A Reusable Architecture with Product Line Technique Applied to Context Sensitive Service

Seojeong Lee[1], S.W. Hwang[1], G.H. Kim[1], G.S. Ryu[1] and Misook Choi[2]

[1] Division of IT Engineering, Korea Maritime University
1 Dongsam-dong, Youngdo-gu, Busan, Korea 606-791
sjlee@bada.hhu.ac.kr
[2] Department of Computer Engineering, Woosuk University
Wanjoo, Chonbuk, Korea 565-701
kms67_kr@hanmail.net

Abstract. In ubiquitous environments, the content adaptable services can be dynamically provided to adapt the frequent changes of contexts. These services have common things that the kinds of context factors are limited to ubiquitous environment, though the contexts are flexible. To reuse service architecture can be reasonable for effective adaptable service. In this paper, we propose a reusable architecture with product line techniques for content adaptable applications in ubiquitous environment. It can describe reusable points for service and optimization policies. Description of reusable points is to define variation points and their variants and to find out the dependencies between them. Optimization policies are to build the variant selection strategies. These can accomplish to define the decision model based on content adaptable service, and to help the reuse more effective.

1 Introduction

Over the years software development has been applied to various logical and physical environments. To support the environments, various methods or development processes have been tried. Recently, growing of the ubiquitous computing, the developers need to consider which features can make sense to aware context and to serve adaptable contents.

The content adaptability is one of important issues for this environment. This means that the adaptable content can be served on the fly, and to do that, the application has to the strategies to decide the appropriate service. These services have to be predefined and modeled. And if well defined and modeled, it can be reasonable to reuse for another application which has same objectives or same concerns.

By adopting an architecture-based approach, [1] provides a reusable mechanisms for specializing that infrastructure to the needs of specific systems. Several researches suggest frameworks that facilitate the development and deployment of web services aware, respectively [2] [3]. So far, the attention of content service via network is now being directed as how to resolve the context awareness not how to define the strategy and reuse. Adapt framework introduced workflow concept for design service so that

can be dynamic web service [4].

In parallel, the product line engineering concept has been initiative at Carnegie Mellon to practice a dependable low-risk high-payoff practice that combines the necessary business and technical approaches to achieve success [5]. Variation point and variant are the core concept of product line. In [6], the authors took their effort for describing the taxonomy of variability dependencies and the types of variability dependencies. This concept can be useful for establish the architecture model for content adaptable application.

In this research, we suggest a reusable architecture for content adaptable application. It is based on product line concepts to describe reusable service factors as variation points, variants and their dependency.

2 Related Works

2.1 ADAPT Framework for Adaptable and Composable Web Service

The emergence and diffusion of web services technology is creating a new business opportunity for providing value added, inter-organizational services by composing multiple basic services (BSs) into composite services (CSs). [4] has provided an overview of the ADAPT framework that we have developed for addressing this novel challenging scenario. On the one hand, ADAPT provides middleware support for developing higly available and dynamically adaptive BSs that can be used to build higher-level CSs. On the other hand, ADAPT includes the technology and software infrastructure necessary for defining, enacting, and monitoring inter-enterprise business processes that are implemented as CSs with guarantees of availability, scalability and adaptability not only to changing network conditions and user requirements but also to reconfigurations and repairs.

In Adapt, a composition only includes a reference identifying which service should be invoked as part of the execution of a specific task [7]. And, [8] discusses how the engine features autonomic self-tuning properties. A closed feedback loop controller has been introduced that monitors the current workload running on the distributed engine and adapts its configuration in order to optimize the system's performance based on several policies. Figure 1 shows the feedback loop of the Adapt composition engine.

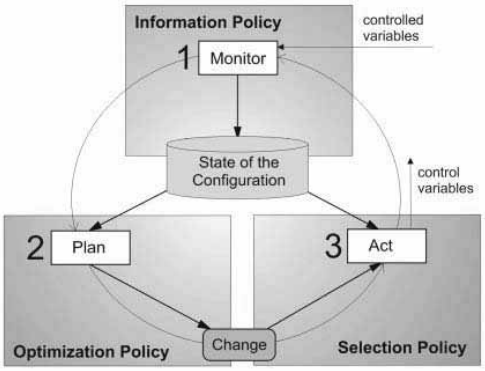

Figure 1. Feedback loop of the Adapt composition engine

2.2 Dependency Decision Model

Binding to variation points, there can be a relationship between the variation point and the selected variant. This relationship may imply certain dependencies (constraints), e.g., a system generally requires that specific variation points are bound to have a working, minimal system. There can be many different types of dependencies and pinpointing them requires a more formal way to describe variability [6].

The following nomenclature aims for describing variability in system-independent terms, i.e., independent from a particular system, method or organization:

- The set of all variation points: $VP = \{vp_a, vp_b, vp_c, \ldots\}$
- The set of variants for vpx: $\text{vpx} = \{\text{vx}_1, \text{vx}_2, \text{vx}_3, \ldots\}$
- The power set (the set of subsets) of all variants:
  $V = \{\{va_1, va_2, va_3, \ldots\}, \{vb_1, vb_2, vb_3, \ldots\}, \{vc_1, vc_2, vc_3, \ldots\}, \ldots\}$
- A relationship between vpx and vx_n, i.e., vpx binds vx_n: (vpx, vx_n)

The dependencies between variation points and variants can be expressed in the form of conditional expressions:

- if vp_x is bound then vp_y should be bound: **if** vp_x **then** vp_y
- if vp_x is bound then vp_y should bind v_{yn}: **if** vp_x **then** (vp_y, v_{yn})
- if vp_x binds v_{xn} then vp_y should be bound: **if** (vp_x, v_{xn}) **then** vp_y
- if vp_x binds v_{xn} then vp_y should bind vy_m: **if** (vp_x, v_{xn}) **then** (vp_y, v_{ym})
- if vp_x binds v_{xn} then vp_y should not bind v_{ym} : **if** (vp_x, v_{xn}) **then not** (vp_y, v_{ym}).

These may explain why dependencies often exhibit different kinds of behavior while appearing similar in the first instance. The constraints imposed by each type have been explored in various areas of research, e.g., the configuration management community, however implicit specification may bother adopting to realize.

3 Reusable Content Adaptable Service Architecture

In the age of ubiquitous computing, users can take various devices and then their needs are complicated. Devices can vary from a workstation, to a mobile phone, a PDA or any other limited terminal. Application developers have to take care of adaptable service by context changes.

Several outputs are impressive excepting non-reusable. Applications on web are most often used in a request/response interactive way. The message is service-oriented. So, web application architecture should identify who or which component is responsible for:

- providing service
- request service
- description where service providers publish their service descriptions
- description where service requestors find services and how to binding

Then, the applications contain some functions for their services [9]: Service Registration, Service Discovery, Service Interface, Definition Repository, Publication, Service Binding, Authentication, Authorization, Functions, Load balancing support, Fault-tolerance support.

The services which directly address the requirements are needed to design. So, the system designer has take into account relevant concerns [10][11] which are integrated functional considerations above mentioned.

In this point, we can find out that the common factors to make service and the differences are on their specific values. If a service model could be depart from real data, building strategies and binding services, it can be reused.

This paper suggests a reusable architecture for content adaptable service, introducing product line technique: variation points and variants. Figure 2 shows an example. *Presentation layer* consists of *Changing context* and *Changing preference*. *Service layer* consists of *Derivation* and *Optimization*. *Data layer* is composed of *Dependency*, *Optimization set* and *Physical data*. Next six sections from 3.1 to 3.6 describe the *Service layer* for this architecture.

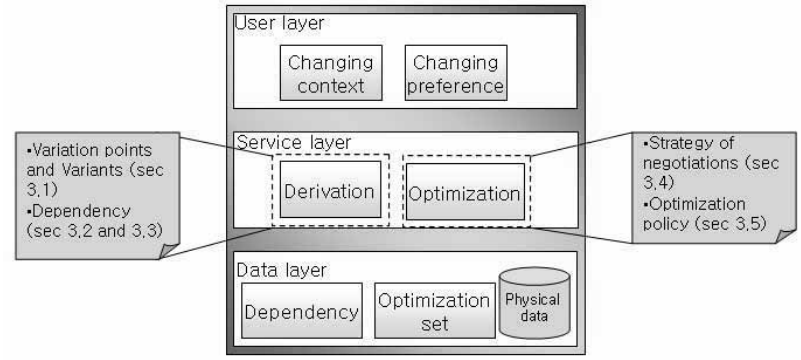

Figure 2. Suggested Content Adaptable Service Architecture to Reuse

3.1 Derive Variation Points and Variants

The context in ubiquitous computing consists of the profile of network, device, service, user and so on [12]. They are presenting current or changeable context and then used for service decision which is context aware and content adaptable.

This paper considers network status, device status, service commitment, avail data status and user bias as functional features, though there are potentially additional functional features. Network status contains the information of the physical transmission ability. Device status represents which device type, how much memory is remained, how fast the processor, and so on. Service commitment describes that any user can take any service in some location. Avail data status describes the hardware requirement and data properties. And, users can specify their own preference on user bias.

For example, network status can be considered of protocol, bandwidth and latency, like as Network status = (protocol, bandwidth, latency). Using set notation, these can be represented as:

- N ={ni | 1<=i<=j}, n represents the variation point of network status
- D ={di | 1<=i<=k}, m represents the variation point of device status
- S ={si | 1<=i<=l}, s represents the variation point of service commitment
- A ={ai | 1<=i<=m}, a represents the variation point of avail data status
- U ={ui | 1<=i<=n}, u represents the variation point of user bias

The graphic notation is often used to set up or understand the model easier. Figure 3 represents the notations for dependency. Notations for feature, variation point and variant are from COVAMOF [13], and compose_of_vp, depends_on_vp and depends_on_variant are introduced in this paper.

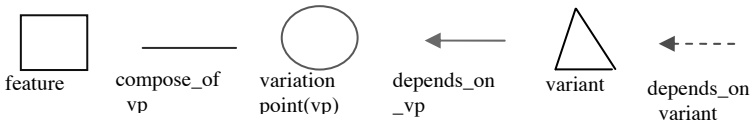

Figure 3. Notations for Representing Dependency

3.2 Derive the Dependency between Variation Points

The dependency means a relation that a variation point effects to decide another variation points. Figure 4 is an example of define the dependency between variation points. Here, "n_1 *depends_on_vp*(←) s_1" means that decision of n_1 is dependent to decision of s_1 .

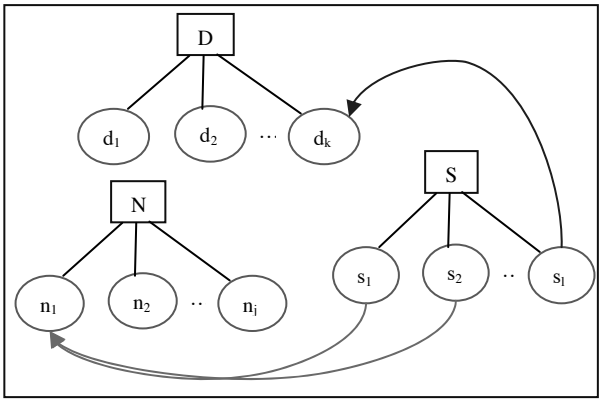

Figure 4. Dependency between Variation Points

There are some guidelines to derive as following:

- A variation point can *depends on* (◄—) one or more variation point(s).
  - n_1 ◄— s_1 and n_1 ◄— s_2 is possible.
- One or more variation point(s) can *depends on* (◄—) one variation point.
  - n_1 ◄— s_1 and n_2 ◄— s_1 is possible.
- Some variation points may not *depends on* (◄—) any variation point.
  - No dependency
- It is not allowed that a variation point *depends on* (◄—) itself.
  - n_1 ◄— n_1 is not possible.

3.3 Derive the Dependency between Variants

If a specific variant may affect to decide another variant, it can limit other variant's value. It is called the dependency between variants [6]. The dependency between variants is a relation that a variation point effects to decide another variation points. Figure 5 is an example of define the dependency between variants. Here, "$n_{1.1}$ *depends_on_variant* (◄- -) $s_{1.1}$" means that decision of $n_{1.1}$ is dependent to decision of $s_{1.1}$.

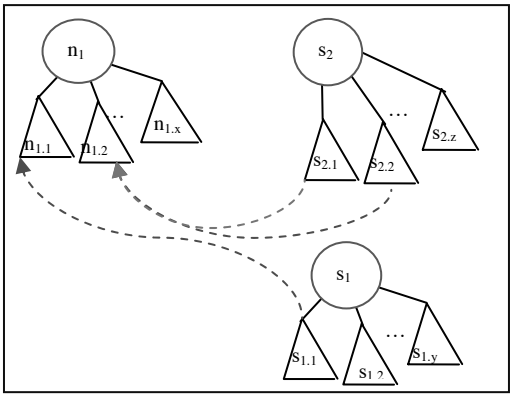

Figure 5. The Dependency between Variants

There are some guidelines to derive as following.

- All variation points are composed of one more variants
  - If no variant, it supports not variability.
- All variants have their own *data_type* and *value_properties*.
  - Described in section 3.2
- If dependency between 2 variation points, at least one variant in a variation point should *depends_on_variant* (⇠) one variant in the other variation point.
  - For example, if $n_1 \leftarrow s_1$ then at least, $n_{1.x} \dashleftarrow s_{1.y}$ should be.
- A variant can *depends_on_variant* (⇠) one more variation point(s).
  - $n_{1.2} \dashleftarrow s_{2.1}$ and $n_{1.2} \dashleftarrow s_{2.2}$ is possible.
- Two more variant can *depends_on_variant* (⇠) one variant.
  - $n_{1.1} \dashleftarrow s_{1.1}$ and $n_{2.1} \dashleftarrow s_{1.1}$ is possible.
- Some variants may not *depends_on_variant* (⇠) any variant.
  - No dependency
- It is not allowed that a variant *depends on_variant* (⇠) itself.
  - $n_{1.1} \dashleftarrow n_{1.1}$ is not possible.

3.4 Optimize the Strategy of Negotiation

To service the content adaptable, the decision to select content has to be performed, called negotiation [12]. The strategy of negotiation is dependent on domain, service and application. Followings are the basic guidelines for successful definition and then these can be potentially additional.

- Every *depends on* (←) should be defined by one more strategies
- Every *depends on_variant* (⇠) should be defined by one more strategies
- The decision value of strategy should be *one_of* or *in_the_range_of* variants value

3.5 Optimize the policies

This step is to optimize the policies instead of dependency negotiation in section 3.4. Figure 6 shows the optimization of policies, which has the basic ideas.

- For one variation point(vp), all variants can be listed sequentially by their priorities
- The range of variants comes from their dependency.
- All variation points are logically orthogonal.
- Every service may not satisfy all needs of user or context
- Adaptable service can be found out on the polygon, which is made of drawing lines among variants
- Adaptable service can be two or more.
- The variants of one adaptable service is to be an optimization set

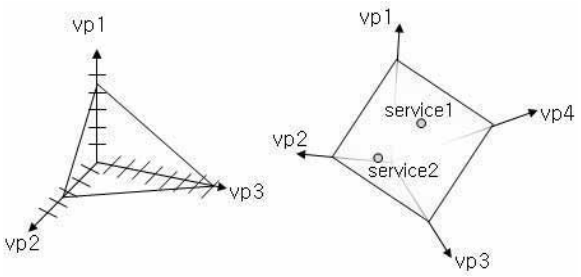

Figure 6. The Optimization Policies

4 Implementation

This architecture is implemented by Java. For import external request, we use XML and SOAP messaging technique. Using JDBC, retrieve a variation point or variants from their own templates. And then, modules for negotiation of strategy are implemented by Java code.

Since the features for content service have been departed from negotiation module, feature template [15] and definition process using product line technique can be reused. Establishing the details for utilizing templates and building optimization set, can help the complete implementation.

5 Conclusion

Context awareness is an important issue in ubiquitous computing. We proposed a service decision modeling method for content adaptable service. It is based on product line concepts such as variation point, variant and dependency. The architectural process consists of deriving variation points and variants, deriving the

dependency between variation points, deriving the dependency between variants, and optimizing the strategy of negotiation and policies.

We also introduced a specific modeling method for content adaptable service and it can bring about optimizing decision strategy. The concept of product line helps it systematically. Otherwise, it can be reused when working upon different requirements. And the more complicated the dependency, the more effective the templates in this paper as a tool of expression.

Acknowledgements

This work was supported by Korea Research Foundation Grant funded by Korea Government (MOEHRD, Basic Research Promotion Fund) (KRF-2005-003-D00327)

References

1. Alberto Bartoli, Ricardo Jiménez-Peris, Bettina Kemme, Cesare Pautasso, Simon Patarin, Stuart Wheater, Simon Woodman, "The Adapt Framework for Adaptable and Composable Web Services," *IEEE Distributed Systems Online*, IEEE, September 2005
2. Michalis Anastasopoulos, "Software Product Lines for Pervasive Computing," *IESE-Report No. 044.04/E version 1.0*, IESE, April 2004.
3. Robert Grimm, Tom Anderson, Brian Bershad, and David Wetherall, "A System Architecture for Pervasive Computing," *Proceedings of the 9th ACM SIGOPS European Workshop*, pages 177-182, September 2000.
4. Tayeb Lemlouma and Nabil Layaïda, "Context-Aware Adaptation for Mobile Devices," *Proceedings of International Conference on Mobile Data Management (MDM),* IEEE, January 2004.
5. Markus Keidl and Alfons Kemper, "Towards Context-Aware Adaptable Web Services," *Proceedings of World Wide Web (WWW'04),* ACM, pages 55-65, 2004.
6. Paul Clements and Linda Northrop, Software Product Lines, Practices and Patterns, Addison-Wesley, 2002.
7. C. Pautasso, G. Alonso, "Flexible Binding for Reusable Composition of Web Services", *Workshop on Software Composition (SC 2005)*, LNCS, April 2005.
8. C. Pautasso, T. Heinis, G. Alonso, "Autonomic Execution of Service Compositions", *Proceedings of the 3rd International Conference on Web Services (ICWS 2005)*, IEEE, July 2005.
9. Dertouzos, Michael L. "The Unfinished Revolution: How to Make Technology Work for Us-Instead of the Other Way Around," HarperCollins Publishers, October 2002.
10. G. Di Caprio, C. Moiso, "Parlay Web Services Architecture Comparison," *EXP online*, Vol. 3, Num. 4, December 2003.
11. Robert Grimm et. al, "System Support for Pervasive Applications," *ACM transactions on*

Computer Systems, Vol.22, No.4, pages 421-486, November 2004.

12. David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl and Peter Steenkiste, "Rainbow: Architecture-based Self-Adaptation with Reusable Infrastructure," *Computer*, IEEE, pages 46-54, April 2004.
13. Girma Berhe, Lionel Brunie and Jean-Marc Pierson, "Modeling Service-Based Multimedia Content Adaptation in Pervasive Computing," *Proceedings of ACM International Conference on Computing Frontiers (CF'04)*, ACM, April 2004.
14. Marco Sinnema, Sybren Deelstra, Jos Nijhuis and Jan Bosch, "COVAMOF: A Framework for Modeling Variability in Software Product Families," *The Third Software Product Line Conference (SPLC)*, LNCS 3154, pages 197-213, 2004.
15. Seojeong Lee and Soo Dong Kim, "A Rendezvous of Content Adaptable Service and Product Line Modeling," *Proceedings of Profes 2005*, LNCS 3547, pages 69-83, 2005.