APPLICABILITY OF DOMAIN-SPECIFIC APPLICATION FRAMEWORK FOR END-USER DEVELOPMENT

Takeshi Chusho
Department of Computer Science, School of Science and Technology, Meiji University
Kawasaki, 214-0033, Japan

ABSTRACT

It is preferable for business professionals to develop web applications which must be modified frequently based on their needs. A website for matching is a typical example because various matching websites for C2C (Consumer to Consumer) have recently been opened in relation to the "sharing economy". In our case studies on end-user development, web applications are developed using domain-specific application frameworks. Applicability of these domain-specific application frameworks has a tradeoff relation with programming granularity in which business logic is expressed. The template for the business logic definition based on three-tier architecture of user interfaces, business logic and databases improves the applicability.

KEYWORDS

Application Framework, Web Application, End-User Development, Applicability, Three-Tier Architecture

1. INTRODUCTION

Most web applications are developed by IT professionals and used by business professionals, domain experts and/or citizens. Such web applications have already become indispensable parts of our lives. On the other hand, a lot of routine work is still performed by manual operations although these manual operations for the routine work can be automated by IT. This is because web application development by IT professionals requires a comparative fund.

Primarily, it is preferable for business professionals themselves to support these web applications since web applications must be modified frequently based on users' needs. Therefore, technologies for end-user development have become important for the automation of the routine work.

Terms for end-user computing (EUC) and papers on EUC came out in the 1980s. Some papers described definitions and classifications of EUC (Cotterman et al. 1989), the management of EUC (Brancheau et al. 1993) and summary of the trends of end-user development without IT professionals' assistance (Sutcliffe et al. 2004). End-user software engineering research for end-user programmers and domain experts appeared also (Fischer et al. 2009; Ko et al. 2009). Reference (Scaffidi et al. 2005) estimated that the number of end-users increased in American workplaces and indicated that it was necessary to clarify what end-users are creating with their programming environments and how to improve those tools. Furthermore, reference (Burnett 2012) indicated that it was important to improve software quality in the end-user software engineering area.

Our research target is the technologies that business professionals with domain expertise can definitely define their business rules for their own jobs as requirement specifications which are transformed into executable software without additional description in programming languages. Generally, there are three approaches corresponding to the user interface (UI), business logic (BL) and database (DB) based on the three-tier architecture which has been popular with web applications. In our studies, we considered that end-users are familiar with the UI-driven approach and then developed domain-specific application frameworks based on components. The construction of the graphical user interface and the simple database system was supported by using application framework and visual modeling technologies (Chusho et al. 2011). As for the business logic, however, it is rather difficult to support it in the same way, because there are various kinds of business logic. The applicability of domain-specific application framework for end-user development is dependent on how the business logic is expressed. Then, the business logic should be expressed from the view of the service providers or the support systems instead of the view of the clients. Finally it is confirmed that the template based on the UI-driven approach is useful for requirement specifications of business logic.

This paper presents basic approaches for web application development in Section 2, applicability of domain-specific framework in Section 3 and the template for a tradeoff solution in Section 4.

2. BASIC APPROACHES

2.1 Domain-Specific Technologies

Our approach to end-user development is explained in the following layers:

- The business level {Business models}

  << The semantic gap: Domain-specific technologies >>
- The service level {Domain models}

  << The granularity gap: CBSE >>
- The software level {Components}

A business model at the business level is defined by those end-users who are business professionals and domain experts. Then, the domain model at the service level is constructed by the set of the required services as the software requirement specification. At the software level, the domain model is implemented using components. In this approach, the granularity gap between components and the domain model is bridged by CBSE (component-based software engineering) such as business objects (Sinha et al. 2013), patterns and application frameworks. On the other hand, the semantic gap between the domain model and the business model should be bridged by domain-specific technologies with domain knowledge (Sprinkle et al. 2009).

Approaches based on the three-tier architecture are classified into the three categories of UI-driven, model-driven and data-driven processes by first focusing on either the UI (user interface), the model (business logic) or DB. The UI-driven approach has recently emerged as web applications have been increasing sharply. In our UI-driven approach, visual forms were defined first and the framework was used. The business logic dependent on the application was included in form definitions. The other business logic was embedded into the framework. One of the main problems for end-user development is how to describe business logic.

In our studies on domain-specific technologies, the application domains such as the reuse of second-hand articles and lending books or equipment were selected because these domains are familiar to everybody. All of them required at least two DB tables for services. One was for member registration and the other was for object registration. The member implies a system user who may become a provider or a requester of an object. The object is an article, a book, or a piece of equipment. The basic operations are CRUD (create, read, update and delete). For example, a member can register an object, read registered objects, update them or deletes them. Although the columns of a record are dependent on the object, these differences are unified by the concept of “matching” between an object provider and an object requester.

Furthermore, business logic must be different for each application. There are many kinds of applications in the domain of matching services. It is difficult to develop an application framework that can be used for all kinds of matching services because such a framework requires a lot of customization and the merit of easy development of an application is lost. Therefore, it is necessary to focus our research target on a limited subdomain. For this purpose, we analyzed and classified matching services (Chusho et al. 2015). Websites for matching services are characterized by the following three factors:

- WHO : providers and requesters
- WHAT : things and services
- HOW : algorithms for matching decision

For the WHO factor, providers and requesters are limited to ordinary users in our research. Such business activities, such as online shopping and hotel reservations, are not our research target, because the requirements for web applications are too complex. Reuse promotion services supported by local governments, however, are our target because these services are operated at actual counters, instead of websites, and often face a shortage of talents or funds. Our research product will solve this problem effectively. Regarding the WHAT factor, our research targets are things which are reused or are lent, as well as services, such as volunteer work for snow shoveling or the repair of houses damaged by floods. As for the HOW factor, our research target is limited to domains with simple algorithms, and applications with complicated algorithms are omitted, because it is difficult for end-users to define the business logic.

2.2 Web applications for Matching Service

There are many kinds of matching service sites in practical use and some of them become topics on TV or in articles in the newspapers in recent years in Japan. For example, crowd funding, crowd sourcing, reuse of secondhand articles, rental space, rental cars, real estate sales, special lectures such as English conversation for the Japanese, parking at a residential area, baby sitting and others.

In particular, two typical examples, namely rental rooms at homes, such as Airbnb, and ride-sharing, such as Uber and Lyft, are popular in many countries. Actually, in Japan, these services are considered illegal and limited to some areas which the government admits. The government intends to decide new rules and regulations for promotion of these services. However, groups of hotel managers or taxi drivers request the government to make the rules and regulations severe for fair competition.

Generally, many kinds of matching services become more and more popular as mobile phones have come into wide use recently. Many matching sites can be accessed from the outside or require registration of a photograph of target objects, such as rooms and clothes, because a user can take photographs by using a mobile phone easily. Furthermore, most web services for sharing economy are kinds of matching services.

Primarily, our target for end-user development is different from such matching services supported by IT professionals. For example, local governments support various events for ecological activities such as flea markets, but the effects are limited because of manual operations without IT. Furthermore, when volunteers are requested for natural disaster areas, the local government cannot support the volunteers effectively because of manual operations without IT also.

In order to make our research target clear, the following two criteria were introduced for the analysis of many kinds of matching domains from the viewpoint of the users:

* Request for the trustworthiness of participants

* Request for quality of things or services

These criteria are essential for matching domains. For example, the reuse support service where an article to be reused is given free to a requester by a provider is characterized by <low, low> as values of two criteria in Fig.1. Another reuse support service where an article to be reused is sold to a requester by a provider is characterized by <high, high>. In this case, there may be troubles with payment. Another reuse support service where the requester pays the provider via a website is characterized by <low, high> because the risk of trouble in payment is reduced. Finally, voluntary snow shoveling where a volunteer may visit a house in which an old person lives alone is characterized by <high, low>.

Typical examples of sharing economy may be considered, namely accommodation for travelers and ride-sharing, and the related services. A hotel reservation site is positioned on <low, low> because hotels are trustworthy and the quality of a hotel room is correspondent with the rate. A taxi calling site is positioned on <low, low> also because taxi drivers are trustworthy and the quality of services is assured by taxi companies. On the other hand, sites for rental rooms at home and sites for ride sharing are positioned on <high, high> because the trustworthiness of service providers and the quality of the rental rooms or ride sharing services are not known.

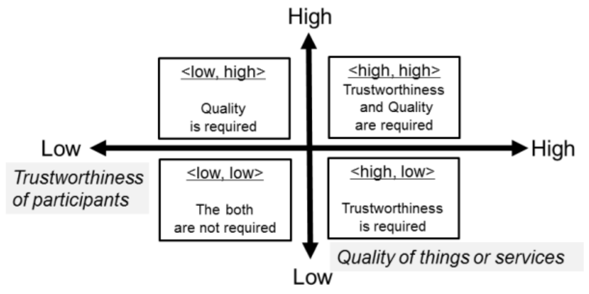
High
<low, high>
Quality is required
<high, high>
Trustworthiness and Quality are required
Low
High
Trustworthiness of participants
<low, low>
The both are not required
<high, low>
Trustworthiness is required
Quality of things or services
Low

Figure 1. The criteria based on user view

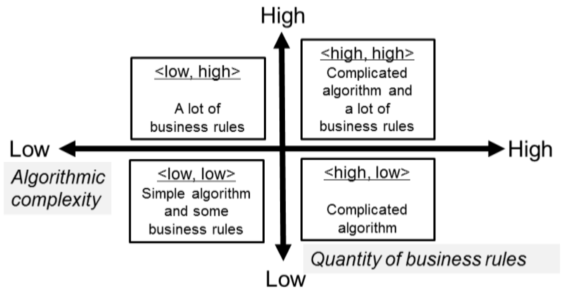

Figure 2. The criteria based on system view

The classification with these two criteria is based on a user view. The end-user development, however, requires a system view instead of a user view because the success of end-user development depends on whether it is easy for end-users to develop web applications for matching sites. Therefore, the following other two criteria were introduced:

* Algorithmic complexity
* Quantity of business rules

The possibility of end-user development depends on these criteria. If the business logic which the end-users define requires a complicated algorithm, the automatic generation of the corresponding source codes will become difficult. If the number of business rules increase too much, it may be difficult for end-users to define the business logic consistently.

Considering the examples above, the reuse support service where an article to be reused is given free to a requester by a provider is characterized by <low, low> in Fig.2 because the matching algorithm will be simple if the first requester for the registered article obtains the service, and business rules will be a few if the first requester negotiates a detailed procedure for getting the service from the provider directly without mediation of the website. A reuse support service where an article to be reused is sold to a requester by a provider is classified into several categories. Sites belong to <low, low> if the price is fixed, the first requester gets the service and the requester pays the money to the provider directly, without mediation of the website. The matching algorithm will become more difficult if the price is decided at an auction. The business rules will increase if the requester pays the provider via the website.

Voluntary snow shoveling is classified into several categories as well. Sites belong to <low, low> if the number of volunteers are fixed and the volunteers are admitted until the number is filled. The matching algorithm will become more difficult and the business rules will increase if volunteers are registered in advance before volunteers are requested for an actual project, and the volunteers for the project are selected out of the registered members.

Sites for rental rooms at homes are positioned on <low, high>. The matching algorithm is simple because the service requester selects a room from registered objects by himself/herself. However, a lot of business rules should be processed because many kinds of properties of each object are registered and used for searching through the registered objects database. Sites for ride-sharing are positioned on <high, low>. The matching algorithm is complicated because the system must select the service provider which is nearest to the requester.

Consequently, our research target in the system view for end-user development is limited to the <low, low> domain in which the algorithm is simple and there are not too many business rules. The <low, low> domain in the user view is preferable for end-user development, because websites which end-users develop and put into practice do not have the sufficient ability to solve troubles with the trustworthiness of participants and/or the quality of the things or services.

A reuse support system for free articles was selected for the experience with the end-user development as one of web applications which satisfy the requirements mentioned above. This is because it is expected that IT (information technology) contributes to saving resources and environmental preservation for a sustainable society. For example, on a reuse support by a local government, the number of articles and the number of participants will increase if business professionals of the local government develop the application for the web site in which providers can register articles to be reused and requesters can search the list of registered articles for their own use easily.

Recently, volunteer support systems have been selected for the experience with the end-user development as another web application which satisfy the requirements mentioned above. Natural disasters such as great earthquakes, tsunami, floods and mudslides by heavy rain, sometimes strike areas in Japan. At that time, many people want to help refugees as volunteers by clearing the garbage from houses and supporting their lives at places of refuge. These volunteers are not actually dispatched to suitable sites where they are required because the local government must process many kinds of requests through a small staff and the management of volunteers and relief supplies tends to be late. The volunteers could be dispatched quickly to suitable sites and relief supplies could be dealt quickly to the people who need them if the local government staff can develop and operate a web application easily for matching volunteers and sites that require a lot of support, or relief supplies and refugees.

3. APPLICABILITY OF DOMAIN-SPESIFIC FRAMEWORK

We have already tried to develop the technologies for end-user development of web applications which are used for websites. The typical five experiences are shown in Fig. 3. The horizontal axis implies the programming granularity. The granularity is small if the business logic is described in programming language such as Java. On the other hand, the granularity is large if the business logic is described in domain-specific language such as business words. The scripting languages are positioned on the middle. Generally, the description level will tend to the non-programming level as the granularity becomes larger. The vertical axis implies the applicability of the domain-specific framework. Generally, the domain which end-user technologies will be applied to is limited as the applicability becomes lower.

In the first case study in the highest applicability in Fig. 3, a domain-specific framework was developed and applied to the development of a reuse support system (Li et al. 2012). This is a component-based framework. An application is constructed not by programming but by defining the components. The end-user models DB, GUI and the business logic by using visual modeling tool and keeps these results in the JSON model, the CSS model, and the SQL statements. The framework runs applications by interpreting these models.

In the practical modeling process, first, the end-user needs to create three tables for management of members, objects, and requests for registered articles, while defining the name and the data type of each column in the tables. Next, the GUI is defined with the information including the items, item attributes, display style, an input/output, etc. while creating the components by dragging and dropping the elements and setting up the attributes of an element. Then the pages are composed of the components. The functions for the business logic are classified into four categories: {validation, DB operation, page transition or navigation}. For example, the SQL statements for the business logic are generated by setting the reference between the data models and the relationship of the data model and the GUI. These relationships are defined by connection lines between objects.

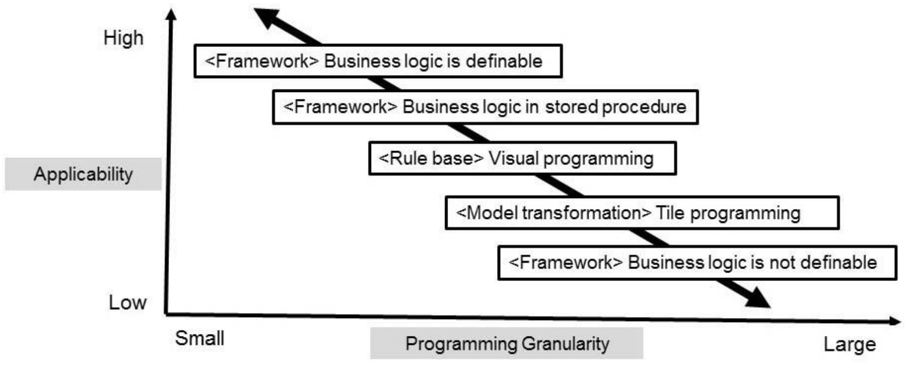
High
Applicability
Low
Small
Programming Granularity
Large
<Framework> Business logic is definable
<Framework> Business logic in stored procedure
<Rule base> Visual programming
<Model transformation> Tile programming
<Framework> Business logic is not definable

Figure 3. The relation between applicability and programming granularity

In the second case study in Fig. 3, a domain-specific framework was developed and applied to the development of a reuse support system (Xu et al. 2014). The main part of procedures for business logic is included in the database. The functions of the reuse support system were defined as follows: (1) The user registration and modification of the user information, (2) The user identification by using the mail address, (3) The article registration and update of the article information, (4) The search and request for articles, (5) The process for making a decision of the receiver. The system was developed based on ASP.NET MVC model. The DB server was implemented by using the SQL server. The six DB tables were introduced for the information of users, articles, requests for registered articles, the kind of article, methods for passing articles, and management of matching state transition.

Let’s consider the following typical rule about article registration:

“If the user has already registered some articles the number of which is the upper limit,
the user cannot register another one.”

This rule requires database access and will be processed as follows: (1) The information about the articles registered by the user is retrieved, (2) The number of registered articles is counted, (3) If the number is less than the upper limit, the article is registered, (4) If the number is the upper limit, the article is not registered.

Almost all processes execute database access as this rule does so. Then business logic including this rule is implemented in a stored procedure, instead of using SQL statements. The maintainability is improved for frequent business rule modification. The description of business rules which end-users input by using a visual tool should be transformed into the stored procedures by an automatic code generation tool. The four basic stored procedures {select, insert, update and delete} for each table of a reuse support system are prepared. The other stored procedures are generated by modifying these basic stored procedures.

In the third case study in Fig. 3, a domain-specific rule-based system was developed and applied to the development of a reuse support system (Iiyama et al. 2015). The main part of procedures for business logic is expressed in rules and then transformed into an Android application. It is convenient for a provider of an article to take and register the photograph by using a mobile phone. End-users define business rules of the web applications by embedding the value components in process components and condition components. The condition components are used for construction of the condition part of a business rule. The process components are used for construction of the action parts of a business rule. Some of them are prepared for CRUD operations for DB access. For these rule definitions by using a visual tool, it is necessary for end-users to understand a few programming concepts.

In the fourth case study in Fig. 3, model transformation technologies based on a model-driven approach were developed and business logic of an application system was described with tile programming (Chusho et al. 2011). The model transformation technologies were applied to development of a small-scale library system since this approach is suitable for workflow-centered back-end systems.

A web application model was designed by using the visual modeling tool with a palette of general components. Some pages were defined, namely the top page of the application, the form for the registration of a book, the form for the definition of business logic of the registration of a book, the form for the announcement of completion of the registration of the book, and the form for database manipulation. Tile programming was adopted and then the system prepares some templates for instruction statements. End-users construct the business logic by combining these templates. This model is transformed into the design model under the condition of the particular platform of the Struts 2 framework. In examples of mapping, some of them are as simple as one page being mapped into one JSP document. Others are as complex as the business logic being mapped into Java classes and a Struts.xml document. This transformation program is described in XSLT since both models are stored in XML in the system. The code generation tool generates the source codes for the application which include Java classes with class names, properties and methods, and JSP files. On the other hand, a set of files which is common to web applications, is not included in the design model, and is appended to the source codes by the code generation tool. The web application model is composed of 25 form definitions, 8 business logic definitions and 3 database table definitions. It was confirmed that it was easy for end-users to construct the web application model.

In the fifth case study in Fig. 3, a domain-specific framework was developed and applied to the development of a small-scale library system without complicated business rules (Hashimoto et al. 2012). This framework was developed based on Ruby on Rails since a simple application with basic access to the database was generated by using scaffolding in Rails. The framework provides visual tool with GUI (graphical user interface) for definitions of database tables. DB search functions and validation functions for input data values were added to the framework. Then end-users can select these functions if the applications needs them. On the other hand, end-users cannot define arbitrary business logic since the framework promotes end-user development with a non-programming method.

Consequently, Fig. 3 shows a tradeoff between the applicability of domain-specific frameworks and programming granularity. Then, such process improvement of business logic definition as the applicability becomes higher for the same programming granularity is important for promotion of end-user development.

4. THE TEMPLATE FOR TRADEOFF SOLUTION

4.1 The Template for Business Logic Definition

For considering web application generation process for solutions of the tradeoff problem mentioned above, an application architecture is limited to a typical three-tier architecture. Then, web applications are defined using a business logic definition tool and a CRUD definition tool. Our web application generation process, named the ABC development model, is expressed as follows:

Application = Business logic + CRUD

A web application is defined at the logical level by end-users as follows:

1. The user interface and the DB tables are defined.
2. The business logic is defined based on the above definitions.

The business logic at the requirement specifications level is mapped into the combination of user interfaces (UI), business logic (BL) and databases (DB) based on the typical three-tier architecture. The following template is introduced because the UI-driven approach is suitable for the end-user development:

1. UI: The system gets a request from a client.
2. BL: The system processes the request.
3. DB: The system accesses the database.
4. BL: The system processes the results.
5. UI: The system displays the results.

This template, named UtoU, implies that the typical process is {UI > BL > DB > BL > UI}. It is easy for an end-user to understand this process because the end-user as a business professional or domain expert is familiar with the following typical work flow such as getting a resident's card: (1) A client fills out an application for the card and hands it to the service counter at a city office. (2) A clerk at the service counter checks the application and passes it to a computer operator in the back. (3) The operator inputs the information about the client and gets the resident's card. (4) The clerk receives it and confirms the contents. (5) The clerk finally hands it to the client. Therefore, the UtoU template is considered to contribute to overcoming the tradeoff between the applicability and programming granularity in Fig. 3 because the process of business logic definition is improved as the applicability becomes higher for the same programming granularity. A case study for this process using the UtoU template is described in the next section.

4.2 A Case Study

Recently, volunteer support systems have been selected for the experience with the end-user development. Many people want to help refugees as volunteers, clearing garbage from houses and supporting their lives at places of refuge after a natural disaster. These volunteers are not actually dispatched to suitable sites which require them because the local government must process many kinds of requests through a small staff and the management of volunteers and relief supplies tends to be late. The volunteers could be dispatched quickly to suitable sites if the local government staff could develop and operate a web application for easily matching volunteers with sites that require a lot of support, or relief supplies for refugees.

Therefore, an application framework for a volunteer support system was developed (Yokoi et al. 2015). The design concepts are simple matching algorithms and minimum requirements for practical application development. If a volunteer select a project after the project is registered, the project manager decides quickly whether or not to accept the volunteer. If the project manager selects volunteers from a list of candidates who are registered in advance, the selected volunteer decides quickly whether or not to accept the request. This framework was implemented by using JavaEE and applied to development of a volunteer support system.

Let's consider applying the UtoU template to a definition of registration of volunteers for supporting refugees of a natural disaster. First, the first user interface of the template: { *UI* > BL > DB > BL > UI} is defined by listing all input columns at the logical level as follows: { the identification number of the specified voluntary project, the member identification number, the name of the volunteer, the check box for a declaration of observing the rules on volunteer activities}. Next, the last user interface of the template: { UI > BL > DB > BL >*UI*} is defined by listing all output columns at the logical level as follows: {the name of a volunteer, the identification number of the specified voluntary project, the registration number of the project}. Then, the DB table of the template: { UI > BL > *DB* > BL > UI} is defined by listing all columns of each record at the logical level as follows: { the identification number of the voluntary project, the registration number, the member identification number of the volunteer, the date, the status of "registration" or "rejection" }. Then, the first business logic of the template: { UI > *BL* > DB > BL > UI} is defined by listing checks of input columns and internal processes as follows: { All input columns are filled in; All input data are valid; The identification of the registration date; The confirmation of being not full; The generation of the registration number;}. Finally, the last business logic of the template: { UI > BL > DB > *BL* > UI} is defined by listing all output columns and internal processes as follows: { The name of a volunteer; The identification number of the project; The registration number on the project;}.

5. CONCLUSION

It is preferable for business professionals themselves to support their web applications since web applications must be modified frequently based on users' needs. Therefore, technologies for end-user development have become important for the automation of the routine work. This paper described domain-specific technologies for the end-user development of web applications with the three-tier architecture of the user interface, business logic and database. The matching domain was selected for case studies and analyzed. Then, the applicability of the domain-specific technologies was analyzed. For improving a tradeoff between the applicability and programming granularity, the UtoU template was introduced as the applicability becomes higher for the same programming granularity. After the DB table definitions and GUI definitions, business logic is defined by using the UtoU template. It was confirmed that these technologies are effective for end-user development.

REFERENCES

Brancheau, J. C. and Brown, C. V., 1993. The Management of End-User Computing: Status and Directions. *In ACM Computing Surveys*, Vol. 25, No. 4, pp. 437-482.

Burnett, M., 2012. End-User Software Engineering and Why It Matters. *End-User Computing, Development, and Software Engineering*, A. Dwivedi and S. Clarke, Eds. IGI Global, PA, USA, pp. 185-201.

Cotterman, W. W. and Kumar, K., 1989. User Cube: A Taxonomy of End Users. *In Communications of the ACM*, Vol. 32, No. 11, pp. 1313-1320.

Chusho, T. et al, 2011. End-User-Initiative Development with Domain-Specific Frameworks and Visual Modeling. *Proceedings of the 10th International Conference on Software Methodologies, Tools and Techniques*. Saint Petersburg, Russia, pp. 57-71.

Chusho, T., 2015. The Classification of Matching Applications for End-User-Initiative Development. *Proceedings of The International MultiConference of Engineers and Computer Scientists 2015*. Hong Kong, China, pp. 476-481.

Fischer, G. et al, 2009. Metadesign: Guidelines for Supporting Domain Experts in Software Development. *In IEEE Software*, Vol. 26, Sep./Oct., pp. 37-44.

Hashimoto, R. and Chusho, T., 2012. Evaluation of End-User-Initiative Web Application Development Technique. (in Japanese), *Proceedings of Forum on Information Technology*. Tokyo, Japan, B-033.

Iiyama, H. and Chusho, T., 2015. Domain Specific Android Application Supporting Tool, (in Japanese). *In IEICE Technicl Report*, Vol. 114, No. 501, KBSE2014-53, pp. 7-12.

Ko, R. A. et al, 2009. Guest Editors' Introduction: End-User Software Engineering. *In IEEE Software*, Vol. 26, Sep./Oct., pp. 16-17.

Li, J. and Chusho, T., 2012. A Web Application Framework for End-User-Initiative Ddevelopment with a Visual Tool. *Proceedings of 2012 IAENG International Conference on Software Engineering*. Hong Kong, China, pp. 816-822.

Scaffidi, C. et al, 2005. Estimating the Numbers of End Users and End User Programmers. *Proceedings of The 2005 IEEE Simposium on Visual Languages and Human-Centric Computing*. Dallas, Texas, pp. 1-8.

Sinha, A. P. and Jain, H., 2013. Ease of Reuse: An Empirical Comparison of Components and Objects. *In IEEE Software*, Vol. 30, Sep./Oct., pp. 70-75.

Sprinkle, J. et al, 2009. Guest Editors' Introduction: What Kinds of Nails Need a Domain-Specific Hammer?. *In IEEE Software*, Vol. 26, July/Aug., pp. 15-18.

Sutcliffe, A. and Mehandjiev, N. (Guest Ed.), 2004. End-User Development. *In Communications of the ACM*, Vol. 47, No. 9, pp. 31-32.

Xu, J. and Chusho, T., 2014. End-User-Initiative Development for Web Application. (in Japanese), *In IEICE Technicl Report*, Vol. 113, No. 475, KBSE2013-81, pp. 13-18.

Yokoi, S. and Chusho, T., 2015. Matching Systems Construction Technique for End-User-Initiative Development. (in Japanese), *Proceedings of The 77th National Convention of IPSJ*. Kyoto, Japan, 1L-08.