

Decomposition and Abstraction of Web Applications for Web Service Extraction and Composition

Michiaki Tatsubori
IBM Tokyo Research Laboratory
mich@acm.org

Kenichi Takahashi
Tokyo Denki University
kenichi@se.sie.dendai.ac.jp

Abstract

There are large demands for re-engineering human-oriented Web application systems for use as machine-oriented Web application systems, which are called Web Services. This paper describes a framework named H2W, which can be used for constructing Web Service wrappers from existing, multi-paged Web applications. H2W's contribution is mainly for service extraction, rather than for the widely studied problem of data extraction. For the framework, we propose a page-transition-based decomposition model and a page access abstraction model with context propagation. With the proposed decomposition and abstraction, developers can flexibly compose a Web Service wrapper of their intent by describing a simple workflow program incorporating the advantages of previous work on Web data extraction. We show three successful wrapper application examples with H2W for real world Web applications.

1. Introduction

Developing Web Services (machine-oriented Web-based services) from scratch is often a costly task, especially when the published service is mostly processed automatically. This is partly because of costly requirements for sufficient consistency and security for the service. Such requirements are usually appropriate for many kinds of organizations such as private enterprises or government entities. For example, a shopping service should not accept a request with a negative number for the quantity of items to buy, while a university course management service should not be broken into using invalid parameter values constructed by students or attackers.

In order to satisfy minimal requirements for consistency and security, constructing and publishing Web Services by “wrapping” existing Web applications (human-oriented Web-based services) can be a reasonable solution candidate. A typical wrapping technique is to provide a proxy server that serves the Web Service by accessing the original Web application. With this ap-

proach, the minimal requirements for consistency and security are naturally satisfied because the existing Web applications have already been well tested and are known to run consistently and securely. For example, a shopping Web Service constructed upon an existing shopping Web application can detect errors in a request with an invalid quantity value for the items to be bought, at least at the level of the underlying Web application that was already developed to check for this. Moreover, the wrapper approach may be applied for using external Web applications as Web Services.

However, the existing high-level support for wrapper construction is not suitable for adapting practical Web applications. There are two essential challenges for such software in adapting human-oriented services to machine-oriented services with wrappers. They are:

- extracting the logical data for the machines from data decorated with HTML for human readers, and
- extracting a noninteractive service for machines from interactive services scattered over multiple webpages for humans.

According to Myllimaki [11], extracting structured data from websites requires solving five distinct problems: navigation, data extraction, structure synthesis, data mapping, and data integration. However, in order to clarify the contribution of this paper, we use a rougher categorization. The problems of data extraction, structure synthesis and data mapping are mapped to the former, logical data extraction category in our categorization. The problems of navigation and data integration are mapped to the latter, service extraction category.

A large amount of work has been done addressing the former challenge of extracting logical data for machines. The research topic is often called *Web data extraction* or *Web content extraction*. For example, RoadRunner [6] automates wrapper generation and the data extraction process based on similarities and differences between HTML pages. Extensive survey papers in this area [9, 10] are available. The conclusion of these surveys is that no single extraction approach handles all of the applicable Web data formats. XML-based techniques [11, 8]

have been proposed to allow developers to incorporate the advantages of previous work on Web data extraction.

However, as far as we know, little work has been done on the problem of extracting noninteractive services, even though it must be solved for wrapping practical and complex Web applications. Without support for solving this problem, we would need to abandon the use of high-level tools and, instead, use very low-level HTTP-access toolkits in software libraries like Apache HttpClient [2] to construct the desired wrappers for practical Web applications. Developing wrappers with such libraries is quite error-prone because of the low-level APIs.

In this paper, we propose a framework for constructing Web Service wrappers from existing, multi-paged Web applications. We regard our framework as mainly for *service extraction*, rather than for data extraction, which has been thoroughly discussed in many articles such as ones referred to in the survey papers [9, 10]. Our focus is on providing a practical solution to adapt real world Web applications to meaningful Web Services. The key goals of the proposed framework for this purpose are:

- adaptability in techniques for data extraction from HTML documents, and
- applicability to multi-paged applications with flexibility in providing various services

The extensibility in data extraction means that we do not provide a specific algorithm for Web data extraction but provide a pluggable architecture that can leverage various techniques developed before, or to be developed in the future. The applicability to multi-paged applications means that we provide a software support for handling multiple pages while retaining the flexibility that can be achieved with low-level HTTP-access toolkits for providing any kind of services based upon the original Web application.

For our framework, we first analyzed some practical Web applications and built a model for decomposing and abstracting a Web application into building blocks that can be used to flexibly and simply compose the desired Web Services. We call this decomposition model *page-transition-based decomposition*. In the page-transition-based decomposition, we decompose the entire human-oriented service provided by a Web application on the basis of page transition, in which each transition from page to page is modeled as a modular piece of the entire service. Then we abstract each transition from a page to another as though it consisted of *parameter passing* and *data extraction* with *context propagation*. With this abstraction, when accessing a webpage, the client (browser) passes some parameters with context information to the Web server, and then the server passes back some extracted data with updated context information.

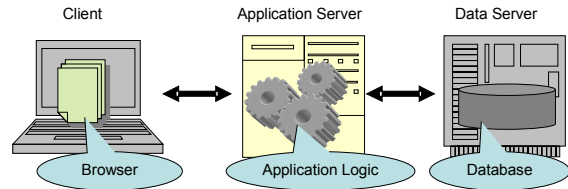


Figure 1: Three-tier model of Web application architecture.

As a proof of concept, we implemented a prototype framework named H2W, and implemented several application examples for real world Web applications. H2W has been developed in an open source manner and its source code is available from:

<http://h2w.sourceforge.jp/en/>

The source code and configuration files of implemented application examples for Web Service wrappers are also available there, and their demonstration services are accessible at a website linked from the introduction page above.

The rest of this paper is structured as follows. First we analyze the semantics of a Web application from the viewpoint of service extraction. Then, in Section 3, we propose a decomposition and abstraction model for practical Web application wrapper construction. We discuss the implementation issues in Section 4 and show concrete application examples of wrapping 3 real world Web applications in Section 5. We discuss the limitations of the proposed approach and compare it to related work in Section 6, and then conclude the paper in Section 7.

2. Web applications

Understanding a Web application's model is the crucial basis for re-engineering it. In this section, we first analyze the semantics of a Web application's model from the viewpoint of service extraction. Then, we discuss the generic navigation design model in Web applications, which we believe is the key for extracting services from multi-paged Web applications. This forms the basis of our notion of service slices. Additionally, we discuss the behavior of Web browsers taken for granted by Web applications, which is important for constructing a Web Service wrapper.

2.1 Semantics of Web application

A Web application is an application delivered to users from a Web server over networks such as the Internet or an intranet. Web applications are popular due to the ubiquity of the Web browser as a client. Many of the Web applications around us are *data-intensive*, where the main purpose of the application is to present a large

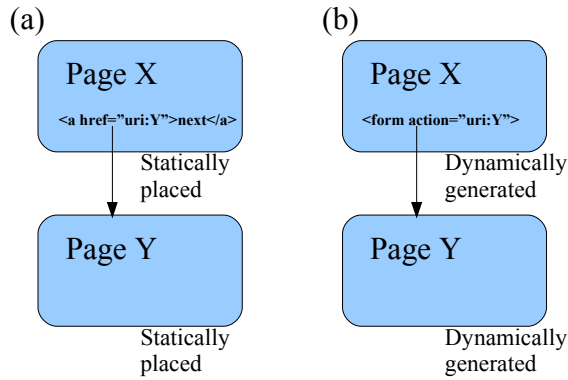


Figure 2: Navigation realization in traditional websites (a) and dynamic Web applications (b).

amount of data to their users [3]. At the same time, such a Web application usually delivers more than simple, decorated data, but also does some services for the users. At least, it does more than a simple Web server or a database.

In this paper, we use the term *services* to denote the additional value provided by a Web application. A Web application is commonly structured as a three-tiered application as shown in Figure 1. In the most common form of the three-tier Web application model, the Web browser is the first tier, an engine using some dynamic Web content technology (such as CGI, PHP, Java Servlets or ASP.net) is the middle tier, and a database is the third tier. The Web browser sends requests to the middle tier, which serves its client by making queries and updates against the database and by generating a user interface. Usually, Web application development requires adaptation of data resources through programming, and programs implementing application logic in the middle tier end up intricately mixing data, navigation, and presentation semantics. Here, what the middle tier does can be regarded as offering services.

In order to extract the services provided by a Web application, we need to understand the characteristics of its navigation design. Here, navigation is one of the three major design dimensions of the characterization of a Web application as described by Fraternali, which are structure, navigation and presentation [7]. Structure describes the organization of the information behind the application logic, in terms of the pieces of content that constitute its information base and their semantic relationships. Navigation concerns the facilities for accessing information and for moving through the application content. Presentation relates to the way in which the application content and navigation commands are presented to the users.

The purpose of the framework proposed in this paper is to support wrapper developers in following the navigation design of a Web application. This is in contrast to

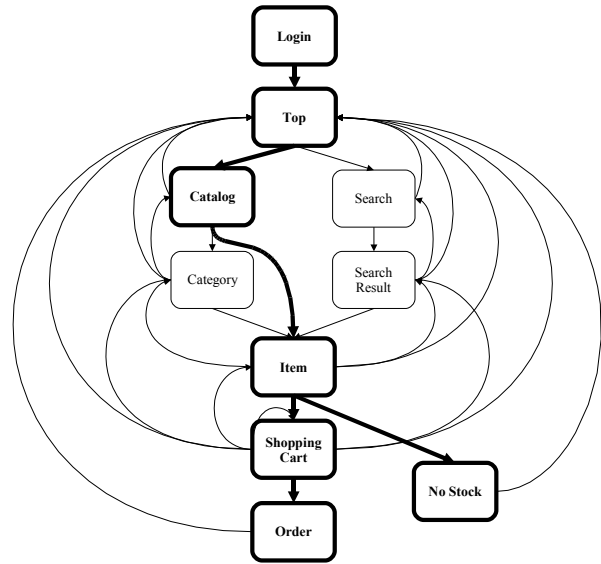


Figure 3: Possible navigation and a service slice in the hypertext design of a Web application.

traditional Web data extraction techniques, which mainly try to remove the presentation and recover the data structures behind it.

2.2 Implementation of navigation

In traditional webpages, navigation is done with the hyperlinks contained in each page using `<a>` tags and pointing to other pages, as shown in Figure 2-(a). There, webpages are statically placed in the website. Even in a Web application, the basic approach for navigation is identical. However, in a Web application, we should generally assume that hyperlinks often be embedded in the action attribute of `<form>` tags and that webpages can be dynamically generated as shown in Figure 2-(b).

In a Web application, the page and its content which users are directed to by following a hyperlink may change dynamically. It generally depends on the state of database, any parameters of the form, and the context of navigation, in addition to the URL specified by the hyperlink. Thus, the target page for navigation from a webpage in a Web application is generally unpredictable.

In reality, the generic type of the target page is often predictable, at least for humans. In order to make a Web application usable, Web application designers are expected to make the results of user actions (the target pages of navigation events) predictable to some extent. This kind of design principle is known as affordance in the lingo of human-computer interaction or perceptual psychology [12], and most parts of a Web application are designed following accord with this principle. Completely unpredictable navigation might be entertaining but is far from good usability.

As a consequence, the dynamically generated hypertext of a Web application can usually be statically modeled as looking like the diagram in Figure 3. This is very similar to the model of a static hypertext website, except that each box in the diagram indicates a certain type of pages but not a specific page. In fact, most data-intensive Web application development tools and approaches like WebML [3] employ this kind of hypertext design model.

2.3 Service slices

Human users can select their desired service from a variety of services provided by a Web application by navigating the hyperlinks. For the example of the Web application depicted in Figure 3, from the top page navigated to after logging in, a user can choose a hyperlink (by clicking a regular link with an `<a>` tag or by submitting a certain form) to a webpage for browsing a catalog of items. The user could also choose a hyperlink to a webpage for searching for items with given keywords. As shown in Figure 3, the page types and navigation links constituting a service are usually limited to only a few routes (depicted with the bold lines) of all the possible page types and navigation links.

We use *service slice* to denote such a set of page types and navigation links. The process of service extraction is to provide a runtime that can extract an instance of a service slice (a set of concrete pages and navigation actions) with the data extraction applied to each page.

2.4 Assumed behavior of browser

To make software to extract services from a Web application, we cannot ignore the behavior of browser, since that is taken for granted in building Web applications. The crucial behaviors are redirection handling, cookies, loading of subordinate pages and images, script execution, and their combinations.

For example, some websites assume that a Web browser reads and save a cookie header associated with an image file referred from the originally accessed webpage. A standard Web browser automatically tries to retrieve images referred to by using `<image>` tags, so it then receives any cookie headers coming with the images. It passes that cookie information to the Web application when the user follows a hyperlink in the original page to receive another page generated dynamically by the Web application. The content of the generated page may depend on the cookie information.

Without appropriately emulating the assumed browser behavior, a wrapper would fail to navigate through the service provided by such a Web application. It is, however, not simple to achieve with low-level HTTP access

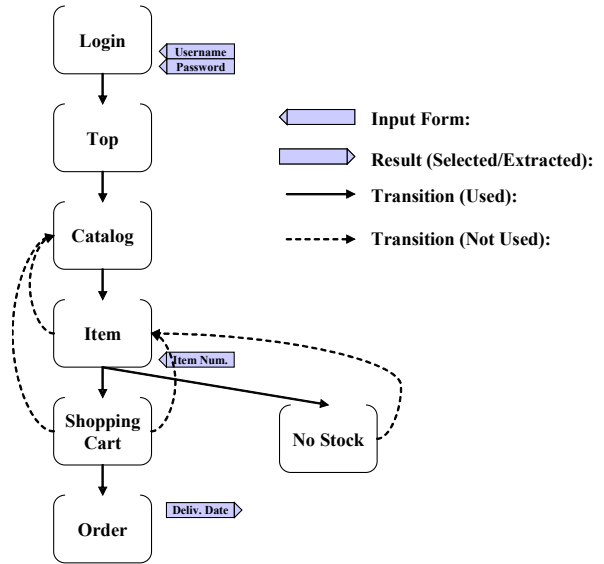


Figure 4: An example service slice of a shopping Web application.

toolkits and should be supported by wrapper construction frameworks.

3. Decomposition and abstraction model

We propose a model for decomposing and abstracting a Web application into modular building blocks that can be used to flexibly compose desired Web Services. We call the proposed decomposition model *page transition-based decomposition*. Based on it, we abstract each transition from a page to another page as it consists of *parameter passing* and *data extraction* with *context propagation*. With these decompositions and abstractions, developers of a Web Service wrapper can describe a simple *workflow* to specify a service slice to extract from the original Web application.

In this section, we first discuss the proposed decomposition model and then describe the proposed abstraction model. Finally, we explain how developers can compose a Web Service capturing their intentions by writing a simple workflow description. Figure 4 depicts a service slice focused in a certain service extraction process. We use this service slice example for the discussion through this section.

In Figure 4, we are focusing on 7 page types: Login, Top, Catalog, Item, Shopping Cart, No Stock, and Order, extracted from Figure 3. Here, the user action scenario corresponding to the service slice expected to be extracted is as follows:

1. In the **Login** page, a user inputs her user name ID (**Username**) and password (**Password**) to be navigated to the **Top** page.

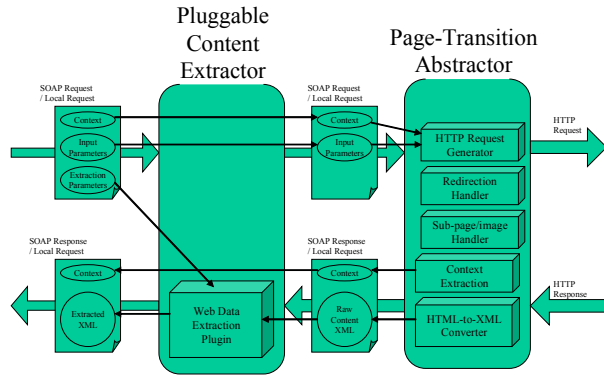


Figure 5: An abstraction process and components for a page-transition.

2. In the **Top** page, she selects to browse a catalog of items to be navigated to the **Catalog** page.
3. In the **Catalog** page, she selects an item and push the button associated with it, to be navigated to the **Item** page.
4. In the **Item** page, she looks at the detail of the selected item, inputs “1” as an amount of the item (**Input Num.**), and push the “buy” button, to be navigated to the **Shopping Cart** page.
5. In the **Shopping Cart** page, she looks at the contents of her shopping cart and push the “confirm” button, to be navigated to the **Order** page.
6. In the **Order** page, she can see the delivery date of the item she bought (**Deliv. Date**).

In addition to the main flow above, there is an alternate flow from the **Item** page. There, she may be navigated to the **No Stock** page to be notified lack of item if so.

3.1 Page-transition-based decomposition

With this decomposition model, we decompose all of the services provided by a Web application on the basis of the page transitions. Each transition from page to page is modeled as a modular piece of the entire set of services. For example, the page transition from the **Login** page to the **Top** page in Figure 4 is modeled as a module, one from the **Top** page to the **Catalog** page as another module, and so on.

In case of the possibility of multiple page types navigated from a hyperlink, we model such navigation as a single module. For example of the page transition from the **Item** page, the resulting page type of pushing the “buy” button may either **Shopping Cart** or **No Stock**. We model these as a single module.

This granularity of decomposition is very natural from the service slice perspective discussed in the previous section. This is because a service slice can be identified by serializing a navigation pattern. In particular, it is fine-grained enough to compose any service slices flexi-

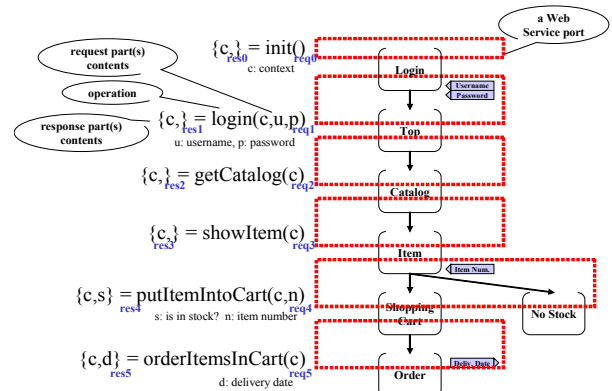


Figure 6: Decomposed service modules used to construct a service operation buyItem.

bly since a page transition or navigation action is an atomic element of a service slice.

3.2 Page-transition abstraction with context propagation

With this abstraction model, we abstract each decomposed module, that is, a transition from a page to another page, as it consists of *parameter passing* and *data extraction* with *context propagation*. When transiting from a webpage to another, a client (browser) passes some parameters with context information to the Web server, and then the server passes some extracted data with updated context information to the client.

This abstraction process is done through two components as shown in Figure 5. The first component (a pluggable content extractor, the right-hand side component in Figure 5) abstracts low-level HTTP accesses for a page transition. The second component (a page-transition abstractor, the left-hand side component) abstracts specific data extraction processes and allows the extraction to be plugged-in.

As a result of the abstraction, a page transition can be regarded as a simple Web Service. This service receives a request consisting of a transition context, input parameters used to fill a form in the source page, and extraction parameters used to control data extraction process (the upper-left message in Figure 5). Then it returns an updated context and extracted data (the lower-left message).

For the purpose of explaining the model proposed in this paper, we use SOAP [13] as the protocol between components. However, actual implementation can use more efficient protocols like local method invocation instead of SOAP. In fact, we have implemented the communication between page-transition abstractor and pluggable content extractor in local method calls in Java.

The role of a page-transition abstractor is:

```

<h2w>
<service name="WineShop">
  <nameSpace>uri:wineshop</nameSpace>
  <location>http://localhost:8080/axis/services</location>
  <uri>http://www.rakuten.co.jp</uri>
</service>

<operation name="getCategory">
  <extractor class="rakuten.CategoryExtractor">
    ...
  </extractor>
</operation>
...
</h2w>

```

Listing 1:An example configuration file for the automated generation of a service module Web Service.

- to construct an HTTP request encoding context information, which it received with the original SOAP request, and
- to extract context information from the received HTTP response and convert its HTML content (HTTP Entity Body) to XML format to construct an SOAP response.

while the role of a pluggable content extractor is:

- to send a request to (or locally invoke) a page-transition abstractor with context information and input parameters received with a SOAP request, and
- to extract an XML-formatted HTTP content included in a received response from the page-transition abstractor, by invoking a specified Web data extraction plugin using extraction parameters included in the received SOAP request.

We provide a Web Service operation (in terms of WSDL [14]) for each decomposed and abstracted service piece of the original Web application. For example with the shopping Web application of Figure 4, we provide an operation named “login” for the page transition from **Login** to **Top** page. The signature of the **login** operation can be:

$$\{c\} = \text{login}(c,u,p)$$

where **c** is a context information value propagated and updated through the service execution, and **u** and **p** are the parameters passed to the service for specifying a user ID and its password respectively.

3.3 Web Service composition with workflow

Once a Web application is decomposed and its page-transitions are abstracted, we can easily compose these decomposed modules to construct a Web Service for a desired purpose. The composition can be often written as a simple workflow program, such as BPEL4WS.

In a workflow, we propagate the context received from a service to subsequent services. Using parameters for the composite service, a workflow program invokes services along with an appropriate service slice. Then it

```

<operation name="getLoginForm">
  <input name="units_0" value="1" />
  <input name="submit" value=" Proceed to Order Page " />
  <extractor class="rakuten.LoginFormExtractor" />
</operation>

<operation name="authenticate">
  <param name="userID" input-name="u" />
  <param name="password" input-name="p" />
  <input name="deliver_to" value="me" />
  <input name="submit" value=" Next " />
  <extractor class="rakuten.AuthenticateExtractor" />
</operation>

```

Listing 2:Example operation-specific parameters.

```

<operation name="getCategory">
  <extractor class="generic.XSLTExtractor">
    <param name="xsl">rakuten/category.xsl</param>
  </extractor>
</operation>

<operation name="getSubCategory">
  <param name="category" type="xsdstring"/>
  <extractor class="rakuten.SubCategoryr">
    <param name="searchkey" inherit="category" />
  </extractor>
</operation>

<operation name="getMerchandiseList">
  <extractor class="generic.XPathExtractor">
    <param name="xpath">
      <xpath>//table</xpath>
    <extractor class="rakuten.MerchandiseExtractor" />
  </param>
</extractor>
</operation>

```

Listing 3:Example extraction plugin specifications.

construct a return value using the values returned from services used in the workflow.

For example, we can write a simple workflow program to provide a composite service operation:

$$\{d\} = \text{buyItem}(u,p,i,n)$$

where **d** is the return value representing a delivery date, while **u**, **p**, **i**, and **n** are parameters for a user ID, its password, the name of an item to buy, and the amount number of the item, respectively. Figure 6 shows the decomposed module services and their parameter values and return values specified in the workflow for the example **buyItem** service operation.

4. Framework implementation

In order to give a proof of concept, we have implemented a prototype framework for the proposed decomposition and abstraction model. This section describes the details of the framework implementation. The source code and the framework documentation are available at:

<http://h2w.sourceforge.jp/en/>

The implementation of the framework is fairly simple. The core of the framework has been implemented in about 20 classes and 1500 lines in Java, excluding libraries like HTTP handling and HTML parsing.

We have implemented the page-transition abstractor using Apache HttpClient (Jakarta Commons), which is a Java library for client applications based on the HTTP

```

...
public class ${service.name} {

    private static final String URI = "http://www.rakuten.co.jp/";

    public ${service.name}() {
    }

    public ApplicationSession init() {
        ApplicationSession session = new ApplicationSession(URI,
            ApplicationSession.GET_METHOD, new Cookie[0],
            new HashMap<String, String>());
        return session;
    }

    public HTTPData[] getCategory(ApplicationSession session) {
        ${extractor.class} payloadExtractor = new ${extractor.class}();
        Map<String, String> inputs = new HashMap<String, String>();
        ServiceStub invoker = new ServiceStub();
        return invoker.invokeHTTP(session, inputs, payloadExtractor);
    }
}

```

Listing 4: The service module implementation template.

protocol [2]. Subordinate links can be simply handled based on the functionality provided by HttpClient. In the rest of this section, we describe notable implementation issues other than that.

4.1 Pluggable content extractor

In order to analyze a web page for content extraction, the page is first passed through an HTML parser that creates a DOM tree representation of the web page. We use the NekoHTML [5] as the HTML parser. NekoHTML is a simple HTML scanner and tag balancer that enables application programmers to parse HTML documents and access the information using standard XML interfaces. Also, it corrects the errors in HTML documents, so we do not need to take care of such kind of problems related to HTML.

We do not describe further details of our DOM-based content extraction and its pluggable architecture in this paper. This is because DOM or XML based abstraction of HTML documents and their processing have already been well studied by Gupta [8] and our techniques used here is almost a subset of the work.

4.2 Context propagation

The notable difference in our implementation from previous work is that it handles context propagation through a process of content extraction, which is handled by plugin modules external to the framework. In our framework, in addition to logical data, a content extraction plugin must extract a hyperlink connected to the next page. The extracted hyperlink may simply be either an `<a>` element, a `<form>` element, or a `<button>` element. Extracted hyperlink information is used by the framework to achieve the context propagation.

Before passing a DOM tree representing an HTML document to a content extraction plugin, the framework does some preprocessing for accomplishing context propagation. In the preprocessing, it extracts application

session information embedded in an HTML document for each hyperlink and associates the extracted session information with the hyperlink. Embedded application session information consists of hidden input tags (`<input>` tag with type “hidden”) and the pointed URI by an `<a>` tag or an `action` attribute of the form tag. By aggregating those embedded session information with the Cookie information extracted from HTTP headers, the framework makes it a context information element.

After the process by a content extraction plugin, the framework picks the context information element associated with the hyperlink extracted by the plugin. In this way, the framework lets service module operation return the context to be propagated through an integration workflow.

4.3 Wrapper components generation

We implemented a generator for automating the implementation and WSDL description of decomposed service modules. First, the generator reads a configuration file and generates Java source code for the service implementation. Then it internally invokes a Java2WSDL tool to generate a WSDL file for the service modules.

Developers supply some service-specific information such as an intended service name to the generator in a configuration file. The configuration file looks like Listing 1. It consists of mainly two parts. The first, `<service>` element specifies the name of the service, the namespace of service, the location to be deployed, the location of the target Web application, and so on. The second, multiple `<operation>` elements specify service module specific parameters such as the name of a class implementing an data extraction plugin.

For each `<operation>` element, developers can specify what to be parameters of the operation, which parameters are used to fill input forms, what constant values are used to fill input forms, and so on. The `<operation>` elements in Listing 2 demonstrate this kind of specification.

For each specification of data extraction plugin, developers can give service-specific and plugin-specific parameters to the plugin. For example, in the specification of a service module named `getCategory` in Listing 3, it specifies an XSL file name supplied to a built-in XSLT-based extraction plugin. The example operation element for `getSubCategory` demonstrates specification of passing parameters to the extraction plugin. As another example, the specification for `getMerchandiseList` supplies an XPath expression and the name of another plugin class which is to be subsequently applied to the node set obtained through evaluating the XPath.

Given a configuration file, the generator generates a Java source code. It reads the configuration file to extract names and other parameters and then fills a source

code template looking like the one shown in Listing 4. Note that a service operation for initiating a service slice is automatically added without any specification by developers.

Finally, the generator compiles the generated source code and invokes Java2WSDL, which is a tool generating a WSDL file from existing Java code, available in Apache Axis [1]. Some parameters may be delegated from the configuration file to the Java2WSDL tool as the WSDL generation parameters.

5. Application examples

Based on H2W, we have successfully developed Web Service wrapper applications for three real-world Web applications. In this section we show these three examples: a shopping mall website, a university course management website, and an open source community website.

We use Ceri's taxonomy for Web applications [3] for characterizing the Web application examples shown here. According to Ceri's proposal, we can classify Web applications into five kinds: commerce, content, service, community, and context. We quote his description of the five kinds of sites here:

- *Commerce* sites sell the core object¹ (B2B, B2C, e-shops, e-malls, virtual marketplaces, e-auctions).
- *Content* sites give users information about the core object (digital libraries, online magazines, recommending systems).
- *Service* sites² offer the core object as a service (order-tracking sites and so on).
- *Community* sites build socially shared core objects (forums, chat rooms, newsletters, reselling systems).
- *Context* sites help to locate core objects (directories, search engines over local data).

All of the five categories of websites are covered by the examples shown here. First, each of the example application websites has the content and context facilities, as is often the case with complex websites. Additionally, the shopping mall website provides a commerce facility, the university course management website provides a service facility, and the open source community website provides a community facility.

¹ Here, *core object* means the data behind the application logic of websites.

² The terminology of *service* in Ceri's taxonomy is different from the one used through this paper.

5.1 Shopping mall service

The first target Web application is the Rakuten, which is the largest virtual shopping mall in Japan. The original website is accessible at:

<http://www.rakuten.co.jp/>

This website is a typical shopping site and similar to the amazon.com. Users of the website can choose a shop and buy items from the shop. The website mainly has the characteristics of commerce, content, and context.

Decomposed services

The selected WSDL operations for decomposed pieces from the original web application for a Web Service named **WineShop** are as follows:

- `init`
- `getCategories`
- `getSubCategories`
- `getMerchandiseList`
- `getMerchandiseInformation`
- `addCart`
- `getLoginForm`
- `authenticate`
- `selectPayment`

Composed service

Based on the decomposed services described above, we compose a Web service called **Sommelier** in WSDL service naming. This service recommends a wine item of the specified age by selecting the one with the cheapest price (`getCheapestWine`) and allows to buy it (`buyCheapestWine`).

The workflow of the `getCheapestWine` service operation involves invocations of `init`, `getCategories`, `getSubCategories`, and `getMerchandiseInformation`. The workflow of the `buyCheapestWine` service operation involves invocations of `addCart`, `getLoginForm`, `authenticate`, `selectPayment` in addition to the workflow of `getCheapestWine`.

5.2 University course management service

The second target Web application is the TDU Dynamic Syllabus, which provides course management services for students of Tokyo Denki University. The original website is accessible at:

<http://www.sie.dendai.ac.jp/ds/>

In this website, anyone can see the syllabuses for lectures and classes in the university. Also, a student can see the summary and time table of the courses he is enrolled in. The website mainly has the characteristics of content, service, and context.

Decomposed services

The selected WSDL operations for decomposed pieces from the original web application for a Web Service named **Syllabus** are as follows:

- `init`
- `getMainMenu`
- `getSyllabusSearchForm`
- `searchSyllabus`
- `getCourseInformation`
- `getLoginForm`
- `getEnrollCourseMenu`
- `getTimeTable`

Composed services

Based on the decomposed services described above, we compose a Web service called **CompositeSyllabus** in WSDL service naming. This Web Service provides two service operations: **searchSyllabus** and **getTimeTable**.

With the operation **searchSyllabus**, people can obtain the course information by giving the code number of a department, the name of a course, and the name of a professor. The workflow of the **searchSyllabus** service operation involves serial invocations of **init**, **getMainMenu**, **getSyllabusSearchForm**, **searchSyllabus**, and **getCourseInformation**.

With the operation **getTimeTable**, a student can obtain the list of courses that he can pursue during the current season. The workflow of the **getTimeTable** service operation involves serial invocations of **init**, **getMainMenu**, **getLoginForm**, **getEnrollCourseMenu**, and **getTimeTable**.

5.3 Open source community service

The third target Web application is the Source Forge, which provides a workshop for open source community. The original website is accessible at:

<http://sourceforge.net/>

In this website, developers can manage their open source projects and people may obtain information about these registered open source projects. The website mainly has the characteristics of content, community, and context.

Decomposed services

The selected WSDL operations for decomposed pieces from the original web application for a Web Service named **Sourceforge** are as follows:

- `init`
- `getMenu`
- `getLoginForm`
- `login`

- `getProjectMenu`
- `searchProjects`
- `getBugList`

Composed service

Based on the decomposed services described above, we compose a Web service called **CompositeSourceforge** in WSDL service naming. This Web Service provides one service operation named **getBugList**.

With the operation **getBugList**, a developer can obtain the list of pairs of bug information and URI for it, by giving his ID and password, the name of project, and the oldest date of bug information submissions. The workflow of the **getBugList** service operation involves serial invocations of **init**, **getMenu**, **getLoginForm**, **login**, **getProjectMenu**, **searchProjects**, **getBugList**.

6. Discussion

6.1 Other application scenarios

In addition to the demands for leveraging existing Web applications, there exist demands for developing human-oriented Web applications first even when both human- and machine- oriented systems are required. For example, developing human-oriented systems may have a relatively higher priority than the machine-oriented systems, which is often the case with today's software development projects.

In addition, there are demands for using external websites from local machines. The wrapper construction framework proposed in this paper can also be very helpful for this purpose. In this scenario, developers can use a wrapper proxy for their sites and make their application programs use the proxy locally, for example, by accessing "localhost" from applications located at the same site as the proxy. Even though information available to developers is more limited compared to the case of constructing wrappers for their own Web applications, this is possible and we designed our model and framework to be used in this scenario. Actually, all of the application examples shown in this paper were developed for this scenario. However, in such cases, we need to consider the legal aspects of reusing the original Web application. Some sites prohibit using their services from automated systems, and so on.

6.2 Limitations of the page-transition-based approach

Most of the recent widely used Web browsers allow an HTML page to dynamically display and interact with users while retrieving remote data asynchronously. Such techniques are called AJAX and are becoming popular

for certain types of websites like Google Maps. Since this is done using some embedded descriptions in scripting languages such as JavaScript, predicting what will be presented in the page is not easy. In addition, defining schema or static types for data extracted from an HTML page using AJAX is very hard since it tends to dynamically change the contents of the page. Page-based decomposition might not work well for Web applications with this type of content. Though it may be addressed by making a workflow program emulate scripts, it may not be so easy.

6.3 Related work

For the problem of extracting a single batch service from interactive services scattered over multiple webpages for human, little work has addressed the problem. Though ANDES [11] partially address this problem, it is in an ad-hoc manner. Their system employs a technique called hyperlink synthesis to allow XSLT filter modules to automatically transform dynamic hyperlinks (returned by submitting HTML forms) into regular static links with `<A>` tags. However, it is often impossible with practical complex websites.

7. Concluding remarks

In this paper, we proposed a fundamental framework, called H2W, for Web service extraction from existing Web applications. Our focus is on following the navigation design of the original Web application, while conventional Web data extraction techniques focus on removing presentation design and on recovering structural information behind its application logic. We propose decomposing the human-oriented services of a Web application on a page-transition basis and abstracting each transition from a webpage to another as parameter-passing and data extraction with context propagation.

H2W allows wrapper developers to incorporate the advantages of previous work on Web data extraction and to flexibly construct a desired service based on the decomposed and abstracted Web application service components. We showed that we successfully implemented Web Service wrappers with H2W for three real world Web applications that cover typical Web application types.

One of the future work in our mind is the further investigation of the data integration that we currently represent as a simple workflow. We do not think the process of Web Service composition by writing a workflow program is enough simple for satisfying requirements of extracting an integrated data gathered from multiple webpages. Further abstraction or tool support is required for

enhancing the ease of use of H2W. However, we believe that the proposed decomposition model and framework can be a good basis for building such abstraction or tools.

References

- [1] Apache Software Foundation, Apache Axis, Web Service Project, <http://ws.apache.org/axis/>
- [2] Apache Software Foundation, Apache HttpClient, Jakarta Project, <http://jakarta.apache.org/commons/httpclient/>
- [3] Stefano Ceri, Piero Fraternali, and Maristella Matera, Conceptual modeling of data-intensive Web applications, *IEEE Internet Computing*, 6(4):20–30, 2002.
- [4] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, March 15, 2001, <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- [5] Andy Clark, CyberNeko HTML Parser, <http://people.apache.org/~andyc/neko/doc/html/>
- [6] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo, Roadrunner: Towards automatic data extraction from large web sites, In *Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy (VLDB 2001)*, pages 109–118, 2001.
- [7] Piero Fraternali, Tools and approaches for developing data-intensive web applications: A survey, *ACM Computing Survey*, 31(3):227–263, 1999.
- [8] Suhit Gupta, Gail E. Kaiser, David Neistadt, and Peter Grimm, Dom-based content extraction of HTML documents, In *Proceedings of the Twelfth International World Wide Web Conference (WWW2003), Budapest, Hungary, May 20-24, 2003*, pages 207–214, 2003.
- [9] Stefan Kuhlins and Ross Tredwell, Toolkits for generating wrappers, In *Proceedings of Objects, Components, Architectures, Services, and Applications for a Networked World, International Conference NetObjectDays, NODe 2002, Erfurt, Germany, October 7-10, 2002*, pages 184–198, 2002.
- [10] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran Soares da Silva, and Juliana S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Record*, 31(2):84–93, 2002.
- [11] Jussi Myllymaki, Effective web data extraction with standard XML technologies, In *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 689–696. ACM Press, 2001.
- [12] Donald A. Norman. *The Psychology of Everyday Things*. Basic Books, 1988.
- [13] W3C, SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation 24 June 2003, <http://www.w3.org/TR/soap12-part1>
- [14] W3C, Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, <http://www.w3.org/TR/wsdl>