Developing a Repository for Component-Based Energy-Efficient Software Development

Doohwan Kim   Jang-Eui Hong
Dept. of Computer Science
Chungbuk National University
Cheongju, Rep. of Korea
email: dhkim@selab.cbnu.ac.kr, jehong@chungbuk.ac.kr

***Abstract*— Software components are one reusable asset which can contain other kinds of software assets like requirement specifications, design patterns, source codes, documents, and so on. It can be used for designing software architecture or implementing a software system as an element like a building block. Therefore, software can be developed easily and quickly by assembling those building blocks. Focused on this nature of component-based development, energy-efficient software development can also be achieved with reusable software components. In particular, low-energy software has become a critical component for embedded and mobile software systems. Therefore, we have to consider energy efficiency to develop embedded software when developing the software based on reusable components. This paper, firstly, proposes how to represent the energy characteristics of the components and how to select a component for energy-efficient software development. We developed a component repository, ECoReS to support the selection of low-energy software components.**

Keywords-reusable software assets; component repository; energy-efficient software; component selection.

I. Introduction

Software reuse has become one of the general processes to develop software systems because reuse can provide huge benefits of error prevention, cost and time reduction, and even quality improvement [1]. The representative paradigm based on the reuse approach is known as CBSD (Component-based Software Development). CBSD can effectively support embedded software development because this kind of software frequently includes the same functions as other embedded software, which are in the product family. Therefore, the software can be developed faster and more reliable than developing it from scratch [2].

To elevate the benefits of software reuse, a repository that manages reusable assets on an organizational level is required. The purpose of a software component repository is to support the reuse of the components that have been acquired at the organization level. The repository also has to provide the functions of component registration, component retrieval, and component selection to software engineers for systematic reuse [3]. Therefore, many component repository systems have been developed that focus on managing and retrieving components to satisfy the functional requirements of developing software. However, embedded software development has to consider not only the functional requirements, but also various non-functional requirements because of limited resources and operational environments [4]. Therefore, these limitations must be considered as one of the characteristics, also known as the quality factors, of the software through the entire development process [5][6][7]. Low-energy consumption, as one of the characteristics of embedded software, has become a very important quality factor in portable or mobile systems like smart phones, MP3 players, and tablet PCs, because those systems are powered from limited energy sources such as a battery. However, there are just a few studies on the development of component-based low-energy software.

The major profits of component-based energy analysis can be considered from two sides: the first one is the reduction of feedback costs by early-phase estimation of energy consumption, and the second is the provision of high reliability of the estimation result. The higher abstraction level tends to lead to less accurate or coarse-grained analysis results [8]. However, reusable components have their own developed code which is managed in a component repository. Because the components can be used as the computational units of software architectural components, they make possible architecture-level analysis for requirements verification, which is a high abstraction level of software. Moreover, the component code will improve the accuracy of the analysis result. Therefore, if there is any method to support the energy analysis based on components in embedded software development, we can take the two advantages which conflict with each other, i.e., early phase estimation and its high reliable result. For these reasons, a component repository supporting low-energy software can be considered as one of the important infrastructures in the CBSD paradigm. Therefore, we developed a component repository, named Energy-based Component Retrieval System (ECoReS), which manages software components with their energy characteristics. The ECoReS can support fast and efficient embedded software development from the perspective of low-energy consumption, by providing the component selection based on energy characteristics.

The rest of this paper is organized as follows: the analysis of related work is explained in Section 2. The strategies to develop an energy-considered component repository will be discussed in Section 3. Section 4 describes the implementation of our proposed component repository, ECoReS. Section 5 describes conclusions and future research.

II. Related work

Each components repository can have different features according to policies of organization, application domain, and engineering environments [9]. The existing repositories have been providing various retrieval methods to find appropriate components. Therefore, each repository has its own distinct features, and specific structure of the repository. There are many repositories that are successfully supporting component reuse in the CBSD process [10][11][12]. Among them, we investigated recently proposed component repositories.

Z. Hai-mei et al. [10] emphasized the importance of component compositions as the issue to be managed by the component library. The authors designed a component library information model to improve reuse rate of managed components. Schema of that model included basic properties, classification information, and composition information. The key information of this research was the composition to enhance the reuse rate of stored and managed components in the library. However, the information model of the component library did not consider any information related to non-functional properties.

The research of C. Li et al. [11] focused on semantic-based component retrieval. They proposed an ontology-based component repository. Therefore, a software engineer who wants to develop component software can easily find software component using the ontology. However, the ontology did not include the terms of non-functional properties of software.

X. Shoukun et al. [12] developed a component library based on a component specification language named UCDL (Universal Component Description Language). By using the UCDL, the library manages the component information with the categories of basic information, classification information, interface specification, and feedback mark. However, this component library does not support the information of non-functional properties of components either.

The above studies implemented their repositories with different structures, and they provided distinct functions to maximize the reuse of components. However, these repositories can cause mistakes in the selection of suitable software components when the software engineer has to consider one or more non-functional requirements. Because the missing non-functional properties can lead to the re-development of large parts of the software [5], the engineer must consider the properties at the first step of component selection. Therefore, absence of the information of those properties can lead to inappropriate selection of software components. This incorrect selection can involve large re-developing costs, or even critical failure of the software.

Even though these component repositories provided convenient functions and they are well designed to support the reuse of components, they should be able to manage and provide information about the non-functional properties of the software components. Because the energy efficiency of software has become one of the most important non-functional properties of embedded systems, the perspective of low-energy must be considered in component-based software development. However, the previous studies and their repositories did not provide the distinguishing functions to support energy characteristics of the managed components. Our component repository provides the distinguishing functions for managing the energy characteristics of software components and supporting energy-efficient component selection.

III. Strategies for Low-Energy Software Development

As mentioned above, the component repository has to support low-energy software development by providing the reusable components. To support systematic reuse, we consider and define some strategies for describing energy characteristics of components, and for selecting a suitable component from our component repository.

A. Desiarable Useage Overview

Like general component repositories, the ECoReS also provides general features for component management and reuse. Additionally, certain distinguishing features are also provided by the ECoReS to support energy-efficient component selection. These features are reflected in the functions for component registration and component searching. When an engineer of an organization enrolls a new component in the ECoReS, it requests additional information that is related to the energy characteristics of the component. This information will be referred to by the software engineer who has to develop energy-efficient software. The availability of searching for the component characteristics in the ECoReS is also a different feature compared to other existing repositories. Because the functional requirement must be satisfied first when selecting a reusable component, the ECoReS also provides this functional requirements-based search. After that search, the software engineer can get a set of components which have the same functionality. We refer to these as “candidate components”. Given the main purpose of the ECoReS, energy-considered component selection will be done at the next step. The ECoReS provides the energy-comparing feature for actual selection from candidate components. Figure 1 shows an overview of the ECoReS including these features.

There are four specialized features in our repository, which are represented by the colored boxes in Figure 1. We also designed a feature, the “Function-based Search”, to provide easier search than just a general one. However, this feature is not colored because it is not the main concern of this research. As shown in the figure, the features of “add Component”, “Delete Component” and “Edit Component” are provided to manage components. Among them, “Add Component” has a specialized function that specifies the energy information. After using the “Function-based Search,” other specialized features will be supported. The software engineer will select the candidate components that have the same functionality to compare their energy efficiency. If the sequential order of searching based on function requirements, selecting candidates, and comparing candidate components were changed, the selection works will be meaningless. The feature, “Select the most suitable component” means the selection of a component that satisfies both functional requirements and energy-efficiency.

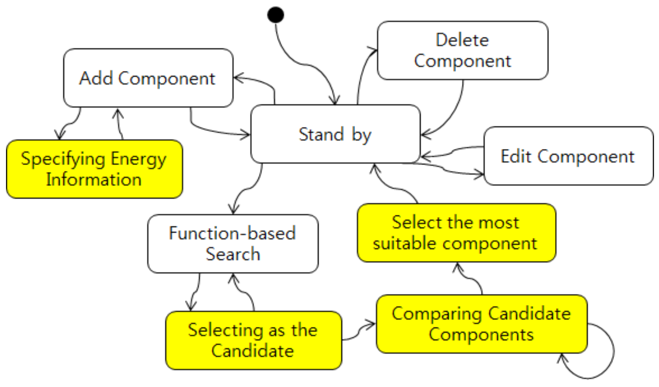

Delete Component
Add Component
Stand by
Edit Component
Specifying Energy Information
Function-based Search
Select the most suitable component
Selecting as the Candidate
Comparing Candidate Components

Figure 1. Usage overview for the ECoReS

B. *Manage Energy Characateristics by Interface*

A component can have several interfaces as the initiating entries of component behaviors. Those interfaces are the only way to request services of the components. Thus, those interfaces are identical to the basic units of the behavioral set that can be provided from components. This is a very critical notion that must be considered to manage the energy characteristics of reusable software components, because software never consumes energy by itself, but it consumes energy by controlling hardware devices when it is executed [13]. Therefore, the energy characteristics of components have to be defined by interface, and then have to be used to select components within the component repository.

C. *Consider the Effects of Interface Parameters*

Energy consumption of an interface can be changed by circumstances such as system conditions and given conditions of parameters. System conditions affect energy consumption on a small scale, while parameters can make a huge difference. For example, a specific parameter can be used as a flag variable which decides the branch of an inside interface. In other cases, some interfaces show exactly the same behaviors repeatedly when a parameter is given as a data stream or a similar one for iterative processing.

Because of these effects from the interference parameters, energy consumption of each interface can be variable. This is the reason why we have to define the energy characteristics of the components with the unit of interface. Moreover, the energy characteristics of components sometimes cannot be expressed as simple scalar values only, but certain ones have to be represented with a regression model. To define the energy model, the following recommendations are delineated.

- Thoroughly investigate the parameters of interfaces to determine if they can affect energy consumption, and how they may affect the energy consumption. The types of those effects are not the same for all possible cases. Therefore, this kind of investigation has to be done first to get an energy model.
- Collect sufficient data for the possible specific conditions of the interface parameters (for example, input data size, data value, etc.) to define an energy model. As mentioned above, the system condition is also a reason for the variance. Only a huge amount of sampling data can include this kind of variation. Therefore, we can define a proper energy model from a regression analysis using the large data set. This kind of model is already defined and used in the research of T. K. Tan et al. [8].

D. *Retrieve Components based on Facets*

It is difficult to find a suitable component from the repository that has many similar components functionally. In particular, if software developers would like to consider non-functional requirements as well as functional ones in component retrievals, finding a specific component that satisfies both requirements might be difficult.

Therefore, a component repository has to fulfill the needs of component retrieval with featured methods. In order to realize the methods, we provide a multi-dimensional facet-based retrieval technique to the ECoReS repository. Our repository provides two facets; the first is the domain facet which is the target domain of the software being developed, and the other is the functional facet which is responsible to the functionality to develop the software. The facets act like filters to show the list of components that could be selected. Although facet-based retrieval is not the main concern of this research, it is designed and implemented to help find a proper component by making a set of candidate components which have the same functionality before actual selection, focused on energy efficiency. Software engineers can limit the boundary of component retrieval by using these facets. Also, how many facets will be used to find a component can be determined with the trade-off analysis for an exact search and plentiful candidates.

E. *Compare Energy Characteristics*

With the facet-based retrieval, software engineers can find the components that are suitable for the functional requirements. However, there is still the remaining problem of selecting a proper component that is suitable for energy efficiency too. Because the result of facet-based retrieval can provide just a list of candidate components from the repository, software engineers have to compare the energy characteristics of the candidates.

Candidate components mean the components that are exchangeable with each other, according to Definition 1.

[Definition 1] Candidate Components: let CS be a set of components, and CC be a set of paired components. Then C_x and C_y are candidate components when satisfying;

$$CS = \{C_1, C_2, C_3, \ldots, C_n\}, \quad (1)$$

$$CC = \{C_x \in CS, C_y \in CS \mid Pre(C_x) = Pre(C_y) \wedge Post(C_x) = Post(C_y)\}, \quad (2)$$

Where, the Pre(Cx) means the pre-conditions (the input of the component) of Cx and the Post(Cx) means the post-conditions (the output of the component) of Cx. Also, $1 \leq x \leq n$, $1 \leq y \leq n$, and $x \neq y$.

The identification of candidate components means finding a set of components that satisfy the required functionalities. After that, the repository has to recommend a lower energy-consuming component among those candidate components, because the major purpose of our repository is selecting the most suitable component, satisfying not only functionality but also energy efficiency.

We considered this problem with the comparison of energy characteristics based on the interfaces, their energy models, and their parameters. Figure 2 shows the energy consuming patterns between two components when they are compared.

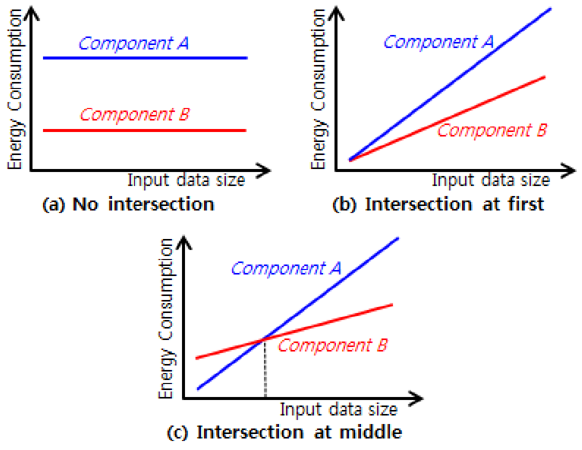


Figure 2. Energy consumption patterns between two components

Let us consider two components "Component A” and “Component B” that have the same function. However, they have different energy models for that function. For example, the energy model of the “Component A” is 2x+7, while the model of the “Component B” is 3x+2. As shown in the figure, the energy graphs of two components can be classified into three types like (a), (b) and (c), where all of energy models are linear regression models. In the case of (a) or (b), determining the more energy-efficient component between them is very easy. However, there is no component that is absolutely energy-efficient on the graph (c) in Figure 2.

In this case of the graph (c), “Component A” is more efficient in energy consumption until the input data size of A is less than the size of the crossing point. However, “Component B” is more efficient after exceeding the crossing point. The comparison of energy characteristics between candidate components from the perspectives of the energy models and input data size makes it possible to select a low-energy component, and also possible to develop component-based energy-efficient software.

IV. DESIGN AND IMPLEMENTION OF ECORES

Reusable software components must be managed and maintained on an organizational level by using a component repository or library [3]. The component repository provides several traditional functions such as the registration of components along with retrieval and selection of components to support the CBSD paradigm. The functions are basic and intrinsic for general component repositories.

However, our component repository, called ECoReS, provides not only the basic functions of general component repositories, but also the functions related to the energy efficiency of components. Those functions have to be considered and designed in a well-organized UI structure and seamless usage flow.

In this section, we explain how we considered the strategies for low-energy software development, what development environments were used to implement our component repository, and which functions are provided in our repository.

A. *Information for Component Specification*

Specifying a component to support easy reuse requires a lot of useful information for the components. However, there is a set of commonly required information such as the name, the usage, the list of interfaces, and the platform of a component for the specification [9][10][11][12]. This set of information is too general to examine further in component specification. Thus we only focus on the information that is required to describe the energy characteristics of components.

In the previous section, we discussed the energy models that describe the energy characteristics of component interfaces and interface parameters. However, some more information is required to describe the energy characteristics of components. This information is related to the platform in which the component is deployed. As mentioned above, software consumes energy by controlling the actions of hardware devices. Thus, the platform information must be covered in the component specification. These are the important platform specifications related to the energy characteristics:

1) *Hardware resources*: Hardware resources are actual energy consuming objects. Therefore, the information of the target (expected) platform of the component should be itemized in the specification. Among many of hardware resources, CPU clock speed and memory size are key information for energy characteristics [14][15].

2) *Operating System* (OS): Almost no application software can be activated without an operating system. The main purpose of OS is to control hardware resources (e.g., CPU, memory, etc.) and to provide system services to the application software like a middle layer broker. Therefore, the actual running environments of application software are controlled by the OS based on its scheduling policy, memory management policy, IO control, and so on. These policies can affect the energy consumption of software [15] and can be differently applied to its types (Android, IOS or something else) and versions.

3) *Compiler*: There are a number of available compilers. Even if the external behaviors of compiled codes (i.e., executable binary) for different source codes are the same, the compilers may have different optimization policies. An optimization policy can be differently applied during the code compilation process by setting different options even though the same compiler was used. Because the different optimization policy can generate different internal behaviors, the energy consumption of software is also different based on the different policies [16]. Therefore, we add the information of the compiler to specify the energy characteristics.

In addition to the energy models of component interfaces, the above information should be managed together. However, we can consider different tactics to manage that information because each part of the information represents different parts of a system. For example, the information of the platform for software is not changed during the lifetime of the software. Therefore, the influence of the platform on energy consumption will be decided at compile time. On the other hand, the information related to the real behaviors of the software can be decided at run time, i.e., which interface is called and how parameters are passed. The information can always change during real operation of the software depending on the requested user service.

We classified the information affecting the energy consumption into two types of factors: Indirect factors that are not changed during software operation, and direct factors that can be changed during the software operation. In the specification of component information, it is sufficient to describe the indirect factors of the component only once, since the factors have an equal effect every time on any interface and on any parameter. Unlike indirect factors, the direct factors (such as interfaces and their parameters) must be described multiple times in the specification, because they are differently affect to energy consumption based on which interface is called and how parameter is configured. TABLE I. shows these factors that affect energy consumption.

TABLE I. FACTORS AFFECTING ENERGY CONSUMPTION

Factors	Factor Types	Effecting Range
Hardware	Indirect factors	Whole component
OS		
Compiler		
Interface	Direct factors	Each interface
Parameter		

B. *Development Environments*

The ECoReS is developed to manage and retrieve software components with consideration of energy consumption. Ultimately, the goal of the ECoReS is to support component-based and energy-efficient software development. In the design of our repository, we separate it into DB-side and client-side because it can define N:M relationships.

Only one component repository is desirable in an organization to support various software projects because the centralized repository is easy to maintain and also easy to provide consistency for stored components, while multiple repositories are also valuable in distributed and collaborative development environments to elevate the flexibility and the variability of component-based development. The decision for the operational configuration of a component repository is dependent on the organization policy.

The client is developed by using JAVA with eclipse Rich Client Platform (RCP), Eclipse Modeling Framework (EMF), Java Data Base Connectivity (JDBC), as shown in TABLE II. Therefore, our repository system is possible to operate and use on any kinds of platforms.

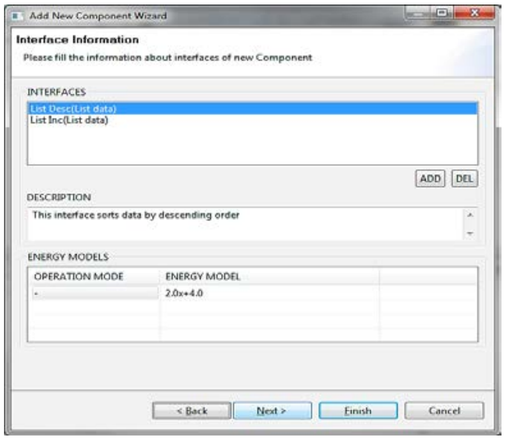
TABLE II. DEVELOPING ENVIRONMENTS OF ECORES

OS	MS Windows 7, 32bit
Language	JAVA(JDK 1.5)
Developing Tool	Eclipse 3.5(Galileo)
Platform	Eclipse Rich Client Platform
Plug-in	JDBC, EMF, etc.
DB	MySQL Server 5.5

C. *Implementation of the Strategies*

The common functional features of our repository are similar to other conventional component repository systems. However, the ECoReS has distinguishing features to support the strategies for energy-based component management, which are explained in Section III. This section is responsible for the implementation of those strategies.

The registration of components is a common and general function of component repositories except that the information of energy characteristics is also required. The component registration of the ECoReS provides a different widget to store the information about energy characteristics as shown in Figure 3.



Interface Information

Please fill the information about interfaces of new Component

INTERFACES

List Desc(List data)
List Inc(List data)

DESCRIPTION

This interface sorts data by descending order

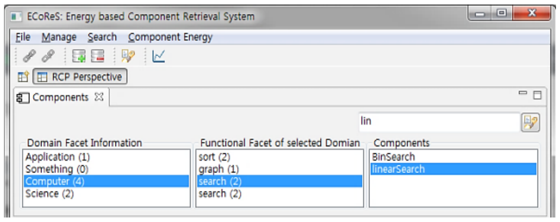
ENERGY MODELS

OPERATION MODE	ENERGY MODEL
-	2.0x+4.0

Figure 3. A Screen for energy characteristics of a component

The UI widget has the fields of “INTERFACES,” “DESCRIPTION,” and “ENERGY MODELS,” which should be filled in the “Add New Component Wizard” function. Because the indirect factors can be also identified as the commonly required information, we focused more on the energy model for each interface. Figure 3 shows an example of the "interface information" step of the component registration.

The facet-based retrieval is implemented with the composition of lists. Although the number of facets can be determined according to the domain hierarchy of the organizational business area, we define three facets such as domains, functions, and components level in our repository, as shown in Figure 4. This facet-based retrieval approach provides a quick and easy search to find a proper component in a functional manner and also helps software engineers think in top-down and systematic ways. Moreover, this approach can use the ontological concept to organize the facet structure.



lin

Domain Facet Information	Functional Facet of selected Domian	Components
Application (1)	sort (2)	BinSearch
Something (0)	graph (1)	linearSearch
Computer (4)	search (2)	
Science (2)	search (2)	

Figure 4. A screen for facet-based component retrieval

After searching for candidate components that satisfy the functional requirements of the target software, then the software engineer can select a proper component based on energy efficiency. To compare energy efficiency, software engineers can simply set the check-box to compare the energy consumption with other components in the "properties" tab of a candidate component, as shown in Figure 5.

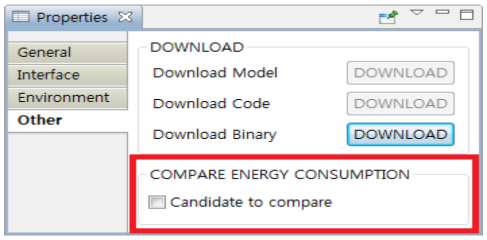


Figure 5. Selecting it as a candiate component

Comparing energy characteristics is the core feature of the ECoReS for selecting energy-efficient components. The comparison result of the energy characteristics between candidate components will be shown to help select the most suitable component for the software engineers. This result is shown in graph form to distinguish their energy efficiencies.

Selecting the specific interfaces of the components can be done when the comparison is activated. For example, there are two reusable components that are responsible for providing search algorithms. Even though the functions of two components are the same, their internal behaviors can differ. Therefore, the ECoReS will compare their energy efficiency based on component interfaces. Figure 6 shows the energy consumption graph of two components, "BinSearch" and "LinearSearch," which implemented a binary search algorithm and linear search algorithm, respectively.

In Figure 6, the "binSearch" consumes more energy than the "LinearSearch" at the first starting point. However, if the input parameter is bigger than 500 bytes, the "binSearch" is rather more energy-efficient than the "LinearSearch".

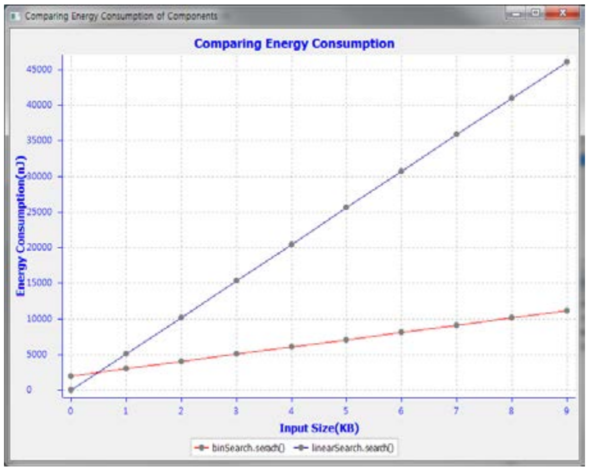


Figure 6. Energy consumption graph for two components

Due to this situation, software engineers who want to find an energy-efficient component have to consider the expected input data that was intended for processing by the component. In some cases in the above example, if we select to reuse the "binSearch" component when the input data size is always under 500 bytes, the selection will involve a worse result when the software is operated. The component always consumes more energy than the other component.

Therefore, we have to carefully predict and analyze the characteristics of the target system, to maximize the correctness of low-energy component selection.

V. CONCLUSION

CBSD has been broadly accepted as a reasonable and systematic paradigm to develop embedded software systems because embedded systems tend to be developed based on the product family approach [17]. The approach of reuse-based software development gives excellent benefits of time and cost reduction, and accuracy and reliability improvement as long as the approach applies well, and its infrastructure also runs well.

In this paper, we presented the design and implementation of a component repository, named ECoReS, which is a major infrastructure for the CBSD approach. Our repository specifically has focused on supporting component-based low-energy software development. Therefore, our proposed repository manages not only the general information of stored components, but also the energy characteristics of the components. It also provides functions like facets-based component retrieval, energy model management based on component interface, and energy consumption comparison.

We expect our component repository to be valuable in industrial and practical application development when a policy of energy-efficient software development is needed on an organizational level, or even on the subcontracting level.

In closing consideration, if any organization wants to combine or replace their existing component repository with the ECoReS, we expect that there will be three kinds of costs. For the first cost, the build or rebuild cost of repository will be needed. As mentioned, almost all component repositories have different structures. Even though the repository of the organization has a very similar structure with the ECoReS, minimum changes or re-builds for the structure are unavoidable. The second kind of cost deals with migration. After a total change of repository structure, migration must be followed. However, that migration can be omitted when the ECoReS is simply adapted to as-is system. The last cost is related to energy modeling. Because other repositories do not support the information about energy-efficiency, the energy modeling of the managed components should be done. Moreover, adaptation of the ECoReS without energy modeling is meaningless. We expect the cost related with energy modeling will be the largest of all.

For future studies, we are planning to upgrade the facet-based retrieval function with a powerful ontological scheme, and to establish the process and techniques for an architecture-based energy analysis framework, which cooperates with the ECoReS.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the NRF of Korea funded by the Ministry of Education (NRF-2014R1A1A4A01005566) and Next-Generation Information Computing Development Program through the NRF funded by the Ministry of Science, ICT & Future Planning (NRF-2014M3C4A7030503).

REFERENCES

[1] S. S. Yau, “Embedded Software in Real-time Pervasive Computing Environments,” in Proceedings of the 28th Annual International Computer Software and Applications Conference, pp. 406-407, 2004.

[2] X. Cai, M. R. Lyu, and K. Wong, “Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes,” in Proceedings of the 7th APSEC, pp. 372-379, 2000.

[3] J. Guo and Luqui, “A Survey of Software Reuse Repositories”, 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 92-100, 2000.

[4] M. Daneva, M. Kassab, M. L. Ponisio, R. J. Wieringa, and O. Ormandjieva, “Exploiting a Goal-Decomposition Technique to Prioritize Non-functional Requirements,” In Proc. Of WER 2007, 10th International Workshop on Requirements Engineering, pp. 190-196, 2007.

[5] N. Siegmund, M. Kuhlemann, M. Pukall, and S. Apel, “Optimizing Non-functional Properties of Software Product Lines by means of Refactorings,” in Proc. Fourth International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'10), Vol. 37 (27-29 January 2010), pp. 115-122, 2010.

[6] M. Marzolla, “Simulation-based Performance Modeling of UML Software Architecture,” Ph.D Thesis, Ca’Foscari University, Italy, 2004.

[7] D. Kim, J. Kim and J. Hong, “A Power Consumption Analysis Technique Using UML-Based Design Models in Embedded Software Development”, Lecture Notes in Computer Science Volume 6543, pp. 320-331, 2011.

[8] T. K. Tan, A. Raghunathan, G. Lakshminarayana, and N. K. Jha, “High-Level Energy Macromodeling of Embedded Software”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 21, No. 9, pp. 1937-1050, Sep. 2002.

[9] G. Jones and R. Prieto-Diaz, “Building and Managing Software Libraries,” in Proc. on COMSAC 1988, pp. 228-236, 1998.

[10] Z. Hai-mei and G. Min, “A Component Library Information Model Supporting Component Composition”, in Porc, 2012 IEEE International Conference on Mechatronics and Automation, pp. 475-479, 2012.

[11] C. Li, X. Liu, and J. Kennedy, “Semantics-Based Component Repository: Current State Of Art and a CalCuation Rating Factor-based Framework,” in Proc. 32nd Annual IEEE International Computer Softare and Applications(COMSAC 2008), pp. 751-756, 2008.

[12] X. Shoukun, C. Xiaomei, and M. Zhenghua, “A Study of Local Component Library Based on UCDL,” in Proc. ICCSE ’09, pp. 904-907, 2009.

[13] K. Naik and D. S. L. Wei, "Software Implementation Strategies for Power-Conscious Systems", Mobile Networks and Applications, Vol. 6, Issue 3, pp. 291-305, 2001.

[14] C. L. Su, C. Y. Tsui, and A. M. Despain, “Low Power architecture design and compilation techniques for highperformance processors,” in Proceeding on IEEE COMPCON’04, pp. 489-498, 1994.

[15] D. Sarta, D. Trifone, and G. Ascia, “A Data Dependent Approach to Instruction Level Power Estimation,” IEEE Alessandro Volta Memorial Workshop on Low Power Design, pp. 182-190, 1999.

[16] M. E. A. Ibrahim, M Rupp, and S. E.-D. Habib, “Compiler-based optimizations impact on embedded software power consumption,” in Proceedings of the Conference NEWCAS, pp. 1-4, 2009.

[17] K. C. Kang, J. Lee, and P. Donohoe, "Feature-Oriented Product Line Engineering", IEEE Software, vol.19, no. 4, pp. 58-65, July/August 2002.