Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Applying Software Product Line Technology to Prototyping of Real-Time Object Tracking

Chia-Chu Chiang and *Bill Marshall

Department of Computer Science
*Department of Theatre and Dance
University of Arkansas at Little Rock
2801 South University Avenue
Little Rock, Arkansas 72204, USA
{cxchiang | wmmarshall}@ualr.edu

***Abstract*—In this paper, a software product line (SPL) architecture that explicitly captures common features of real-time object tracking systems using Cricket wireless sensors is presented. A software product line process is also presented including user requirements, architecture design, component development, and systems integration. The focus lies on the application of SPL to object location tracking, such as supply chains and transportation. Thus, this paper introduces three prototypes of the SPL member system including shop navigator systems, low/no visibility navigation systems (LVNS), and games for mazes where each one is created from the software product line architecture. The prototypes are experimented with and lessons are learned.**

Keywords—cricket, incremental development, indoor positioning, object tracking, rapid prototyping, software architecture, software architectural styles, software product line (SPL)

I. INTRODUCTION

Traditional approaches to software reuse have proved to be ineffective when providing small-grained reuse [1]. One approach to large-grained reuse is the software product line approach. This approach has been used in industry to reduce costs, increase productivity, and improve the quality of the systems. A software product line is a family of software systems that share some common functionality that meets the specific user requirements of an application domain [2]. Each system derived from the software product line shares the common components of the SPL where the components can be tailored through various mechanisms such as polymorphism, inheritance, or parameterization [3].

Prototyping allows users to experiment using quick and cheap systems and learn lessons from running the system [4]. Once the user requirements are met the prototype can be thrown away. In this paper, a software product describing the prototyping of object tracking using wireless sensors will be applied. The detailed design of the SPL will be discussed when the SPL is described using UML (Unified Modeling Language).

The remainder of this paper is organized as follows. In the next section, related work will be presented. Section 2 formulates the requirements of object tracking. Section 3 describes the detailed design of the prototyping of the SPL architecture. Section 4 demonstrates the reusability of the SPL architecture by presenting three prototypes. The evaluation of the development based on the criteria proposed by Matinlassi is presented in Section 5. The final section summarizes the paper.

II. RELATED WORK

The software product line approach to software reuse uses domain knowledge to identify commonalities of a family of software products and separate the commonalities into s baseline for all products in the family. Studies have shown that organizations using this approach can improve productivity, increase quality, reduce costs, and shorten the time to market [1, 5, 6].

Chiang and Lee [7] describe the reengineering of a software product line architecture used for developing a family of reverse engineering commercial systems. Due to software evolution, the legacy software product line architecture was reengineered to handle distribution and interoperability. The reuse of the software product line architecture shortens product-specific development time and improves the quality of product. Chiang and Lee also present the problems and solutions of reusing the software product line architecture in an industrial setting.

Shao and Lee [8] apply the software product line to the simulation modeling of emergency response facilities. The authors present the user requirements, analysis, and design of a family of simulation systems used in emergency response facilities. UML was used to describe components and capture the characteristics and constraints of the components along with relationships among the components. A hospital ER simulation system was derived from the SPL of an emergency response facility. With this hospital emergency room simulation model, users can learn lessons from simulations to help them make better decisions in real emergency scenarios.

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

III. SOFTWARE PRODUCT LINE ENGINEERING

Like other software development processes, there are several software product line processes available to support development [9]. The process we will use in this project is the ESAPS (Engineering Software Architectures Processes and Platforms for System Families) process [10]. The steps found in the process include software product line user requirements, software product line architecture design, software product line component development, and software product line systems integration as shown in Figure 1.

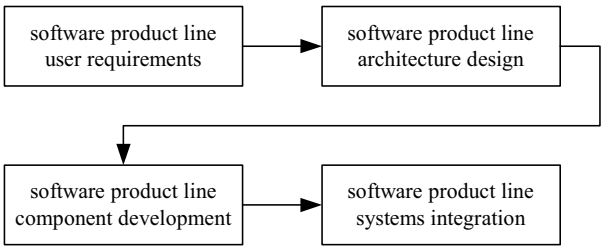

Figure 1. ESAPS software product line process.

Software product line user requirements are used to determine the user requirements of the product line. The purpose of this step is to determine what to build, not how to build. Since the product line requirements span in a family of products, the commonalities need to be located, extracted as a core asset, and maintained separately. Variations in the requirements can be instantiated for individual products.

Based on the product line user requirements, a software product line architecture design can be created from a family of architectural styles. An architectural style defines a family of systems in terms of a pattern of structural organization. More specifically, an architectural style defines the type of components and connector types and sets constraints on how they can be combined. There are several architectural styles, including repository, dataflow, call and return, object oriented, and layered, available for software architects to develop software product line architectures [11].

The software product line components specified in the software product line architecture are implemented with the required variables to fulfill expected software product line user requirements. The resultant components can either be part of the core assets of the product line or be developed for a specific product.

Software product line systems integration combines software components and subsystems into an integrated whole [6]. There are two steps of systems integration including the installation of core assets into the core base layer followed by the building of individual products in a family on the core base.

A. User Requirement of Object Tracking

In the product line user requirements, Cricket wireless sensors are used to track moving objects originally developed at MIT [12] and now distributed by Crossbow [13]. The Cricket wireless sensors have physical characteristics that may impose constraints on the architecture.

The inputs, outputs, and user interface requirements for the object tracking system SPL are identified as follows. Inputs include the distances from the Cricket wireless sensors. The distances provided by the sensors may not be accurate due to external noises in the setting environment. Outputs include the Cartesian coordinates in (x, y, z). User interface requirements define the need of a two-dimensional display of the locations of mobile objects. Thus, the tracking of moving objects is required to be continuous and real-time.

1) Use Case Modeling: Figure 2 depicts the SPL use case model for the object tracking system.

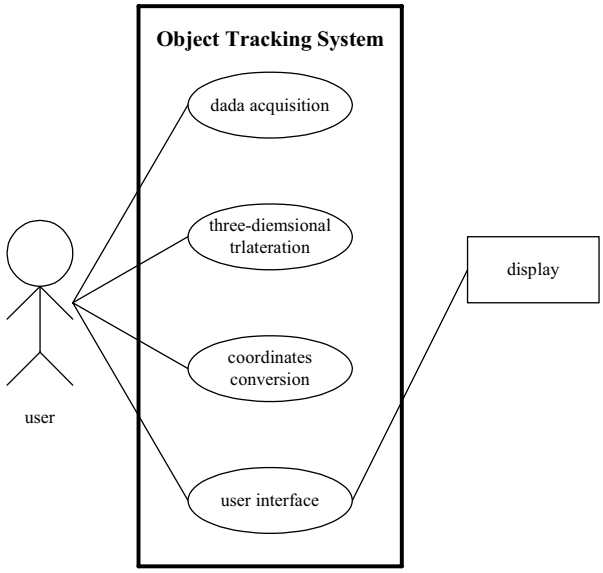

Figure 2. Use case model for object tracking software product line.

Basically, a user turns on the sensors and runs the system. The data acquisition will interface with the Cricket daemon to obtain the distances using the identities of the beacons. The data will be filtered for noises in the environment. Valid data will be passed to the three-dimensional trilateration function to compute the location of the moving object in the Cartesian coordinates (x, y, z). Due to the varying requirements of different products, the coordinate conversion function will be left open for instantiation. The Cartesian coordinates in (x, y, z) will then be converted to other coordinate systems depending on each specific product. The user interface is also left open for instantiation. Thus, there are two variations in the requirements of the object tracking system. The variations may be substantial. For example, a specific product might not need to covert the Cartesian coordinates in (x, y, z) into another coordinate system. Thus, the variation of coordinate conversion can be mapped to null.

The details of the user case model of the system are provided in Table 1, Table 2, Table 3, and Table 4.

TABLE 1. USE CASE FOR DATA ACQUISITION

Use Case ID	1
Use Case Name	Data Acquisition
Summary	The data acquisition function interfaces with the Cricket daemon and reads in the distances between the object and the beacons.
Actors	User
Trigger	Turn on the listener and beacons
Pre-conditions	• Listener on • Beacons on
Description	1. User starts to run the system. 2. The system starts to collect the distances with the identities of the beacons.
Post-conditions	• Distances with the beacons' identities
Exceptions	• Listener off • Beacons off

The variation point in the data acquisition use case is the number of beacons used in this scenario. The default value is 3.

TABLE 2. USE CASE FOR THREE-DIMENSIONAL TRILATERATION

Use Case ID	2
Use Case Name	Three-Dimensional Trilateration
Summary	The three-dimensional trilateration function reads in the distances with the identities of the beacons and computes the position in the Cartesian coordinates (x, y, z) of the object.
Actors	None
Trigger	None
Pre-conditions	• Valid distances d_1, d_2, and d_3 • Given locations of beacons (x_1, y_1, z_1), (x_2, y_2, z_2), and (x_3, y_3, z_3)
Description	1. Solve the following equation to obtain the position of the object, (x_p, y_p, z_p), where $d_1 = \text{square-root}((x_1 - x_p)^2 + (y_1 - y_p)^2 + (z_1 - z_p)^2)$ $d_2 = \text{square-root}((x_2 - x_p)^2 + (y_2 - y_p)^2 + (z_1 - z_p)^2)$ $d_3 = \text{square-root}((x_3 - x_p)^2 + (y_3 - y_p)^2 + (z_2 - z_p)^2)$. 2. The system starts to collect the distances with the identities of the beacons.
Post-conditions	• Location of the object, (x_p, y_p, z_p)
Exceptions	• Negative d_1, d_2, and d_3

The variation point in the three-dimensional trilateration use case is the number of sensors transmitting the distances with the identities back to the system. The default value is 3.

TABLE 3. USE CASE FOR COORDINATE CONVERSION

Use Case ID	3
Use Case Name	Coordinates Conversion
Summary	The coordinates conversion translates the Cartesian coordinates (x, y, z) into another coordinate system.
Actors	None
Trigger	None
Pre-conditions	• Valid Cartesian coordinates (x, y, z)
Description	1. Convert the Cartesian coordinate system into another coordinate system using a simple translation formula.
Post-conditions	• Translated coordinates
Exceptions	• Invalid translated coordinates

The variation point in the coordinate conversion use case is the selection of a coordinate system from a menu or the entry of a coordinate system by the user.

TABLE 4. USE CASE FOR USER INTERFACE

Use Case ID	4
Use Case Name	User Interface
Summary	The user interface function interfaces with the user to determine a scenario to be experimented and draws the floor plan on the screen.
Actors	User
Trigger	User runs the system
Pre-conditions	• System running
Description	1. User starts to run the system. 2. System prompts for user id and password. The user enters user id and password. The system validates the user id and password. 3. The system draws a floor plan on the screen.
Post-conditions	A floor plan displayed
Exceptions	• wrong floor plan chosen

The variation point in the user interface use case is the selection of user interface type from the menu, or an alternative is for the user to enter a user interface type code.

B. Architecture Design of Object Tracking

Since the software product line architecture proves the core components to the products, the software product line architecture determines the success of any software product. In this project, we decided to use the layered architectural style to develop the software product line architecture. A layered architectural style organizes its components hierarchically where each layer provides services to the layer above it. Figure 3 shows how the layers interact with each other.

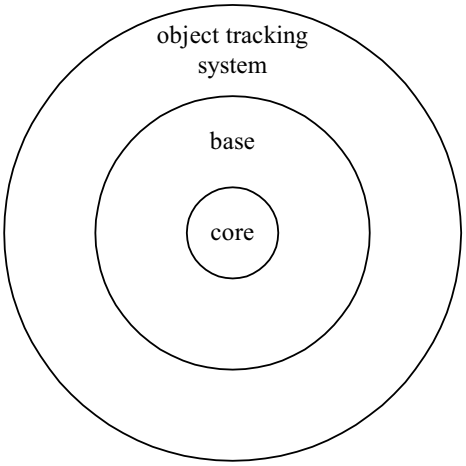

Figure 3. A sample layered architectural style.

The core layer includes component interfacing with the different sensors that are defined by hardware connections. The base layer includes components that need the services of the components in the core layer. This base layer includes components such as three-dimensional triangulation and coordinate conversion. The system layer is the layer interfacing with the user or another larger system. Components used for user interface are suitably deployed in this layer.

The use of the layered architecture for the software product line was motivated by the architecture's increased levels of abstraction [11]. Since each layer only interacts with at most two layers, above and below, changes to the components of one layer affected at most two other layers. This impact can be reduced if variations are designed into the architecture through parameterization and instantiation mechanisms.

C. Component Development of Object Tracking

The components specified in the software product line architecture are implemented with the required variability to fulfill the expected user requirements of object tracking.

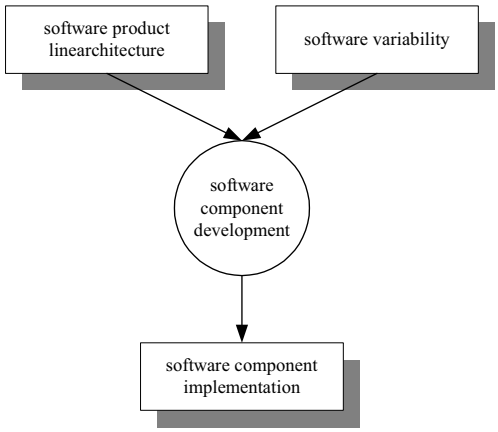

Figure 4. Component development activities.

Software component development includes the following steps: component requirements engineering, component design, and component implementation. From the software product line architecture and the variations in the requirements within the family of products, component requirements are obtained and fed into the design of components. The resultant components can then be a component used in the core layer, the base layer, or the system layer.

D. Systems Integration of Object Tracking

Software system integration is the combining of software components into an integrated whole. The approaches of systems integration include top-down, bottom-up, and mixed. When using a layered architecture, we suggest the bottom-up approach for integration. The components in the core layer are first installed into the software platform. The components in the base layer are then installed on the core layer. The components in the system layer are finally deployed on the base layer. This approach also makes incremental testing very efficient. The components in the core layer are first tested for correctness. The components in the base layer are then added to the testing followed by the system components. The bottom-up approach with the incremental testing approach decreases the risk of experiencing problems at the end of the software development process. Furthermore, the bottom-up integration approach allows for the quick creation of a working prototype for further experimentation. The other integration approaches do not provide this benefit.

IV. Object Tracking Prototyping System Software Product Line

This section presents three prototypes of an object tracking system that is a software application derived from the SPL architecture discussed in the previous section. The goal is to create three systems from the software product line architecture. All three prototypes will utilize the object tracking system to fulfill the specific requirements of the project. The scopes of these three prototypes are listed in Table 5.

TABLE 5. SCOPE OF THE PROTOTYPES

Low/No Visibility Navigation System (LVNS)	Aid emergency personnel when navigating through an unfamiliar building
The Location Tracking and Maze System (LTAMS)	Find the solution to a maze
Shop Navigator System (SNS)	Provide shoppers with the ability to easily locate products on store shelves

The LVNS is a tool that aids emergency personnel in navigating through unfamiliar buildings during low to no visibility operations. The tool first displays the layout of the simulated building, the waypoints to be used in the simulation, and a static compass for use in the simulation. Then, the LVNS detects the current location of the person, computes the distance to the closest waypoint, and displays the direction to the waypoint. Thus, a user can use this tool to direct an individual to specific waypoints within the simulated building.

LTAMS randomly chooses a maze that is drawn on the screen. The player moves the Cricket listener across a table and finds the solution to the maze. All of his/her actions will be checked and displayed on the screen. LTAMS is able to throw out bad data and avoid cheating. The player also has the ability to change the size of the playground on which he/she is actually moving the listener.

SNS provides a location aware system that guides a user through a simulated supermarket to complete his/her shopping list. The shopping list is assumed to have been entered from the user's home into the store's web site and stored remotely. Upon entering the store, the user will obtain a shopping cart with the device attached. In this prototype, a store plan is drawn on the screen. Cricket beacons will have specific, fixed locations, and the Cricket listener will be moved across the floor plan as shown in Figure 5 to simulate the user's movement while locating each item on the shopping list.

The architectures of these three prototypes created from the software product line architecture are shown in Figure 5.

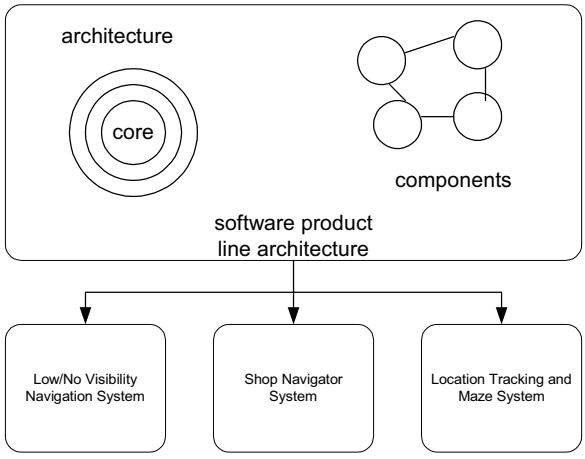

Figure 5. Design of the software architectures.

The experimental results of these prototypes all show a similar latency problem. If the listener is moved too quickly, the system may lose the object. Another common problem occurs when the sensors are blocked by obstacles, causing inaccurate positions to be transmitted.

V. Evaluation of Product Line Architecture Development Method

Matinlassi [9] presents a framework to evaluate software product line architecture development methods based on context, user, content, and validation. In this section we are applying the same criteria to the development method proposed by Linden [10]. The criterion of context is evaluated in terms of the specific goal of the method, the product line aspects that the method covers, the application domain the method is focused on, inputs to the method, and outputs produced by the method. The specific goal of this product line development method is to have an architecture centric and component-based development method. The method starts with the user requirements followed by the architecture design and ends with the implementation of components. The application domain that the method is focused on is real-time object tracking.

The user criterion is evaluated in terms of stakeholders addressed by the development method, motivation of using the development method, needed skills for the method, and guidance to the development. The stakeholders addressed by this method are mainly software engineers. The benefits of using this development method are software reuse for higher quality and shorter market time. The users may need to be familiar with UML notations to accomplish the tasks required by the method. Guidance to the development is currently absent.

The content criterion is evaluated in terms of design steps used in the method, artifacts created by the development, architectural viewpoints the method applies, language to define the models, diagrams, other artifacts the method produces, variability supported by the method, and tool supporting the development. The steps of the development include domain engineering followed by software engineering. The architecture mainly describes the functional viewpoints of the system. UML can be used as a language to describe the design of the system. The variations of the system are explicitly specified in use cases. UML tools are applicable to aid in development.

The validation criterion is evaluated in terms of the development method validation and how the development method validates the quality of the product. From the development of these three prototypes, we observed that not every component in the software product line architecture is used by every member of the product line. For example, user interface requirements differ so widely across LVNS, LTAMS, and SNS that most of the user interface components are only used once but are still maintained as product line assets. However, the components are designed to be flexible and configurable for software evolution. Nevertheless, these three prototypes share the components of data acquisition and three-dimensional trilateration. We also observed that the architecture has limitations in some applications. For example, we are conducting a research project where Cricket wireless sensors are used to automatically control the lighting by tracking an actor's position on the stage. The coordinates of the actor's position in x, y, and z need to be converted into the coordinates of pan and tilt so a DMX-512 controller can control the fixture lighting accordingly. Currently, the component interfacing with external systems such as DMX-512 controller was not taken into consideration in the architecture [15].

VI. Summary

Wireless sensors have been used in applications of industrial and building automation, object tracking, and environmental monitoring. In this paper, we presented a software product line architecture used for real-time object tracking for software reuse in various applications that used Cricket wireless sensors. Object tracking is needed in various applications, such as supply-chain and transportation. The presented software product line architecture enables the

architecture of object tracking to be embedded in other larger software architectures [11, 14].

ACKNOWLEDGMENT

We sincerely thank the students in the software engineering class at UALR for implementing the prototypes of the project. Our thanks also go to the group of wireless network at University of Arkansas at Little Rock (UALR) for sharing their wireless sensors, kits, and platforms with us. Without their help, this project would not be accomplished.

REFERENCES

[1] J. Bosch, Design & Use of Software Architectures, Addison-Wesley, 2000.

[2] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2002.

[3] C.-C. Chiang and J. E. Joseph, "Scalable templates for specification reuse," Proceedings of the Twentieth-First Annual International Computer Software & Applications Conference, IEEE Press, New York, NY, 1997, pp. 396-401.

[4] A. Davis, "Operational prototyping: a new development approach.," IEEE Software, 1992, pp. 70-78.

[5] L. Brownsword and P. Clements, A Case Study in Successful Product Line Development, CMU/SEI-96-TR-016, Carnegie Mellon University, 1996.

[6] SEI PLP, Framework for Product Line Practice, April 15, 2003, DOI=http://www.sei.cmu.edu/plp/

[7] C.-C. Chiang and R. Lee, "Developing tools for reverse engineering in a software product-line architecture," Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, IEEE Press, New York, NY, 2004, pp. 42-47.

[8] G. Shao and Y. T. Lee, "Applying software product line technology to simulation modeling of emergency response facility," Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, 2007. (In press)

[9] M. Matinlassi, Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA, and QADA, June 5, 2007, DOI= http://spic.kaist.ac.kr/~selab/html/Study/MS%20Study/2007/5.Comparison%20of%20Software%20Product%20Line%20Architecture%20Design%20Methods(COPA,FAST,FORM,%20KobrA%20and%20QADA).pdf

[10] V. D. Linden, "Engineering software architectures, processes and platforms for system families – ESAPS overview," Proceedings of the Second International Conference on Software Product Lines, Springer, Heidelberg, Germany, 2002, pp. 383-397.

[11] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996.

[12] Cricket, The Cricket Indoor Location System, Retrieved November 19, 2008 from *http://cricket.csail.mit.edu*

[13] Crossbow Technology, Retrieved November 19, 2008 from http://www.xbow.com/wireless_home.aspc

[14] L. Bass, P. Clements, and R. Kazman, R., Software Architecture in Practice, Addison Wesley, 1998.

[15] H. Park, M. B. Srivastava, and J. Burke, "Design and implementation of a wireless sensor network for intelligent light control," Proceedings of the Sixth International Conference on Information Processing in Sensor Network, ACM Press, New York, NY, 2007, pp. 370-379.