Journal of Web Engineering, Vol. 4, No. 1 (2005) 003–020

A COMPARISON OF TWO APPROACHES FOR AUTOMATIC CONSTRUCTION OF WEB APPLICATIONS

MITSUHISA TAGUCHI, KORNKAMOL JAMROENDARARASAME
KAZUHIRO ASAMI, and TAKEHIRO TOKUDA
Department of Computer Science, Tokyo Institute of Technology
Meguro, Tokyo 152-8552, Japan
{*mtaguchi, konkamol, asami, tokuda*}*@tt.cs.titech.ac.jp*

Received September 29, 2004
Revised November 8, 2004

To support development of consistent and secure Web applications, we have designed a number of Web application generators. These generators can be roughly classified into two types of approaches: an annotation approach and a diagram approach. In this paper, we try to make clear the roles of these generators, and compare the two approaches in terms of target applications, development processes and target users. While both approaches are sufficiently powerful and flexible enough to efficiently construct typical Web applications, the most appropriate generator should be chosen according to the characteristics of the application and the development process.

Keywords: Web applications, code generation, annotations, diagrams

1 Introduction

Today, Web applications such as database query systems and transaction systems are widely used especially on the Internet. The development of such applications, however, is costly and requires considerable experience on the part of developers because of the complexity of implementing features such as security checks and session management. To support development of consistent and secure Web applications, we have designed a number of Web application generators [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. Web application generators generate executable code necessary to execute a Web application. They encapsulate the complexity unique to Web applications and allow developers to concentrate on the business logic of the application.

These generators can be roughly classified into two types of approaches. The first is an annotation approach, which concentrates on input data and embedded values on each Web page [1, 2]. Developers first compose Web page templates and associate annotations with them. From the annotated Web page templates, the generator generates an implementation. The second is a diagram approach, which concentrates on dataflow relationships in the application [3, 4, 5, 6, 7, 8, 9, 10, 11]. Developers first compose diagrams that describe overall dataflow relationships among Web components such as Web page templates, server side programs and databases. After developers select appropriate program templates and components, a generator then can generate an implementation, together with prototypes of Web page templates, from the diagrams.

Each approach makes certain assumptions and identifies specific roles in development processes. Thus the most appropriate generator should be chosen according to the characteristics of the application and the development process. In this paper, we try to make clear the roles of these generators, and compare the two approaches in terms of target applications, development processes and target users.

The organization of the rest of this paper is as follows. In section 2 and section 3 respectively, we describe an annotation approach and a diagram approach, and give examples of our generator systems based on each approach. In section 4, we compare the two approaches in terms of target applications, development processes and target users. Finally, we discuss future work and provide concluding remarks in section 5.

2 An Annotation Approach

2.1 Basic Idea of Annotation Approach

From the viewpoint of user interfaces, we can consider general Web applications as transitions between Web pages just like ordinary Web sites that have no server side programs. In most Web applications, the greater part of each Web page template is static, and the other parts are dynamic where actual values are embedded by server side programs. Based on this idea, we present an annotation approach to automatic construction of Web applications. In this approach, we define the behavior of Web applications by associating annotations with dynamic parts of Web page templates. More precisely, the following steps are generally taken in the generation method.

1. We compose Web page templates for intended Web applications. We can use Web page composers and authoring tools to visually compose Web page templates. Dynamic parts of the templates, where actual values are embedded at run time, are represented by special characters to distinguish the parts from static parts.

2. We associate annotations with dynamic parts of Web page templates. Annotations are declaration of data processing that is done on the server side. The tasks are mainly related to dataflow relationships among Web components as follows.

   - Input data checking such as checking the types and the length of input values, and constraints among them.
   - Session management such as checking the beginning and the end of the session.
   - Handling of data storage such as the access to database management systems.
   - Communications with external programs such as invocation of Web services.

3. From Web page templates with annotations, a Web application generator automatically generates source code of server side programs. The generator also generates additional code for session management and the standard level of security by analyzing transitions and parameters between Web page templates.

An example of a related but substantially different annotation approach is Zolar [12]. Zolar concentrates on manipulations of relational databases. We can use declarative annotations for raising SQL queries and showing the results, which are represented by extended tags. It

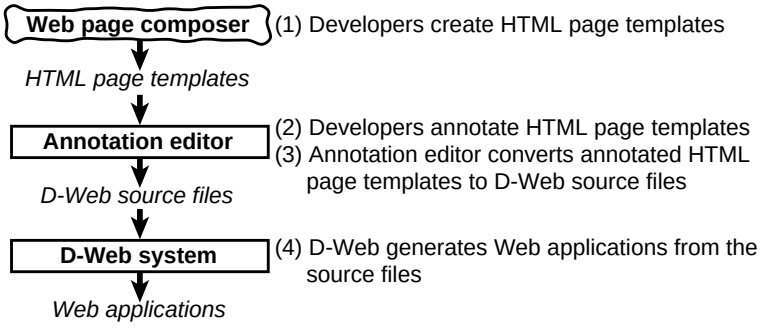


Fig. 1. The architecture of the A-Web system

encapsulates the complexity of database handling. Other examples of a related but substantially different annotation approach are server side scripts such as ASP [13] and JSP [14]. In server side scripts, we can use procedural annotations represented by fragments of programs whose source code is written in procedural programming languages such as Java.

2.2 A Generator: A-Web System

Based on the annotation approach, we designed and implemented a prototype generator called *A-Web system* [1, 2]. While the current prototype can generate only CGI-based applications, we may be able to generate Web applications based on other architectures by replacing part of the generator, because the annotation approach itself is independent of the specific architecture.

The architecture of A-Web system is shown in Fig. 1. A-Web system consists of two parts: *an annotation editor* and a Web application generator called *D-Web system*.

Web page templates

Web page templates should be prepared as input of A-Web system. We can use general visual composers and authoring tools to compose Web page templates because A-Web system requires ordinary XHTML documents. Dynamic parts of the templates should be represented by special characters ${*scope.valiable*}. *Variable* is a name of the variable to distinguish it from other variables in the application and should be unique in each *scope*. Because the special characters are ordinary strings, not extension of HTML tags, general tools can deal with these templates.

Fig. 2 shows an example of Web page templates in a simple member registration system. This application first requires a user to input an id and a password that the user chooses, a name, an e-mail address, a telephone number and an address on a Web page 'Registration'. If the id is already registered or the input data don't satisfy given conditions, a page 'Error' is generated. Otherwise, a page 'Confirmation' is generated. After the confirmation, the registration becomes definite.

An annotation editor

An annotation editor is a part of A-Web system, which is a special editor to annotate Web page templates and convert them to D-Web source documents. The editor analyzes Web page

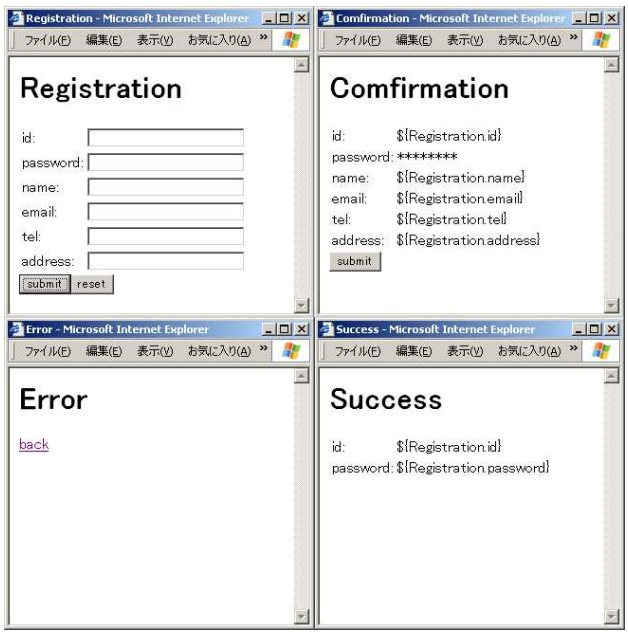


Fig. 2. An example of Web page templates

templates, points out where we can associate annotations and allows us to visually annotate the templates. In the current prototype, we introduce five types of annotations: input check, constraint, session, SQL and SOAP.

- **An input check annotation** defines conditions for accepting users' input. If the type of the input is a string, we can define the pattern and the length of acceptable characters. In the case of a number, we can define a range of acceptable numbers. If necessary, we can define new types such as a type 'E-mail' using regular expression and add them into the editor. This annotation can be associated with each field of an input form.

- **A constraint annotation** defines relations that must be satisfied among input data. The relations are described as logical formulas. This annotation can be associated with each input form.

- **A session annotation** defines the behavior of the session for each Web page template. When a user accesses a Web page with a session annotation '*begin*', the server side program starts a new session. In the case of '*check*', the program checks whether the session is valid or not. In the case of '*end*', the program terminates the session. This annotation can be associated with each Web page template.

- **A SQL annotation** describes SQL statements to access database management systems. We can deal with database transactions when more than one statement is used. This annotation can be associated with each Web page template and each table to show the results of the queries.

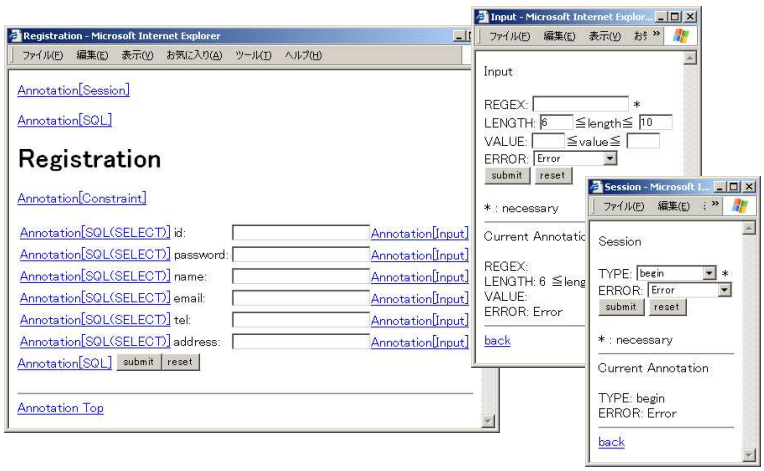


Fig. 3. The annotation editor of A-Web system

- **A SOAP annotation** describes statements to invoke external Web services using SOAP protocol. This annotation can be associated with each Web page template.

Fig. 3 shows the annotation editor of A-Web system. It analyzes Web page templates and automatically adds hyperlinks to special annotation pages. Fig. 3 shows a Web page 'Registration', a session annotation page and an input annotation page.

From annotated Web page templates, the annotation editor generates D-Web source code, which has a set of extended tags as follows.

- <**input**> has extended attributes, which correspond to an input check annotation.
- <**constraint**> is a child element of <form>, which corresponds to a constraint annotation.
- <**session**> is a child element of <head>, which corresponds to a session annotation.
- <**sql**> is a child element of <head>, which corresponds to a SQL annotation.
- <**service**> is a child element of <head>, which corresponds to a SOAP annotation.

Fig. 4 shows an example of D-Web source code, which is generated from a Web page template 'Registration' and its annotations.

D-Web system

D-Web system is a part of A-Web system, which gets D-Web source documents as input and generates source code of Web applications. To keep the consistency of generated applications, D-Web system analyzes all variables in all Web page templates and transitions between the templates, and then automatically generates code to correctly pass the values to Web pages where they are used. At the beginning of the analysis, D-Web system constructs diagrams called *template transition diagrams*, which describe the transitions between Web page templates. Fig. 5 shows an example of template transition diagrams. The system finds that

```
<html>
<head>
  <title>Registration</title>
  <session type="begin" error="Error.html"/>
</head>
<body>
  <h1>Registration</h1>
  <form action="Confirmation.html" method="Post">
    id:<input type="text" name="id" length="[4,10]"/>
    password:<input type="text" name="password" length="[6,10]"/>
    name:<input type="text" name="name"/>
    email:<input type="text" name="email" regex="^[^@]+@[^.]+\..+$"/>
    tel:<input type="text" name="tel" regex="^[0-9]+$"/>
    address:<input type="text" name="address"/>
    <input type="submit" value="submit"/>
    <input type="reset" value="reset"/>
  </form>
</body>
</html>
```

Fig. 4. An example of D-Web source code

the input values of 'id' and 'password' should be passed to a page 'Confirmation' and a page 'Success'.

All generated programs take steps at run time as follows.

1. After receiving input data, the program checks whether the user comes from a correct Web page according to the template transition diagrams.
2. The program checks whether user's session id is valid according to the session annotation.
3. The program checks whether input data are correct according to the input check annotations and the constraint annotation.
4. The program executes the business logic according to the SQL annotations and the SOAP annotation.
5. The program finally generates a dynamic Web page.

If at least one error occurs during the above process, the user's request is redirected to an error page.

3 A Diagram Approach

3.1 Basic Idea of a Diagram Approach

From the viewpoint of dataflow relationships among Web components such as Web pages and programs, we can consider Web applications as applications based on pipes and filters architecture. A filter corresponds to a program and a pipe corresponds to a Web page. Based on this idea, we present a diagram approach to automatic construction of Web applications. In this approach, we first compose diagrams describing overall behavior of the application,

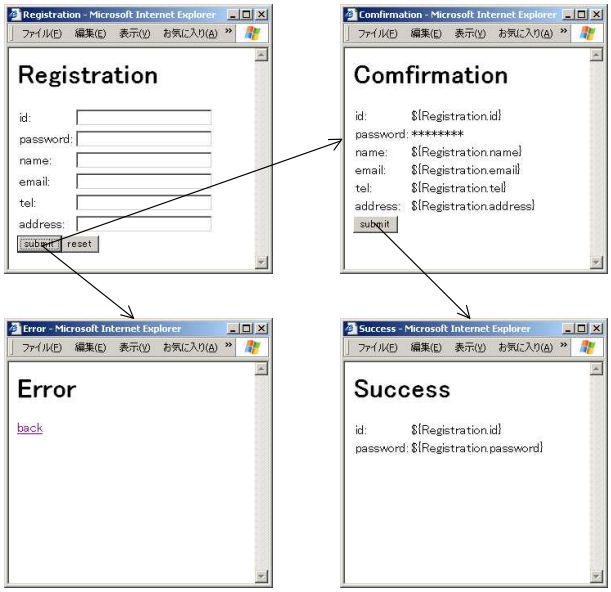


Fig. 5. An example of template transition diagrams

and then select appropriate program templates and components to generate executable applications. More precisely, the following steps are generally taken in the generation method.

1. We compose directed graphs whose nodes represent Web components such as Web page templates, server side programs and databases, and whose edges represent dataflow relationships among the components.

2. A Web application generator has predefined program templates and components that are independent of specific application domains, for example, for purposes of database manipulations and sending electronic mail. Referring specifications of these programs, we define correspondence between nodes in diagrams and predefined programs, and then specify parameters of the programs.

3. From the diagrams and a set of values of parameters, a generator automatically generates an implementation, together with prototypes of Web page templates. The generator also generates additional code for session management and the standard level of security by analyzing the diagrams.

An example of a related but quite different diagram approach is WebRatio [15]. WebRatio uses WebML as a modeling language of Web applications [16]. We first compose a structural model, which expresses data contents in the Web site. Using the structural model and predefined content units, we compose site views, which express the structure of Web page templates and the navigation. We can generate executable Web applications from these diagrams.

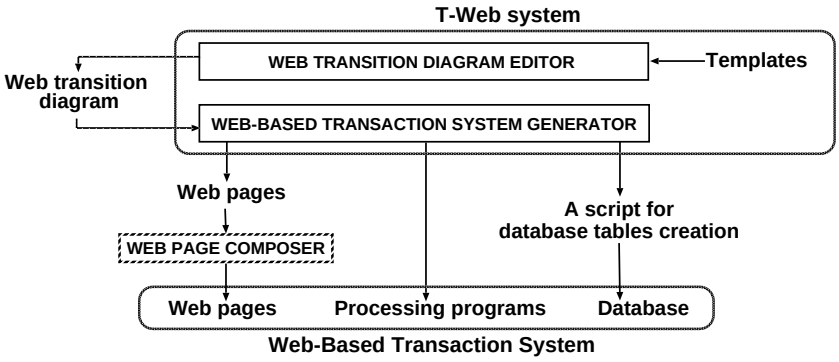


Fig. 6. The architecture of the T-Web system

3.2 A Generator: T-Web System

Based on the diagram approach, we designed and implemented a prototype generator called *T-Web system*, which generates Web applications from directed graphs called *Web transition diagrams* [3, 4, 5, 6, 7, 8, 9, 10]. While we have prototypes to generate CGI-based applications, JSP/Servlet-based applications and ASP-based applications respectively, we may be able to generate applications based on other architectures by replacing part of the generator, because the diagram approach itself is independent of the specific architecture of Web applications.

In this section, we explain the basic architecture of T-Web system using the generator for JSP/Servlet-based applications. The architecture of T-Web system is shown in Fig. 6. T-Web system consists of two parts: *a Web transition diagram editor* and *a Web application generator*.

Web transition diagrams

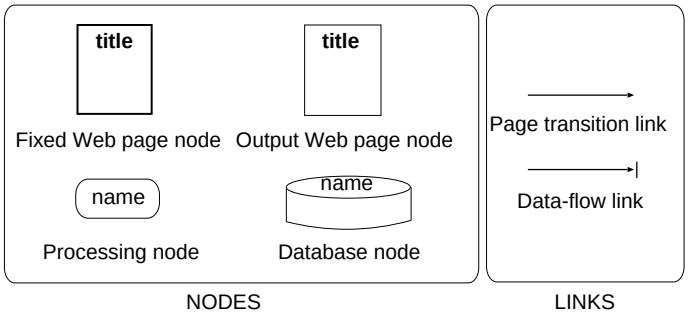
We first describe overall behavior of the intended application using directed graphs called Web transition diagrams. Basically, Web transition diagrams consist of four types of nodes and two types of links as follows.

- **A fixed Web page node** is a static Web page, which is accessible by a certain URL. Its notation is a rectangle with its name, whose line is thick. It may have a number of page elements such as hyperlinks and input fields inside the rectangle.

- **An output Web page node** is a dynamic Web page, which is generated by a server side program. Its notation is a rectangle with its name, whose line is thin. Like a fixed Web page node, it may have a number of page elements.

- **A processing node** is a server side program, which is activated by users' requests. Its notation is an oval with its name.

- **A database node** is a relational database table in a database server. Its notation is a cylinder with its name. The schema of the table is represented by a list of column names $\{col_1, col_2, ..., col_n\}$.



Element type	Representing symbol
a hyper link	name
a text input (text)	name [ ]
a text input (password)	name [*******]
a textarea	name [ ]
checkboxes	name ☐ choice1 ☐ choice2
radio buttons	name ◉ choice1 ○ choice2
a button	[Submit]
a visible parameter	<id>
a hidden parameter	(id)
database information	<<columns>>

Fig. 7. The notation of Web transition diagrams

- **A page transition link** is a hyperlink relationship between Web pages. Its notation is a directed line.

- **A dataflow link** is a dataflow relationship among Web components such as Web pages, programs and database tables. Its notation is a directed line with a blocked line. The link may have a data set to be passed represented by a list of variable names $<var_1, var_2, ..., var_n>$, and a condition of the transition represented by [*condition*].

The notation of nodes, links and page elements is shown in Fig. 7. Each generator may have extension of Web transition diagrams according to the target architecture. The extended diagrams have other kinds of data storage nodes such as temporal server memory, client's cookies, mail servers and Web services. Fig. 8 shows an example of Web transition diagrams describing a simple member registration system, which is the same application as given in section 2.

A Web transition diagram editor

A Web transition diagram editor is a part of T-Web system, which is a special editor to support composition of consistent Web transition diagrams. The editor allows us to do all operations visually. The following steps are generally taken to compose Web transition diagrams.

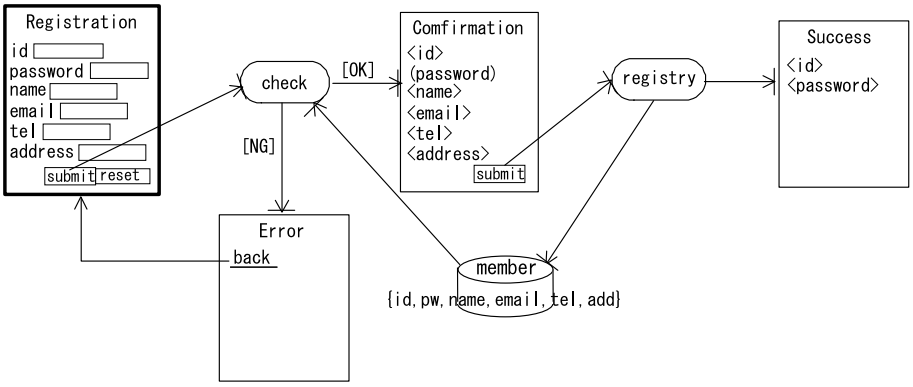


Fig. 8. An example of Web transition diagrams

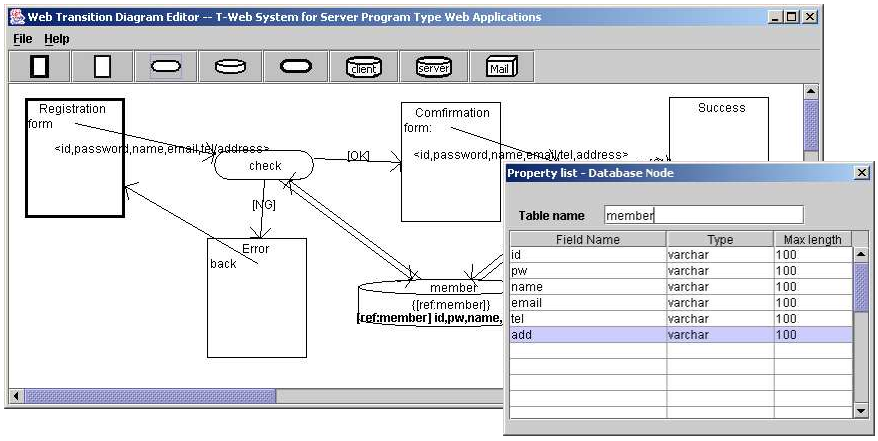


Fig. 9. The Web transition diagram editor of T-Web system

1. We draw Web transition diagrams by selecting a node from a list, placing it on the drawing field and arranging nodes.

2. We specify details of each node using a special window called *a property window*. On the property window for processing nodes, we select the most appropriate program template from a list. To keep the consistency of Web transition diagrams, a number of parameters can be also specified by selecting an appropriate value from a list.

Fig. 9 shows the Web transition diagram editor of T-Web system.

A Web application generator

A Web application generator is a part of T-Web system, which gets descriptions of Web transition diagrams and a set of values of parameters as input, and generates source code of server side programs and prototypes of Web page templates. The prototype system shown in Fig. 6 generates JSP documents written in HTML and servlet source code written in Java.

```
/*package PACKAGE;*/
import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class /*CLASSNAME*/ extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                                throws ServletException, IOException {
    String myHttpsURL = "/*HTTPSBASEURI*/"+"/*PROGRAMURI*/"
                      +"/*PACKAGE.*/"+"/*CLASSNAME*/";
    String dbName = "/*DBNAME*/";
    String tbName = "/*TABLE*/";
    String[] dbColNames = {/*"FIELDNAME"*/};
    /**String PARAMETER = "";**/
    Connection con = TWeb.doConnect(dbName);
    String[] concolNames = {/*"EXISTPARAM"*/};
    String[] concolValues = {/*"'"+INPUTPARAM+"'"*/};

    if (!checkExist(con,tbName,concolNames,concolValues)) {
      gotoPage("/*/JSPDIRECTORY/*//*NGPAGE.jsp*/",request,response);
    } else {
        --- omission ----
      gotoPage("/*/JSPDIRECTORY/*//*OKPAGE.jsp*/",request,response);
    }
    con.close();
  }
}
```

Fig. 10. An example of program templates

The generator has general-purpose program templates that are independent of specific application domains. The current prototype has 16 program templates for database manipulations and sending electronic mail. Fig. 10 shows an example of program templates in T-Web system. The program templates have parameters represented by words between special characters '/*' and '*/'. Special characters '/**' and '**/' mean a repetition part. As the generator rewrites the parameters to the corresponding values, the program template becomes a complete server side program. In the case of the application shown in Fig. 8, we should apply a template that looks into a database table to a processing node 'check', and a template that inserts input values into a database table to a processing node 'registry'.

Each generator generates source code of server side programs and prototypes of Web page templates according to its generation rule. Fig. 11 shows the generation rules of our prototype generators. To generate applications based on server program type architectures such as CGI, fixed Web page nodes correspond to static Web page templates, and other nodes correspond to server programs. In the case of server page type architectures such as ASP, a part of output Web page nodes, processing nodes and database nodes correspond to dynamic Web page templates with the business logic. In the case of combination of the above two

<table>
<tr><th></th><th>Server program type</th><th>Server page type</th><th>Combination type</th></tr>
<tr><td>Fixed Web page</td><td>Static pages</td><td>Static pages</td><td>Static pages</td></tr>
<tr><td>Output Web page</td><td rowspan="3">Server programs</td><td>Dynamic pages</td><td rowspan="2">Dynamic pages</td></tr>
<tr><td>Processing</td><td rowspan="2">Dynamic pages
with the business logic</td></tr>
<tr><td>Database</td><td rowspan="2">Server programs</td></tr>
</table>

Fig. 11. Examples of generation rules

architectures such as JSP/Servlet, output Web page nodes correspond to dynamic Web page templates, and processing nodes and database nodes correspond to server programs.

All generated programs take steps at run time as follows.

1. After receiving input data, the program checks whether user's session id is valid.
2. The program checks whether input data are correct according to the specifications of Web page nodes in diagrams.
3. The program executes the business logic according to the specifications of the processing node and checks whether output data are correct according to the specifications of database nodes or other storage nodes in diagrams.
4. The program finally generates a dynamic Web page.

If at least one error occurs during the above process, the user's request is redirected to an error page.

4 Comparison of Two Approaches

We compare the annotation approach and the diagram approach in terms of target applications, development processes and target users. We discuss target applications by dividing into three viewpoints: application domains, flexibility and scalability. In addition to "development" processes, we also discuss "maintenance" of generated applications. We discuss them according to the following criteria.

1. **Backgrounds** show why the viewpoint is important to select an appropriate generator.
2. **Characteristics** show the advantage and the disadvantage of the two approaches and compare them.
3. **Examples** show our practical experience and other points to notice.

4.1 Application Domains

Backgrounds Web applications can be generally classified into two types according to their functions: data-intensive Web applications and control-intensive Web applications [17]. Data-intensive Web applications have side effect computations such as operations to update databases, while control-intensive Web applications concentrate on execution of the complex business logic. The above difference of the target applications has a great influence on developers to select an appropriate generator.

Characteristics The annotation approach is more appropriate to data-intensive Web applications than control-intensive ones, because we describe the control-flows using annotations, which are dispersed among Web page templates. In the diagram approach, it depends on the abstraction level of the diagrams. While detailed diagrams are applicable to both data-intensive and control-intensive applications, highly abstracted diagrams should be specialized for either data-intensive or control-intensive ones. Of course, highly abstracted diagrams are more helpful to encapsulate the complexity of implementing features of Web applications.

In general, the annotation approach is easy to deal with Web page templates whose page layouts are flexible, because we can use general Web page composers and authoring tools to compose Web page templates before the generation. The diagram approach may not be good at dealing with such templates, especially having clickable maps and multiple frames, because it is hard work to associate data-flows with the above components. On the other hand, the diagram approach doesn't care the ratio of static parts and dynamic parts of Web page templates, while the annotation approach is not good at dealing with templates whose ratio of dynamic parts is high.

Examples Both A-Web system and T-Web system concentrate on data-intensive Web applications. The generators aim to encapsulate the complexity of implementing features such as security checks and session management, which are greater problems in data-centric Web applications than in control-centric ones. Most of our generators can deal with applications having functions as follows: database manipulations, invocation of external programs and sending/receiving electronic mail. We are successful in developing typical data-intensive Web applications using A-Web system and T-Web system respectively: shopping cart systems, guestbook systems, glossary systems, schedule organizing systems, member registration systems and reservation systems. On the other hand, our generators are not good at dealing with complex page transitions that depend on the results of the processing.

4.2 Flexibility

Backgrounds When we need new business logic to generate an intended Web application, it is a big problem how to add new program templates and components into the generator. It is also important what types of architectures the generator can deal with, because we often have constraints on the execution environment of the application.

Characteristics In the annotation approach, we can use new programs by setting new programs and adding new annotations to invoke them into the generator. In the diagram approach, we can use new program templates and components by adding new programs, together with documents that describe programs' specifications, into the generator. While both annotations and predefined programs are enough to encapsulate invocation of external programs, the diagram approach seems to be more flexible than the annotation approach, because we can select appropriate program templates and components in the later period of the generation process. When we want to generate Web applications based on other architectures, we can also replace part of the generator in both approaches.

Examples Using A-Web system and T-Web system respectively, typical data-intensive Web applications can be generated without new annotations and programs. When we construct complex Web sites such as shopping sites, however, we possibly need new business logic. In that case, it requires programming skills to implement new program templates and components. In particular, it is costly to test the implementation because the programs should be general-purpose ones and they have a number of parameters. We have implemented generators and confirmed that our approaches can deal with applications based on CGI, ASP and JSP/servlet architectures. When we extend a generator so that it can deal with new architecture, it may require high programming skills.

4.3 Scalability

Backgrounds The scale of an intended application is also an important problem as well as the complexity of the business logic.

Characteristics In the annotation approach, we can basically concentrate on transitions between two Web pages because a generator automatically generates code to correctly pass values to the pages where the values are used. Thus, even if the scale of an intended Web application becomes larger, the generation task doesn't become so complicated. In diagram approach, however, the larger the scale of an intended Web application becomes, the more complicated the management of diagrams becomes. While we can make diagrams nested, it is still a big problem how to manage the specifications of components that are used in more than one diagram. To solve the problem, development tools have to fully support composition of large scale diagrams.

Examples Generators based on diagram approach can easily deal with no more than 15 Web page templates for each session. Although 15 Web page templates are enough to construct typical Web applications, a complex shopping site may require more than 15 templates. In that case, site diagrams are convenient to look over the structure of the Web site and static relationships between Web page templates.

4.4 Development Processes

Backgrounds Web applications are characterized by three major design dimensions: *structure*, *navigation* and *presentation* [17]. We discuss effective development processes in each approach by taking notice of the order of these three design activities.

Characteristics In the annotation approach, we generally take the following development process.

1. We first define requirement specifications and analyze them.
2. We decide what Web pages are needed in the application and design data structure (as a structure model) that the application deals with. From the above chosen pages and the structure model, we design Web page templates (as a presentation model) and implement them. We also design dataflow relationships among the application (as a navigation model) based on the above models.
3. From the implementation of Web page templates and the navigation model, a generator generates server side programs.

In the diagram approach, we generally take the following development process.

1. We first define requirement specifications and analyze them.
2. We decide what Web pages are needed, design data structure (as a structure model) and design dataflow relationships among the application (as a navigation model). From the above models, we compose diagrams describing overall behavior of the application.
3. From the diagrams, a generator generates server side programs and prototypes of Web page templates. After that, we design Web pages (as a presentation model) and revise generated templates.

If we have to revise the appearance of Web pages after the generation, we should use XSLT and CSS to compose Web page templates in both approaches.

Examples The annotation approach has the advantage of composing prototypes of Web page templates early in the development process and reusing them for the final product. We can take an iterative and incremental process to compose Web page templates. On the other hand, it is hard to implement the navigation before the composition of Web page templates. The diagram approach has the advantage of designing and implementing the structure and the navigation iteratively and incrementally. On the other hand, it is hard to implement Web page templates before the generation.

4.5 Maintenance

Backgrounds Most development tools tend to focus on the development of the first version of the application. However, it is also an important problem how to update the implementation.

Characteristics When we modify the data structure in the annotation approach, we edit related annotations and re-generate server side programs. In the diagram approach, we edit the diagrams, specify related parameters, and re-generate server side programs and Web page templates. The annotation approach is advantageous to the small modification, because we can concentrate on Web page templates that should be revised. On the other hand, the diagram approach is advantageous to the extensive modification that ranges over a good number of Web page templates.

When we modify the navigation or the business logic in the annotation approach, we revise hyperlink relationships between Web page templates, edit related annotations and re-generate server side programs. In the diagram approach, we edit the diagrams and related parameters, and re-generate server side programs and Web page templates. The diagram approach is advantageous to the modification of the navigation, because diagrams are helpful to understand the whole navigation of the application.

When we modify the presentation in the annotation approach, we edit annotations related to the appearance of Web pages and re-generate server side programs. In the diagram approach, we specify parameters related to the appearance of Web pages and re-generate server side programs and Web page templates. The annotation approach is advantageous to the modification of the presentation, because we can concentrate on

Web page templates to be revised. When we update only the Web page layouts, we have only to revise stylesheets such as XSL and CSS files in both approaches.

Examples Although A-Web system and T-Web system check the consistency of revised Web page templates and diagrams respectively before the generation, they don't fully support the maintenance yet. According to our small experiment, it becomes easier to modify the navigation in diagram approach by using XForm and XLink .

4.6 Target Users

Backgrounds The minimum knowledge a generator requires is also a great problem to use it.

Characteristics Of course both approaches don't require programming skills to use generators. In the annotation approach, we possibly need the basic knowledge of markup languages to edit the documents of Web page templates. In addition, generators require the knowledge of data types because a number of annotations are related to data types. On the other hand, we don't have to know the architecture of Web applications because the behavior of the application is defined by dataflow relationships between Web pages. The diagram approach doesn't require the knowledge of markup languages to use generators. When we re-generate server side programs after the revision of Web page templates, however, we have to take notice of components that are related to the appearance of Web pages. On the other hand, we have to know the basic architecture of Web applications and the concept of session management because we first compose diagrams describing overall behavior of the application. In addition, generators require a skill in selecting and using general-purpose programs.

The annotation approach is particularly effective for non-programmers who have the basic knowledge of data types. The diagram approach is particularly effective for inexperienced developers who have the basic knowledge of Web applications.

Examples A-Web system doesn't require the knowledge of HTML at all, because the annotation editor points out where and what types of annotations we can use and allows us to visually annotate Web page templates. We may use regular expression and SOAP protocol to construct advanced Web applications. T-Web system requires the basic knowledge of static/dynamic Web pages and data storage is necessary. Although we have to select appropriate program templates and components from pre-defined ones, it doesn't take so much time before we get used to them.

5 Conclusion and Future Work

To support development of consistent and secure Web applications, we have designed and implemented a number of Web application generators. In this paper, we classified these generators into two types of approaches, made clear the roles of the generators and compared the two approaches in terms of target applications, development processes and target users. The annotation approach is particularly effective for non-programmers who know data types to construct Web sites whose Web page layouts are flexible. The diagram approach is particularly effective for inexperienced developers who know the basic architecture of Web applications

	An annotation approach	**A diagram approach**
Application domains	Appropriate to data-intensive Web applications. Easy to deal with flexible Web page layouts.	Appropriate to both data-intensive/control-intensive Web applications. Difficult to deal with flexible Web page layouts.
Flexibility	We can set new programs and add new annotations into the generator.	We can add new programs and documents of their specifications into the generator.
Scalability	Applicable to large scale applications.	Disadvantageous to large scale applications.
Development Processes	Appropriate to iterative and incremental composition of Web page templates. Disadvantageous to rapid prototyping of the whole application.	Appropriate to rapid development of executable applications. Disadvantageous to early composition of Web page templates.
Maintenance	Advantageous to the small modification of the data structure, and the modification of the presentation	Advantageous to the extensive modification of the data structure, and the modification of the navigation.
Target Users	We need the knowledge of markup languages and data types. We don't need the knowledge of the Web architecture.	We need the knowledge of the Web architecture and the concept of session management. We don't need the knowledge of markup languages.

Fig. 12. A summary of the two approaches

to rapidly develop executable applications. Using our prototype systems, we confirmed that both approaches are sufficiently powerful and flexible to construct typical data-intensive Web applications.

Most popular development methods and generator systems such as UML-based methods [18, 19] tend to be highly flexible and deal with any kinds of complex applications. On the other hand, we concentrate on the easiness and the rapidness so that both non-programmers and programmers can efficiently construct typical Web applications. As our future work, we may try another approach based on the combination of the two approaches. This approach may allow us to implement programs and Web page templates concurrently, and thus make the development process more flexible and efficient.

References

1. K. Asami and T. Tokuda. Generation of Web Applications from HTML Page Templates with Annotations. *Proceedings of the IASTED International Conference APPLIED INFORMATICS*, pp.295-300, 2002.
2. K. Asami and T. Tokuda. Generation of Web Applications from Annotation-Based Definitions. *Proc. of Engineering Information Systems in the Internet Context*, pp.69-79, 2002.

3. T. Matsuzaki, T. Suzuki and T. Tokuda. A Pipe/Filter Architecture Based Software Generator PF-Web for Constructing Web Applications. *Computer Software of Japan Society for Software Science and Technology Vol.19 No.4*, pp.266-282, 2002.
4. K. Jamroendararasame, T. Suzuki and T. Tokuda. A Generator of Web-based Transaction Systems Using Web Transition Diagrams. *Proc. 17th Japan Society for Software Science and Technology*, 2000.
5. K. Jamroendararasame, T. Matsuzaki, T. Suzuki and T. Tokuda. Generation of Secure Web Applications from Web Transition Diagrams. *Proc. of the IASTED International Symposia Applied Informatics*, pp.496-501, 2001.
6. K. Jamroendararasame, T. Matsuzaki, T. Suzuki and T. Tokuda. Two Generators of Secure Web-Based Transacion Systems. *Proc. of the 11th European-Japanese Conference on Information Modelling and Knowledge Bases*, pp.348-362, 2001.
7. K. Jamroendararasame, T. Suzuki and T. Tokuda. JSP/Servlet-Based Web Application Generator. *18th Conference Proceedings Japan Society for Software Science and Technology*, 2C–1, 2001.
8. K. Jamroendararasame, T. Suzuki and T. Tokuda. A Visual Approach to Development of Web Services Providers/Requestors. *Proc. of the 2003 IEEE Symposium on Visual and Multimedia Software Engineering*, pp.251-253, 2003.
9. M. Taguchi, T. Susuki and T. Tokuda. Generation of Server Page Type Web Applications from Diagrams. *Proc. of the 12th Conference on Information Modelling and Knowledge Bases*, pp.117-130, 2002.
10. M. Taguchi, T. Suzuki and T. Tokuda. A Visual Approach for Generating Server Page Type Web Applications Based on Template Method. *Proc. of the 2003 IEEE Symposium on Visual and Multimedia Software Engineering*, pp.248-250, 2003.
11. T. Tokuda, T. Suzuki, K. Jamroendararasame and S.Hayakawa. A family of Web diagrams approach to the design, construction and evaluation of Web applications. *Proc. 12th European-Japanese Conference on Information Modelling and Knowledge Bases*, pp.293-306, 2002.
12. International System Research Inc. *Zolar: Connection tool for WWW server and database.* http://www.isr.co.jp/products/zolar/
13. A. Homer, D. Sussman, B. Francis et al. *Professional Active Server Pages 3.0.* Wrox Press, 1999.
14. M. Hall. *Core Servlets and JavaServer Pages.* Prentice Hall PTR, 2000.
15. Web Models Inc. *WebRatio Site Development Studio.* http://www.webratio.com/
16. S. Ceri, P. Fraternali and A. Bongio. Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. *Proc. of the 9th World Wide Web Conference*, 2000.
17. Piero Fraternali. Tools and Approaches for Developing Data-Intensive Web Applications: A Survey. *ACM Computing Surveys Vol.31 No.3*, pp.227-263, 1999.
18. J. Conallen. Modeling Web Application Architectures with UML. *Communications of the ACM Vol.42 No.10*, pp.63-70, 1999.
19. R. Hennicker and N. Koch. A UML-based Methodology for Hypermedia Design. *Proc. of UML 2000 Conference*, pp.410-424, 2000.