover a process architecture, we identified three active objects from the object relationship diagram depicted in Fig. 3 by detecting process creation code – `CEXE_dialogDlg`, `CRMainControl`, and `CSL`. These objects create three processes "Motion Controller (MC)" (consisting of "CC Command Controller", "Face Detection", and "Actuator Controller" operational units), "Clap Recognition (CR)" ("Clap Recognition" unit) and "SL Sensing (SLS)" ("Structured Light Sensing" unit) respectively as depicted in Fig.4.b). MC receives data such as the distance to an obstacle and the direction of clap sound from SLS and CR respectively. MC determines the moving direction based on these data. Thus, without a smart control logic in MC, feature interaction between CR and SLS may happen because both processes can control MC at the same time.

4. Refined Feature Model of HSR Product Line

In this section, we describe a refined feature model of HSR. First, we extracted features from the legacy application implementing CC service, which are indicated in bold font in Fig. 5. Newly added features and refined features are indicated in italic font in Fig. 5. The detailed explanation of the refined feature model is as follows.

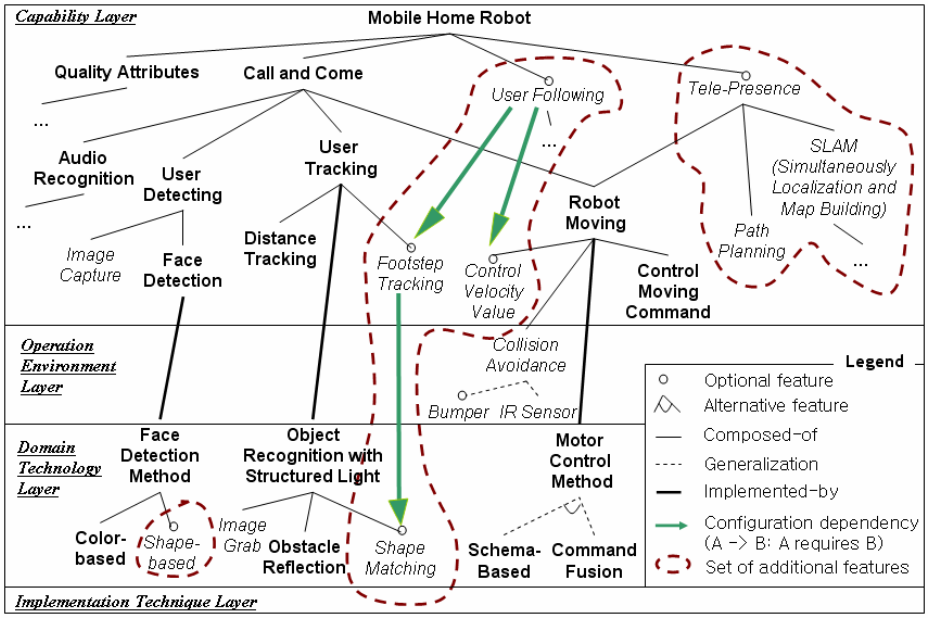


Fig. 5. Feature model for SH100 including CC service

First, we added new services targeted for different markets. For example, HSR supporting only CC service can be produced for a low-end market as a delivery robot, while HSR with CC, UF, Tele-presence, and Security Monitoring services can be produced for a high-end market as an intelligent home agent. Based on the legacy

feature model for the CC service, we created a new model by adding features for new services, operations, and domain technologies, and also dependency relationships between features. Newly added services require operational features not included in the original feature model. For example, newly added UF service needs to follow user's footsteps ("Footstep Tracking"). In addition, to follow the user smoothly, UF service controls HSR in a velocity oriented way via "Control Velocity Value" (e.g. set the velocity of left wheel as 1 m/s, and the right wheel as 0.8 m/s). Furthermore, a new operational feature may require new domain technologies. For example, "Footstep Tracking" requires "Shape Matching" in order to recognize user's footsteps.

Second, we refined the feature model by including optional features to accommodate anticipated changes. For example, in the legacy CC application, "Face Detection Method" used only a color-based detection algorithm. We refined this feature by adding an optional feature "Shape-based" for its improved accuracy adequate for high-end markets, but at the cost of high computational resources.

Third, due to the advances of technologies, some features considered as important capabilities can simply be supported by the operational environment as SoC (System On Chip) or by OS. In the legacy CC application, "Collision Avoidance (CA)" feature was implemented in software and placed in the Capability Layer. We moved CA to the Operation Environment Layer because of CA SoCs available in the market.

5. New Architecture Design of HSR

One of the quality attributes with the new architecture is its *flexibility* in adding, removing, and/or replacing components as products evolve. For this purpose, we adopted C2 architectural style [10] for its substitutability of components. Also, we enforced *1:N* mapping from features to components whenever possible for easy inclusion/exclusion of features into/from products. Furthermore, through an analysis of legacy applications [11] and the refined feature model in Fig. 5, we decided to adopt three engineering principles in redesigning the architecture of HSR (for details on these principles, see [9]).

First, the legacy architecture intermixed control components with computational components, which caused difficulty in analyzing behaviors of applications. Therefore, we proposed the first principle – *separation of control aspects from computational aspects.* By separating the control plane which consists of control components from the data plane with computational components, we could separate data flows from control flows, thus making it possible to visualize and analyze behaviors of the system. As a consequence, addition/removal of components becomes easier because responsibilities of each component become clear.

Second, we aimed to minimize ripple effects caused when services are added or removed - simple integration of new services, without consideration of how features should be related with each other, has easily led to feature interaction problems. The legacy architecture did not provide careful coordination among service components, thus resulted in feature interaction problems when a new service was added. To address such problems, we proposed the second principle - *separation of global behaviors from local behaviors.* Service components are separated to be executed *locally*,

i.e., independently from other service components. Therefore, effects from addition/removal of components to other components are localized, which helps implementing variation points. The coordination responsibility among different service components is assigned to a special component called *Mode Manager* which controls global system behavior such as interaction policies between service features.

Finally, we found that there existed hierarchy between some variable features. For example, "Object Recognition with SL" feature has three sub-features – "Image Grab", "Obstacle Reflection", and "Shape Matching" (see Fig. 5). "Image Grab" simply captures SL images whereas "Obstacle Reflection" detects objects in front of HSR by analyzing the SL images obtained by "Image Grab". "Shape Matching" works more sophisticatedly by analyzing object images obtained from "Obstacle Reflection" to recognize user's legs (e.g., footsteps). Therefore, we made three component layers corresponding to these variable features according to the third principle - *layering in accordance with data refinement hierarchy*. Different services may request operations from different layers of a single component. By adopting a layered architecture for computational components, addition/removal of variable features in the Domain Technology Layer could be implemented cleanly because the layered architecture provides well-defined interfaces between layers.

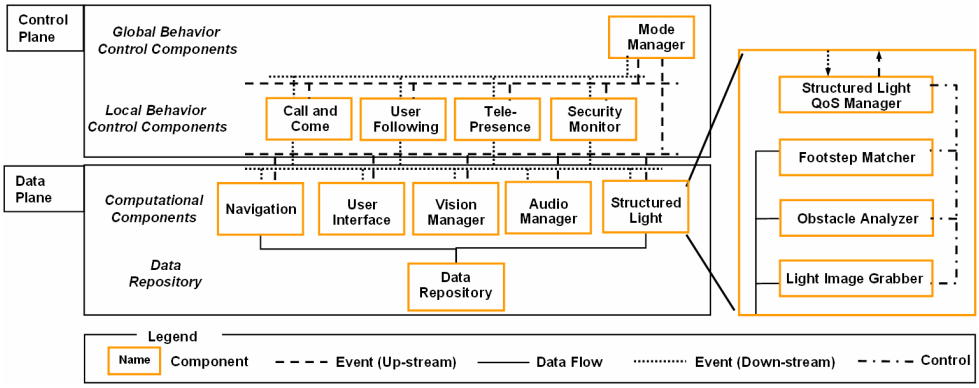


Fig.6. New architecture for HSR

Fig.6 illustrates the new architecture designed according to the three re-engineering principles.[3] First, we identified four control components: CC, UF, Telepresence, and Security Monitoring. And we identified five computational components: Navigation, Structured Light, User Interface, Vision Manager, and Audio Manager. Mode Manager was specified to control global behavior of HSR by receiving all up-stream events and managing the control components. Most computational components read raw input data from sensors and process them to generate outputs to

[3] This architecture reflects typical software architecture of embedded systems (especially application layer) such as network gateways or vehicle controllers which distinguish control data from computational data.

other components. The generated outputs are transferred to the control component through a data connector/bus.

Based on the new architecture, we designed components with a macro-processing mechanism (to incorporate variable features) [12]. In addition, we extracted sub-components from the existing code through refactoring techniques [13]. Fig. 7 illustrates the structured light component. The left part of Fig.7 shows a layered template for computational components and the structured light component instantiated from the template. The legacy structured light component was implemented as a long procedural function. Thus, we extracted reusable portion of the function into "Footstep Matcher", "Obstacle Analyzer", and "Light Image Grabber" components. These layered components were instantiated for the selected features using a component specification [14].

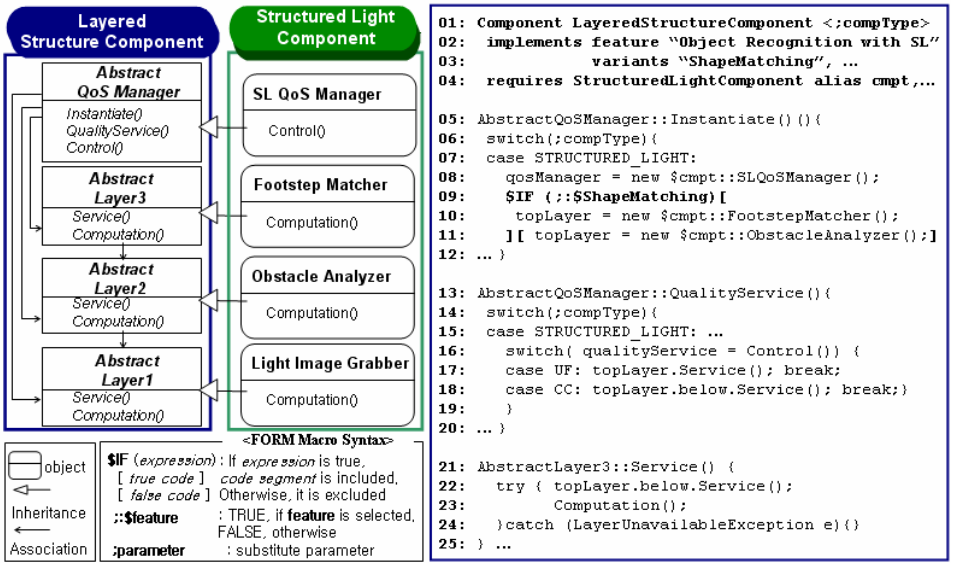


Fig. 7. A design object model and component specification

Lines 1-4 of the right part of Fig. 7 specify instantiation of `LayeredStructureComponent` implementing "Object Recognition with SL" feature (with variable feature "ShapeMatching") from `StrcutredLightComponent`. Lines 5-12 describe how structured light and vision manager are instantiated. Especially, lines 9-11 specify that if a variant feature "Shape Matching" is selected, the instantiated component will have "Footstep Matcher" as its topmost layer; otherwise, "Obstacle Analyzer" as its topmost layer. Lines 13-20 illustrate how a service is selected for the service requestor. For example, at line 17, if UF requests service of structured light components, the service of topmost layer (i.e. "Footstep Matcher") should be provided (with an assumption that "Footstep Matcher" feature is enabled). Lines 21-24 show a service chain between layers.

6. Validation of Product Line Assets

We have generated HSR applications using re-engineered product line assets. First, without difficulty, we have instantiated two applications supporting CC and UF respectively by selecting features required by the services. We could check that new applications worked successfully according to the given service specifications. For these two applications, Mode Manager does not enforce control on global behaviors because the HSR applications run only a single service.

Then, we have instantiated an application supporting both CC and UF services. The CC and UF services share computational components. Concurrent accesses to the computational components except “Navigation” did not cause any feature interaction problem between the CC and UF services; operations requested to the computational components by CC and UF are mainly reading analyzed data, not updating data. In addition, the layers accessed by the two services are different. For example, CC accesses the “Obstacle Analyzer” layer while UF accesses the “Footstep Matcher” layer of the “Structured Light” component. Operations requested by UF and CC to ”Navigation”, however, are mostly for controlling actuators. Thus, to prevent a feature interaction problem, Mode Manager coordinated CC and UF using a priority scheme. Code modification required for priority enforcement was not obstructive because CC and UF components except Mode Manager did not need to be modified. Therefore, we have shown that the re-engineered product line assets for HSR are suitable for creating applications of the home service robot.

7. Lessons Learned

In this section, we describe lessons we have learned from this re-engineering project.

7.1 Importance of Pre-planned Asset Integration

Hardware-oriented or technology-oriented organizations usually consider product development/instantiation as a last-minute task that can be achieved by simply integrating technology-intensive components. Without a fore-thought architectural consideration and component integration strategies, however, products often suffered from feature interaction problems and maintenance of applications became costly.

In this case study, we could alleviate these difficulties by providing an architectural framework based on the refined feature model and engineering principles we adopted for asset development. In addition, the explicit mapping between features and architectural components made the inclusion/exclusion of features visible. We also observed that a feature model could play a central role in identifying relationship between pre-existing features and new features. For example, for the addition of "User Following" feature, the feature model in Fig.5 shows additional new features such as “Footstep Tracking” and their relationships with the features of the legacy applications.

Based on the feature analysis results, we could determine component integration scheme. For the integration of the "Footstep Tracking" feature, for instance, the component that implemented "User Tacking" was modified to accommodate the "Footstep Tracking" feature and the modified component could confine the variations between "Distance Tracking" and "Footstep Tracking" by providing a common interface.

7.2 Benefit of a Feature Model in Architecture Layering

Through the case study, we found that the feature model provided a useful information for identifying layers in the component architecture. The feature model has features representing different levels of computation. Especially variation points show services of different levels. For example, “Shape Matching”, “Obstacle Reflection”, and “Image Grab” features (see Fig. 5) are used for UF, CC, and Tele-presence services respectively. These features altogether represent computational hierarchy, i.e., “Shape Matching” uses result from “Obstacle Reflection” and “Obstacle Reflection” from “Image Grab”. Accordingly, these features are implemented as a “Footstep Matcher” layer, an “Obstacle Analyzer” layer, and a “Light Image Grabber” layer of the structured light component (see Fig. 7). Similarly, we found that “Face Detection Method” feature also had a hierarchy among its sub-features and, thus, corresponding “Vision Manager” component was built as a layered structure. Therefore, layering based on the feature model was very helpful for creating component architecture for product line engineering.

7.3 Analysis Aid of Process Architecture

Process architecture can help finding possible feature interactions among concurrent processes. For example, from the process architecture in Fig. 4.b), we could guess that MC might suffer feature interaction problems due to concurrent input data from CR and SLS (see Sect. 3.3). Furthermore, process architecture also helps analyzing the legacy application design. For example, UF service implemented in the legacy application does not use the front camera, not following the UF service specification (see Sect. 2). We could find the reason based on the process architecture. In order to utilize the front camera for UF, the front camera should capture images continuously to detect user’s face. The “Face Detection” operational unit in the legacy application, however, was a sequential component of MC, not a separate process running concurrently (see Fig. 4.b)). That was the reason why legacy UF application did not use the front camera.

8. Conclusion

In this paper, we describe re-engineering legacy home service robot applications into product line assets via a feature-oriented method. We believe that feature-oriented re-

engineering approach can help robot manufacturers to take advantage of product line framework – decrease in development cost and increase in application flexibility.

As a future work, based on the re-engineered HSR product assets, we plan to study evolution of HSR product line assets and evaluate both weaknesses and strengths of the current product line assets. Secondly, we will study and develop guidelines for evaluating product line assets.

References

1. DeBaud, J.M., Girard, J.F.: The relationship between the Product Line Development Entry Points and Reengineering, 2nd International Workshop on Development and Evolution of Software Architectures for Product Families, LNCS 1492, pp. 132-139, (1998)
2. Bayer, J., Girard, J.F, Wuerthner, M., DeBaud, J.M., Apel, M.: Transitioning Legacy Assets to a Product Line Architecture, 7th European Software Engineering Conference (ESEC/FSE'99), LNCS-1687, pages 446-463, (1999)
3. Bosch, J., Ran, A.: Evolution of Software Product Families, Software Architectures for Product Families: International Workshop(IW-SAPF-3), LNCS 1952, pp. 168-183, (2000)
4. Maccari, A., Riva, C.: Architectural Evolution of Legacy Product Families, Software Product Family Engineering: 4th International Workshop (PFE-4 2001), LNCS 2290, pp 64-69, (2002)
5. Kang, K., Lee, J., Donohoe, P.: Feature Oriented Product Line Engineering, IEEE Software, 19(4), July/August, pp. 58-65, (2002)
6. Eixelsberger, W., Kalan, M., Ogris, M., Beckman, H., Bellay, B., Gall, H.: Recovery of Architectural Structure: A Case Study, 2nd International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families LNCS Vol. 1429. Springer-Verlag, Berlin Heidelberg New York (2002)
7. Bergey, J., O'Brien, L., Smith, D.: Option Analysis for Reengineering (OAR): A Method for Mining Legacy Assets (CMU/SEI-2001-TN-013). Pittsburgh, PA:Software Engineering Institute, Carnegie Mellon University (2001)
8. Lee, K., Kang, K., Lee, J.: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In: Gacek, C. (eds.): Software Reuse: Methods, Techniques, and Tools.Lecture Notes in Computer Science, Vol. 2319. Springer-Verlag, Berlin Heidelberg
9. Kim, M., Lee, J., Kang, K., Hong, Y., Bang., S.: Re-engineering Software Architecture of Home Service Robots: A Case Study, International Conference on Software Engineering, Missouri, USA, pp.505-513, (2005)
10. Medvidovic, N., Taylor, R. N.: Exploiting architectural style to develop a family of applications, Software Engineering. IEE Proceeding, Vol. 144, No 5-6. October/December (1997)
11. Lago, P., Vliet, H.: Observations from the Recovery of a Software Product Family, Software Product Line Conference 2004, LNCS Vol 3154 Springer-Verlag, Berlin Heidelberg New York (2004)
12. Basset, P.G.: Framing Software Reuse: Lessons from the Real World. Prentice Hall, Yourdon Press (1997)
13. Fowler, M., Beck, K., Brant.,J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code, Addison-Wesley (2000)
14. Bosch, J., Hogstrom, M.: Product Instantiation in Software Product Lines: A Case Study. Second International Symposium on Generative and Component-Based Software Engineering LNCS 2177 (2001)