A Web Application Framework for End-User-Initiative Development with a Visual Tool

Jing Li, and Takeshi Chusho

Abstract—**In recent years, many web applications are used in various business fields. These web applications are often developed on the basis of a framework with reusable components. Furthermore, it is increasingly required that business professionals build their web applications by themselves. In this paper, a web application framework for end-user-initiative development is proposed to support small and medium-sized businesses. A web application is developed first and the domain independent parts are extracted as a framework. Then the web application is rebuilt with component-based architecture for increasing the domain independent parts and the web application framework is completed. Finally a visual tool for designing a web application without programming is developed and the realization of end-user-initiative development is confirmed.**

Keywords—**End-User, Web Application, Framework, Component-base, Visual Tool**

I. Introduction

In recent years, many web applications are used in various business fields. However, it is a big workload to develop each web application from scratch. Therefore, it has become more efficient to develop a web application on the basis of a framework that provides a series of foundations [1]. Commonly, information systems or web applications have been developed by technology experts, and used by a limited audience. With the widespread use of personal computers and workstations, it is increasingly required that business professionals build their web applications by themselves [2] [3].

Therefore, it has become important to quickly develop small-scale web applications at a low cost which target the small business sector and individuals responsible for business professionals on a daily basis. With that viewpoint, we have studied development and maintenance techniques that can be led by business experts in their field operations [4].

II. Target and Approach To non-programming

We propose an end–user-initiative framework to accommodate frequent changes in service. The framework is intended for small-scale web applications that are harder to secure the budgeting necessary to outsource. We aim to enable business experts to build web applications using a framework based on their business knowledge [5].

Thus, in order to develop and evaluate a web application framework that can make an end-user build a web application using a GUI modeling tool without any coding, we propose the approach as follows:

- Select the scrap exchange system as an exercise application, and build it for a holistic view of software development.
- Rebuild the exercise application using JSON[6] and Ajax[7] technologies, and extract the components from the exercise application, and design the API to connect the view and the logic.
- Build a framework based on the extracted components.
- Develop a visual tool that can enable the end-user to build the web application.

III. Experiments and Results

A. The scrap exchange system

We have taken up the scrap exchange system as an exercise because we acknowledge the existence of environmental problems. IT technology is expected to be part of a maintainable social realization, the saving of resources, and the achievement of environmental preservation (Green-by-IT). For example, because the scrap exchange shop is voluntarily managed by a town association or a local self-governing body, if a person working at the shop can build a website for the scrap exchange easily, it will be expected to achieve the Green-by-IT.

In our study, we assumed that the scrap exchange system mediates those who offer a disused article, and those who want to utilize it. By using the scrap exchange system, the user can register, apply for, or retrieve the scrap information at a PC terminal. Fig. 1 shows the page transition of the scrap exchange system. The system is composed of 10 pages, and is comprised of the following user experience:

- Register E-Mail address.
- Register and update the personal information of the user.
- Login to the scrap exchange system.
- Register and update the scrap information.
- Search the scrap information.
- Apply for a scrap.

Jing Li is with the Software Engineering Laboratory, Graduate School of Science and Technology, Meiji University, Japan.(e-mail: libing_anne@hotmail.com).
Takeshi Chusho is with the Software Engineering Laboratory, Graduate School of Science and Technology, Meiji University, Japan.(e-mail: chusho@cs.meiji.ac.jp, phone: +81-44-934-7449).

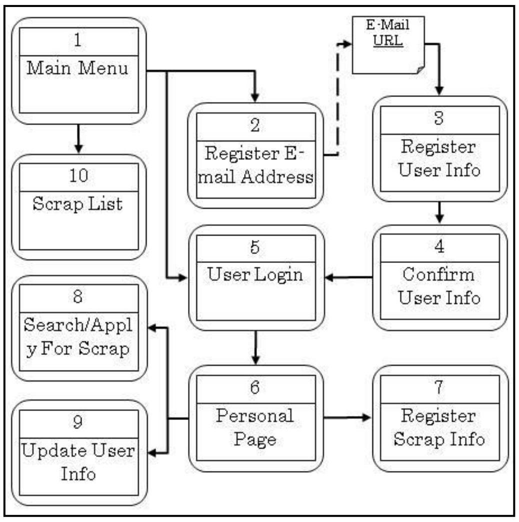


Fig 1.The page transition of the exercise application.

B. Architecture

A three-tiered architecture is used for this system and is constituted of the presentation layer which offers a user interface, the application layer which performs the processing of an application, and the data layer which manages data. At the implementation level, it respectively corresponds to a web browser, an application server and a database server.

This system was designed to adopt the MVC pattern. The roles of MVC are separated definitely. It corresponds to the model holding the state of application, the view which manages the display and output, and the controller which receives an input and controls the view and the model according to the contents of the input.

C. The experiments and results

In this study, we implemented 2 versions of the exercise application for this experiment. Domain dependency was measured for a major portion in the MVC pattern. In order to build a system using a GUI modeling tool, it is necessary to convert the GUI model to program source codes which have domain-dependency. Therefore, it is ideal that the implementation has low domain-dependency. Furthermore, the portion which has domain-dependency should be easy to convert from the GUI model.

The exercise application version 1

Version 1 was developed with a simple framework, the EcoFW, based on JSP/Servlet model. Fig. 2 presents the system architecture using the framework EcoFW. In EcoFW, the logic class is bound to the URL from the client. The view and logic are mapped by EcoFW, which plays the C role in the MVC pattern. Additionally, the DAO pattern is adopted, with the ability to facilitate the operation of the database. A business-delegate design pattern is adopted and business logic is processed in the Logic class. The view is written in JSP. In Fig. 2, the solid lines indicate the domain-independent portion, and the dotted lines indicate the domain-dependent portion.

In the domain dependence portion of version 1, one JSP corresponded to one logic class. Moreover, the SQL was created in advance, and was saved in the properties file.

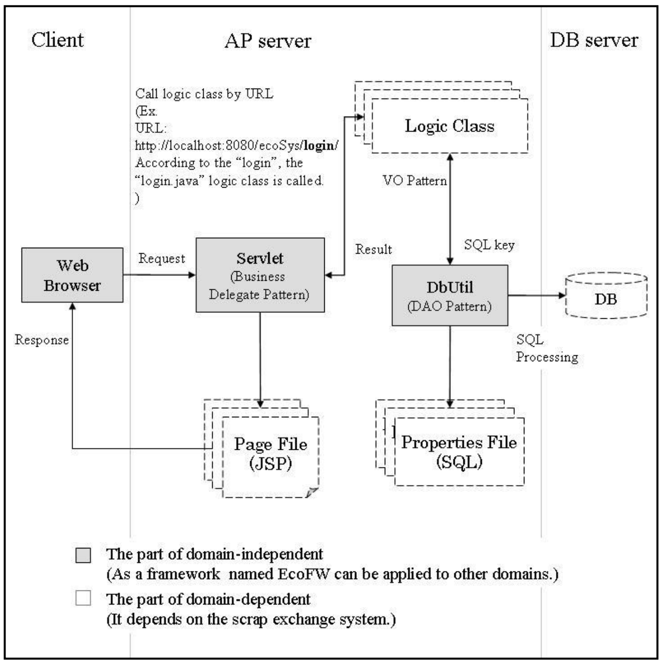

Client
AP server
DB server
Call logic class by URL (Ex. URL: http://localhost:8080/ecoSys/login/ According to the "login", the "login.java" logic class is called.)
Logic Class
VO Pattern
Result
SQL key
Web Browser
Request
Servlet (Business Delegate Pattern)
DbUtil (DAO Pattern)
DB
Response
SQL Processing
Page File (JSP)
Properties File (SQL)
The part of domain-independent (As a framework named EcoFW can be applied to other domains.)
The part of domain-dependent (It depends on the scrap exchange system.)

Fig 2. The Architecture of our framework (Version 1).

Table I presents the domain-independent portion in the implementation. Since the implementation of version 1 does not have components, the source of jsp and java, which is domain dependent, exists in large quantities. So, it is difficult to convert the GUI modeling for the end-user into an applicable domain-dependent portion.

Table I
Domain-independent part of version 1.

MVC	Function	Implement	Domain dependent/ independent
V	View definition	Jsp	dependent
	View initialize		dependent
M	Business logic definition	SQL	dependent
	Business logic processing	DbUtil	independent
C	Controller	Servlet	independent

The exercise application version 2

Through the development of version 1, we concluded that for implementing a web application, certain specifications are required as follows:

- ■Database design (business logic definition)
- ■Page layout (view definition)
- ■Business logic design (business logic definition)
- ■Data resources of a page layout (view initialization)
- ■Page transition figure (view definition)

Even if the GUI modeling tool is offered to the end-user, these specifications need to be visualized and saved as some kind of data model, like XML or JSON.

A web application can be built by converting modeling results into a source code or configuration file which can be recognized by our framework. Therefore, for the conversion we defined the domain-dependency of version 1 in the configure file which is independent of the programming language. If the configure file can be read by the framework, it is impossible to realize a web application from the GUI modeling by the end-user.

As for a small-scale system like the scrap exchange system, the business logic is responsible for the exchange of information between the DB and user interface, and relies on the SQL statements. In version 1, most of the business logic is understood to operate the DB with SQL statements.

Therefore, we can believe the definition of the business logic can be modeled by the SQL statements. Of these, the login function must be implemented as a common feature of the framework.

In version 2, we strengthened the function of the framework by modifying EcoFW to component oriented framework. It can make us construct a project not by programming but by defining the components and the SQL statements in the configuration file.

Therefore, when a GUI modeling method is provided by a visual tool, we can convert the GUI model to the configuration file used in EcoFW. It is possible to build a web system by using a GUI modeling tool for the end-user.

Fig. 3 shows the architecture of the framework in version 2. The gray portion in Fig. 3 shows the framework EcoFW. In version 2, the view control engine was developed on the client side, which comprises Page-driver, Component-Driver and Logic-driver. They work as follow.

Page-driver sends a request for getting the page definition file to the server, which defines a group of the components of the page.

After the Page-driver receives the response, the Component-driver sends a request to the server for getting the component definition file for each component one by one, which is defined in page definition file. In fact, the content of component definition file is a group of elements.

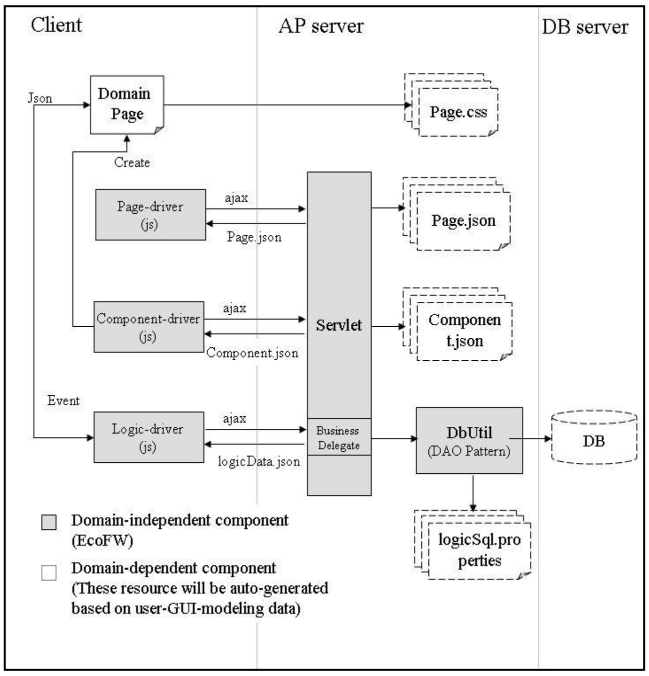
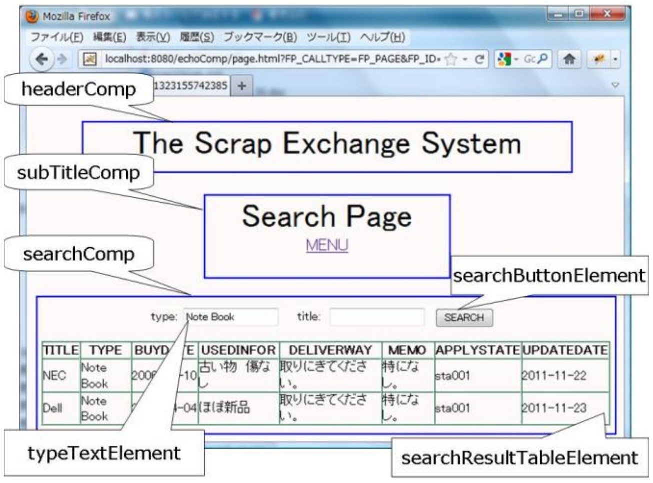


Fig 3. The architecture of the framework (Version 2).

After the Component-driver receives the response, it creates the elements which are defined in component definition file one by one. When these processes are completed, a domain page is created dynamically as a GUI. The domain page links to the CSS files for the style information.

When the event is driven in the domain page, the Logic -driver sends a request to the server to process the busses logic module. After it receives the response as a result, the Logic-driver reflects the result to the domain page. On the server side, the business logic is processed by operating DB with a SQL statement.



TITLE	TYPE	BUYD TE	USEDINFOR	DELIVERWAY	MEMO	APPLYSTATE	UPDATEDATE
NEC	Note Book	200 -10	古い物 傷なし	取りにきてください。	特になし。	sta001	2011-11-22
Dell	Note Book	-04	ほぼ新品	取りにきてください。	特になし。	sta001	2011-11-23

Fig 4. The structure of domain page (GUI).

Fig. 4 shows the domain page as an example. Next, we will describe how the page is created and how the business logic is processed in detail by the framework EcoFW.

In version 2, the part of the view is made into the components, and defined to the configuration file of the JSON model. In the configuration file, a page is defined by, and constituted of a group of the components. There is a group of the elements for each component. The next is the definition of the domain page in Fig. 4, search page.

■The definition of search page(searchPage.json)

```
01 {"id":"goodlistPage",
02  "componentIDs":[
03      "headerComp",
04      "subTitleComp",
05      "searchComp"
06   ]
07 }
```

In the above definition, line 01 defines the page id. From line 02, a group of the components are defined. In this page, there are three components. Those are headerComp, subTitleComp and searchComp. The serachComp component is defined as follows:

■The definition of the searchComp component
(searchComp.json, a part of is omitted)

```
01{  "id":"searchComp",
02  "elements":[
03    { "id":"typeLableElement",
04     "type":"lable",
05     "caption":"type:"},
06    { "id":"typeTextElement",
07     "type":"text",
08     "caption":""},
09 . . .
10    { "id":"searchButtonElement",
11     "type":"button",
12    "caption":"SEARCH",
13    "eventsList":[
14  {"name":"onclick",
15   "func":"F_CRUD",
16   "module":"search.select",
17       "params": " typeTextElement;titleTextElement",
18   "view":"searchResultTableElement"}
19      ]
20    },
21 …
22   { "id":"searchResultTableElement",
23    "type":"table",
24   "caption":""}
25  ]
26}
```

In the definition of the search component, a group of the elements are defined. From line 02, one element has a unique id and has the “type” property. The value of the “type” property is provided by the framework in advance. In line 06-08, a “text” element for input is defined. In line 22-24, a “table” element for displaying search result is defined. A “button” element is defined in line 10-20.

These elements defined in the element group will be created one by one by the Component-driver. The property “eventsList” is defined in line 13, and an event “onclick” is defined in line 14-18.

In this case, when the button is clicked, the Logic-driver sends a request to the server with the parameters. One of the parameter is the “module” which is defined in line 16. The “module” parameter is the key of a SQL statement. It tells the server which logic is processed. The other parameters are defined in line 17, which are the search conditions. The Logic engine gets the values of elements, which are defined in “params” property and sets these as the request parameters. The request parameters are sent as follow:

[… moduel=search.select&typeTextElement=PC& …]

When it receives the response from the server, the logic engine reflects the result to the element which is defined in “view” property in line 18. The result is a JSON object and is defined as follow:

■The definition of the result

```
01  { "result":"0",
02   "titles":[ "TITLE","TYPE",…,"UPDATEDATE"],
03    "records":[
04     ["NEC "," Note Book ",…,"2011-11-22"],
05     ["NEC "," Note Book ",…,,"2011-11-23"]
06     ]
07   }
```

If it is success, the value of property “result” will be “0”. And the result data defined by property “titles” and “records” in line 02-06, which is set to the table element “searchResultTableElement” in this case. The display of processing the business logic is shown in Fig. 4.

D. *Considerations*

We built two versions of the scrap exchange system. The version 1 performed the functions of the scrap exchange business. We also knew the necessary specifications to build a web system. Therefore, it was possible to decide the functions of the visual tool support for the end-user. In addition, domain independence was separated from the exercise application, and created as a framework called EcoFW.

In version 2, we aimed to convert more domain dependence into independence until it became a part of the framework. Table II presents the domain-independent portion in the implementation. In order to measure the domain independency, we listed the detail functions based on the MVC pattern as shown in Table I and Table II. As a result of version 2, the domain independent functions were provided as a part of the framework. Another result was that domain dependence can be set up with the JSON model, the CSS model, and the SQL statements. These are easier to read and analyze than java, jsp, and javaScript of programming language, and allowed us approach to the visual tool. As for the business logic can be not played on the SQL statements, we are considering to develop the logic components like the function of sending email.

Table II
Domain-independent part of version 2.

MVC	**Function**	**Implementation**	**Domain dependent/ independent**
V	View definition	JSON file Style sheet(CSS)	dependent
	View initialize	Page-driver Component-driver	independent
M	Business logic definition	SQL	dependent
	Business logic processing	DbUtil	independent
C	Controller	Servlet Logic-driver	independent

IV. The Visual Tool

A. *Design concepts*

As the above mentioned, in order to assist the end-user, we have proposed a visual tool that is a GUI modeling tool. Before implementing the version 1 of the scrap exchange system, we had designed external specifications. These are a table definition document, a detailed design of the pages, the page transition diagram shown in Fig. 1, and the SQL statements for the business logic. As for end-user-initiative development, the end-user needs to design the external specifications of these by modeling a GUI. So the visual tool needs to be equipped with three basic functions of business-use DB creation, GUI specification design, and business logic creation.

Fig. 5 shows the support system for the end-user. The right part is the generated web application. The gray boxes in the right part imply the above developed framework version 2. The left part is the visual tool support application development for the end-user. By using the visual tool, the end-user designs a DB, GUI, and the business logics by modeling. Modeling results will be kept in the JSON model, the CSS model, and the SQL statements. The developed framework runs applications by interpreting the JSON file and the SQL statements.

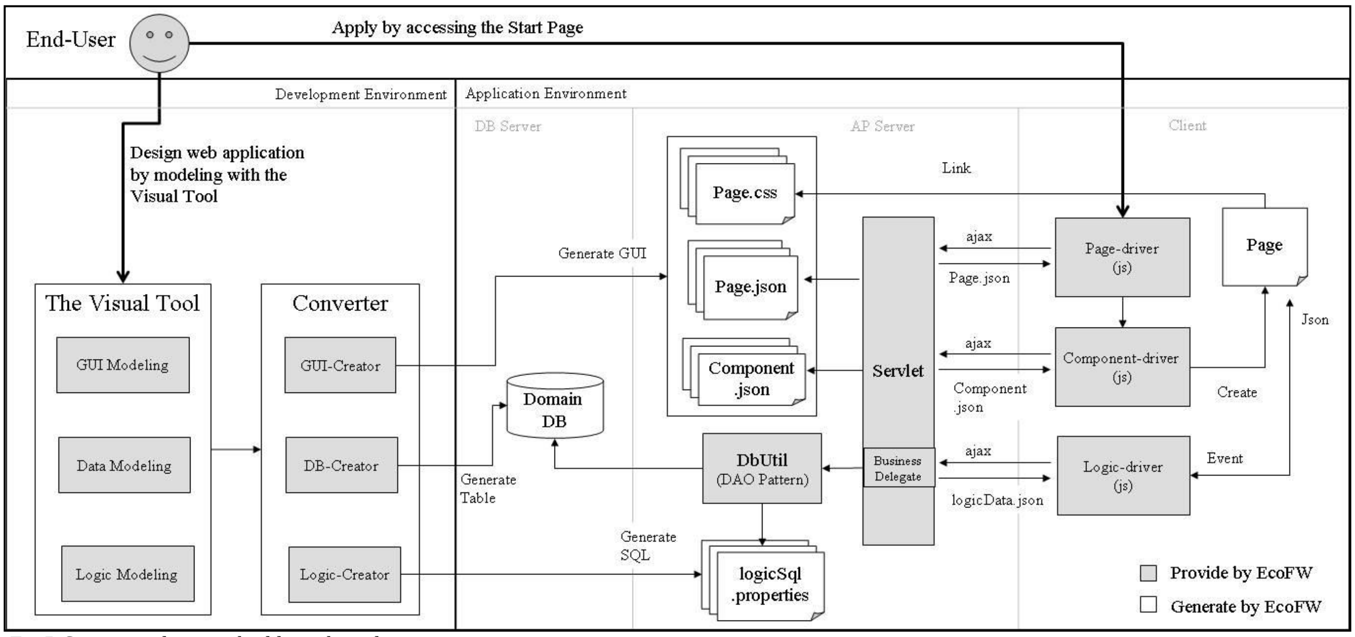


Fig 5. Support end-user to build a web application.

Specifications were examined based on each function of the visual tool. Fig. 6 shows the GUI specification of the visual tool. Depending on the capabilities provided by the visual tool, it consists of five pages of data modeling, component creation, page creation, logic modeling, and main page. The main page links to another four pages. The next section describes the each function of the visual tool.

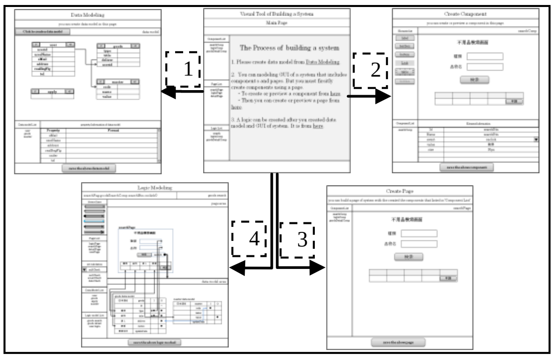


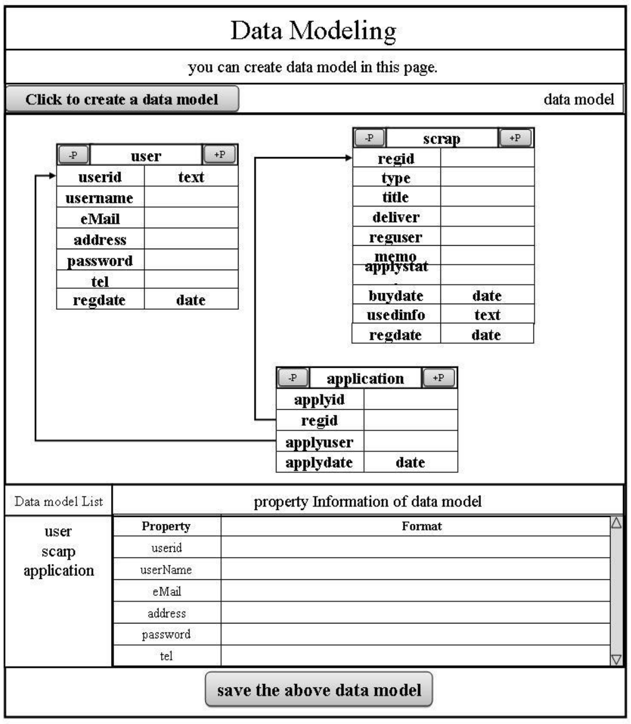
Fig 6. The page transition of the visual tool .

B. DB creation

When we built the scrap exchange system, a table was created according to a management target. The tables were created according to the target of user, scrap, and application. At the same, the end-user needs to create these tables by using the visual tool.

What is pointed out from 1 of Fig. 6 is the data modeling page which performs data model creation. A data model is connected to another data model to set for external reference by a line. The tables are created on the DB for every data model. If a data model is corrected, the table of correspondence will also be corrected automatically. The end-user needs to perform data modeling, before creating the business logic.

Fig. 7 shows the data models of the scrap exchange system according to the management targets, user, scrap and application. By clicking the button of +P and -P, the end-user


Data Modeling

you can create data model in this page.

Click to create a data model | data model

-P	user	+P
userid	text	
username		
eMail		
address		
password		
tel		
regdate	date	

-P	scrap	+P
regid		
type		
title		
deliver		
reguser		
memo		
applystat		
buydate	date	
usedinfo	text	
regdate	date	

-P	application	+P
applyid		
regid		
applyuser		
applydate	date	

Data model List	property Information of data model	
	Property	Format
user scap application	userid	
	userName	
	eMail	
	address	
	password	
	tel	

save the above data model

Fig 7. The data models of the scrap exchange system.

can add and delete a column of the data model. The left of column field is the name, and the right is the type. If the column's type is not inputted, it will be recognized as text type by default. According to the data models in Fig. 7, three tables will be generated into DB. Those are "user", "scrap" and "application". In addition, the table "application" refers to the table "user" by "applyuser" column is foreign key. And, it also refers to the table "scrap" by the foreign key of the "regid" column. The SQL statement is to create a table as follows:

■The SQL statement of creating the table "application" (a part of is omitted)

```
CREATE TABLE application
(  applyid text NOT NULL,
    …
    CONSTRAINT application_pkey PRIMARY KEY (applyid),
    CONSTRAINT application_fkey FOREIGN KEY (regid) REFERECES
scrap (regid)
    …)
```

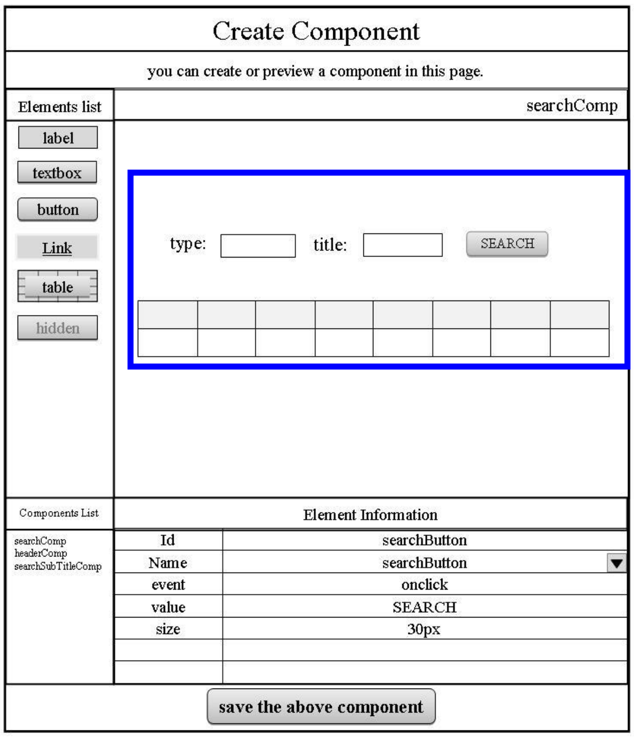
C. Modeling GUI

The above mentioned page specification has defined the information including the items, item attributes, display style, an input/output, etc. When the end-user uses the visual tool, it needs to create the components and the pages. The page is composed of the components. The components are created by incorporating the elements.

Component creation

Point 2 of Fig. 6 is the page where a component is created. The end-user creates a component by dragging and dropping the elements which are provided by the visual tool. However, the visual tool only provides the template of the element. If the attributes of an element are set up when creating a component, the component turns into a domain component. The domain component will be saved to a JSON file. Currently, although the elements provided are restricted to the exercise application, extendibility is held, and the element can be added as necessary.

Fig. 8 shows the sample of creating the search component which is also defined in the JSON file in chapter 3. In Fig. 8, the search component is named searchComp. It consists of two label elements, two textbox elements, a button element and a table element. The end-user adjusts the size and the position by dragging and moving the elements with the mouse. The elements are saved to a JSON file which is named searchComp.json. The format is similar to the introduced JSON file in chapter 3. The element's style information is saved to a CSS file which is named searchComp.css.



Id	searchButton
Name	searchButton
event	onclick
value	SEARCH
size	30px

Fig 8. The sample of creating the search component.

Page creation

Creating a page is pointed out by point 3 of Fig. 6. Here, the above created components are listed. The end-user incorporates these components and creates a page. In addition, the addition of a component and the adjustment of its position are performed by the operation of drag and drop. A JSON file is created to save a page. When the application is operated, the page is drawn by calling the JSON files of the components and the pages.

Fig. 9 shows the sample of creating the search page which is also defined in the JSON file in chapter 3. In Fig. 9, the search component is named searchPage. It consists of three components. They are named headerComp, searchComp and searchSubTitleComp and listed in the components list. The end-user adjusts the position by dragging and moving the components with the mouse. The components of the page are saved to a JSON file which is named searchPage.json. The format is similar to the introduced JSON file in chapter 3. The component's style information is saved to a CSS file which is named searchPage.css.

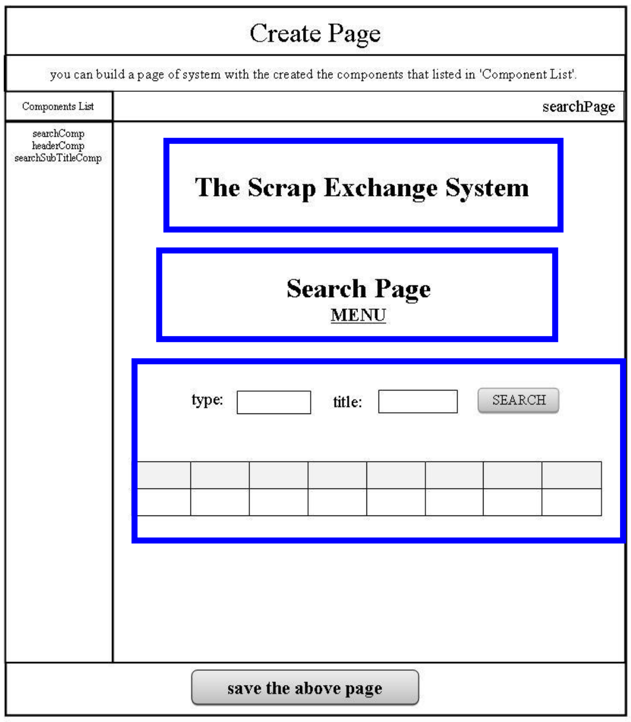


Fig 9. The sample of creating the search page.

D. Modeling business logic

For the business logics, the functions are divided into validation, DB operation, page transition, and otherwise navigation such as email transmission. In this study, we decided to implement in two phases. In the first stage, we implement the functions of the validation function, DB operation, the page transition. Moreover, the functions of domain dependence, such as email transmission, are added in the 2nd step. In this paper, we only describe the first step.

What 4 of Fig. 6 points out is a page to create the business logic. In this case, it aims at creating the SQL statement, which takes charge of the exchange of the information between user interfaces and database, setting up page transition relation and checking input data. The end-user needs to create a logic module for each event.

Generating the SQL statements for the business logic is achieved by setting the reference between the data models, and the relationship of the data model and GUI. The visual tool can represent relationships between objects using lines, and provide a set of access lines according to their relationships.

At this stage, there are six types of access lines. These are GD lines which represent the relationship between the data model and the elements in the GUI, DD lines which refer to the reference relationship between the data models, EM lines which correspond between the module and the events, EP line which point to the page transition, PP lines which point to the pass parameters between two pages, and EV lines for mapping of the input element and the validation function. In the future, we will provide the access lines to support the complex logic.

A SQL statement is created by setting up a relationship between the GUI and the data model like Fig. 10. And all the relations which the functional lines point out are held. The framework described in Chapter 3 achieves page transition and plays the business logic using the relationships and the SQL statements which were held.

Fig. 10 shows the logic creation of search scrap. The logic's specification is the same as the search button in Fig. 4. When the search button is clicked, the scrap table is searched, and the records are displayed to the table. The type's value and the title's value are used as the parameters.

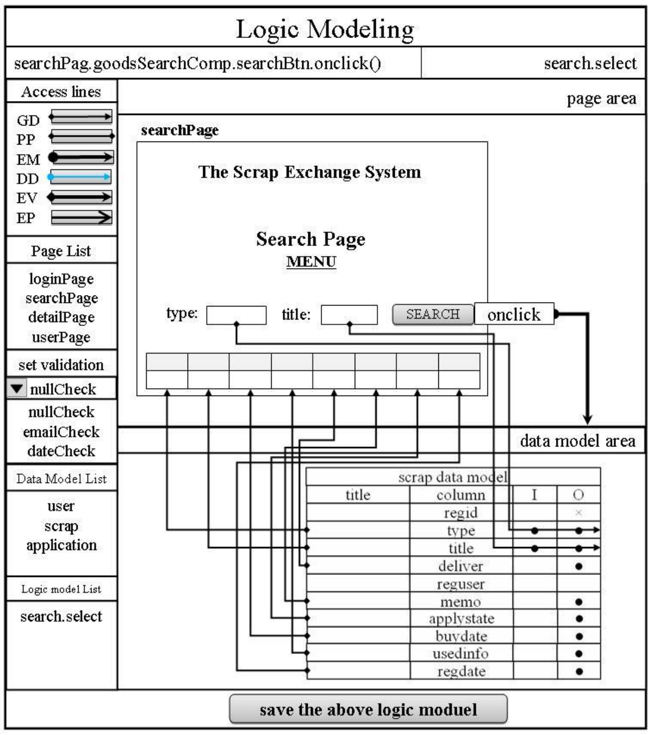


Fig 10. The logic creation of search scrap.

In Fig. 10, the created searchPage is dragged to the page area, and the created scrap data model is dragged to the data model area. The logic module is named search.select. The onclick event is selected, and pointed to the data model area by EM line.

As for a column of data model, the attribute of I or O can be set up. The attribute of I is a parameter for executing the module. It is accessed from an element of GUI by GD line. The accessed element's value is used as a parameter. The O attribute is an output column of executing the module. It is accessed to an element of GUI by GD line. Therefore, the SQL playing the logic in Fig. 10 is created based on the next format;

■The format of the SQL

```
SELECT
  O column, …
WHERE
  I column = element's value, …
FROM  data model
```

In the case of the logic including update, delete and insert, other formats are applied.

As the same time the above SQL statement is generated, input information, output information and module name will be added to fill the JSON file of the searchComp component.

E. *Considerations*

When the specifications of this visual tool were examined, they were designed to have extendibility. From this point, it will be thought that more function can be achieved. Also, we are examining the possibility of creating the element and the SQL statement by the end-user alone.

V. Conclusion

As part of the study of a web application framework for end-user-initiative development, we developed a web application as an exercise application—the scrap exchange system—and performed an evaluation and expressed considerations.

We separated a domain-independent framework called EcoFW from the version 1. In addition, we extracted more domain independence from the version 2, and converted it into our framework. In version 2, the domain dependence can be set up with the JSON model, the CSS model, and the SQL statements.

The visual tool was proposed based on the experimental result of the version 1 and the version 2. We will evaluate it in next stage.

References

[1] I. Crnkovic, et al., *Specification, implementation, and deployment of components. Communications of the ACM*, 45, 10(2002), 35-40.

[2] J. Sprinkle, M. Mernik, J. Tolvanen and D. Spinellis, Guest editors' introduction: What kinds of nails need a domain-specific hammer?, *IEEE Software, 26, 4 (July/Aug. 2009)*, 15-18.

[3] A. J. Ko, R. Abraham, M. M. Burnett and Brad A. Myers, Guest editors' introduction: End-user software engineering, *IEEE Software, 27, 5 (Sep/Oct. 2009)*, 17-17.

[4] Feng Zhou and Takeshi Chusho: A Web Application Framework for Reservation Systems and its Reusability Evaluation, *Proc. The 2009 IAENG International Conference on Software Engineering (ICSE'09)*, pp.1027-1032 (Mar. 2009).

[5] Jing Li and Takeshi Chusho: *A Web Application Framework for End-User-Initiative Development with Domain Knowledge*, IEICE The Technical Report Vol.111, No.282, Knowledge-Based Software Engineering(KBSE) KBSE2011-39,19-24(Nov. 2011) (in Japanese)

[6] "Ajax",http://ja.wikipedia.org/wiki/Ajax, (2011.7.27）.

[7] "JSON", http://www.JSON.org/,（2011.7.27）