An Integration System of Web Information Sources for Mobile Users

Wisut Sae-Tung Tadashi OHMORI Mamoru HOSHI
The University of Electro-Communications, Graduate School of Information Systems
Tokyo 182-8585, JAPAN
e-mail: wisut@hol.is.uec.ac.jp

Abstract

Today, there are many Web Information Sources (WIS) suitable for mobile users. How/When to integrate such Web contents (which may be location-dependent contents) into a new simpler WIS for mobile users will become a problem, because different mobile users need different integration. A possible solution is to allow each individual mobile user to directly define, on his PDA, which pair of WISs should be integrated in what ways. Thereafter, system-side servers should materialize a definition (issued from a user's browser) as a new WIS. For this purpose, this paper introduces a new style of integration called **navigational integration** *of WISs, and then shortly describes a system architecture.*

1 Introduction

Recently, the number of autonomous Web sites which provide *location-dependent* information for mobile users has been increasing. A Web site (denoted by *Web Information Source* or *WIS* in this paper) is called *location-dependent* if it provides contents which vary according to current locations of the users. Typically, mobile users can get these Web contents through hand-held devices such as PDAs (Personal Digital Assistants) or smart phones. From a viewpoint of such mobile users, it will be helpful if location-dependent contents of such a WIS are integrated with services of other related WISs.

For example, assume that in an exhibition, there is a WIS which announces, to each user, the list of 10 booths nearest to his current location. Suppose that a mobile user (a participant) finds this WIS (denoted by WIS1) during his movement. Suppose also that he already knows another Web-document server (denoted by WIS2) which explains about the booths of this exhibition. Then, it will be useful if this mobile user can integrate, on his PDA, WIS1 with WIS2 into a new single WIS in such a way that output pages of WIS1 contain further links to the contents of WIS2. This new WIS will become a useful portal site for the user to move efficiently.

This paper describes a system prototype which supports the above style of WIS integration. Our originality is that each individual mobile user can directly define the integration of various WISs on his PDA according to his intention and location. This is in contrast with current Web integration studies [1, 2], in which a system designer must prepare all possible views of integration.

Figure 1 describes our assumption of a system environment. In Fig.1, an *area* is an autonomous unit of organization, and it maintains WISs which provide information related to this area. [1] An area must have a *MIR*, which is a yellow-page server that announces WISs of this area. Mobile users move across different areas either physically or electronically, visit MIRs and find interesting WISs from there. Concerning WISs, we assume that a WIS consists of a client-part (= Web-pages downloaded into users' browsers) and a backend database server. (i.e., we consider ordinary Web document servers or CGI/Java applications with database backend servers). Also, access to some WISs can be restricted to the access from certain areas. Some WISs may provide location-dependent information, and other WISs may be location-independent.

Under these assumptions, our goal is to let mobile users integrate services of appropriate WISs according to users' intention. Because such users move across different areas and are frequently disconnected, this integration must be defined successfully on a user's PDA even if it is disconnected from the network. Thereafter, when a PDA is network-connected, this definition must be materialized by system-side servers. For this goal, popular "pre-defined view" approaches [1, 2] need modifications. This paper describes a new style of integration called *navigational integration* for our goal, and overviews a system architecture.

[1] An area can be a geographic unit such as a building or a university campus, or it can be a non-geographic unit such as an organization.

0-7695-0789-1/00 $10.00 © 2000 IEEE

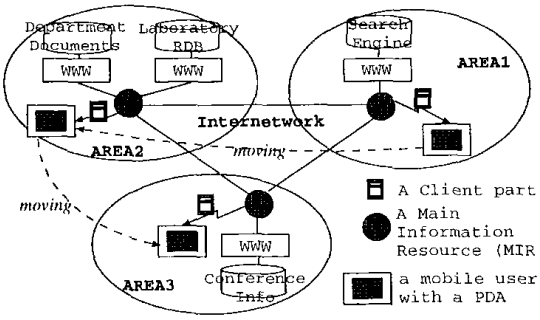

Figure 1: A system environment

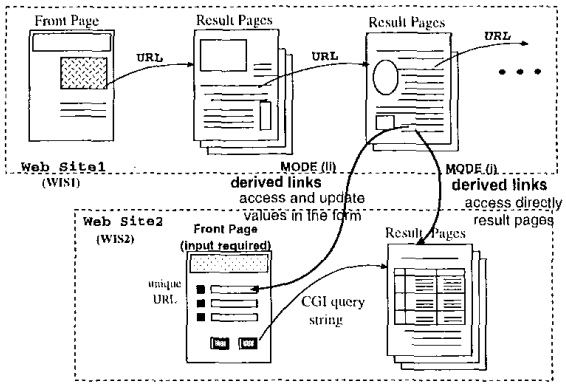

Figure 2: A model of navigational integration

2 Navigational Integration and System Architecture

Navigational integration is a style of integrating two WISs by adding *derived links* between Web-pages of the WISs. Roughly speaking, *derived links* are function-invoking links embedded at user-specified data-fields in Web pages of a WIS. These links work in such a way that when a derived link is clicked, a service function of another WIS is invoked under appropriate parameter-values of a current Web page of the current WIS.

Figure 2 shows how two WISs work as an integrated resource under the navigational integration. In Fig.2, assume that the first WIS (WIS1) is a (location-dependent) Web-document server which provides mobile users with time- and location-dependent information about one area (such as a conference program in a building). Assume also that the second WIS (WIS2) has a CGI-form page (as a user interface) with a back-end database server, and it provides information about a larger area (e.g., about laboratories of departments in a university). Furthermore, each of the WISs is assumed to consist of a front-page and server functions, in such a way that the front-page accesses result-pages through the server functions. [2] Under these assumptions, Fig.2 shows two kinds of derived links from WIS1 to WIS2. These links pass data-values from the result pages of WIS1 to WIS2, and work in either of the following modes: (i) one mode is to directly invoke the service of WIS2 under the passed data-values and to access its result pages; (ii) the other mode is to set input-parameters of the front page of WIS2 by the passed values and to allow the user to enter additional parameters for further operations. (In either mode, the passed data-values are those of data-fields at which derived links are embedded). In this way, *derived links* are the links which invoke service functions of a WIS under appropriate data-values of a current Web page. As a result, two WISs work as a single WIS.

Our objective is that navigational integration of WISs can be defined by a mobile user freely on his PDA. To realize this goal, our system must support the following three points: i) a user caches a minimal set of metadata about related WISs when he finds them; ii) Based on the cached metadata, a user writes a query command that defines navigational integration; this must be done solely on a PDA during network-disconnection; iii) When the PDA is re-connected to network, the user's browser issues the query to system-side servers; the query is then executed as a single WIS resulting from the navigational integration.

Figure 3 shows our system architecture satisfying the above points. This is the case of three areas. Each area has a *main information resource* (MIR) as a yellow-page server. In MIR, there are a *wrapper* and a *proxy executor*. The *wrapper* must wrap client-parts of WISs in a common (object-oriented relational) data model. Namely, the data structures and server-calling functions of a client-part are described by *interface definitions*(ID). Furthermore, all MIRs share a common repository called *domain hierarchy*, which is used for automatically resolving semantic conflict of data between areas (see [3, 4]).

Under these preparations, while moving, a mobile user works as follows: firstly, he sets up a connection to the network, accesses a MIR, and downloads certain metadata (of interesting WISs) from the MIR into his PDA (denoted by machineX here). These metadata are: the interface definitions (IDs) of such WISs and a minimal set of *domain rules* [3] related with these IDs.

[2] As a terminology, a *front page* of a WIS refers to the first Web-page that a user downloads when accessing the WIS. The other pages of the WIS are termed *result pages*. Result pages are generated by calling server-functions of the WIS under some parameters from a given Web page. In our study, each of these Web pages together with its server-calling function is regarded as a client-part.

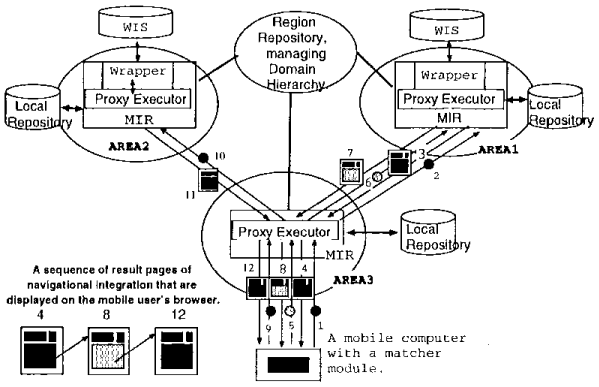

Figure 3: A system architecture

(These are the only metadata to be cached). In this way, a user moves across areas and collects IDs from there. Next, the user disconnects machineX from the network, and activates a *matcher* on machineX. The *matcher* is a graphical query builder, which helps the user to define navigational integration by a query command. (It uses the cached domain rules for semantic conflict resolution). Finally, the user again sets up a connection to the network, and materializes navigational integration by sending the query command to a proxy executor from machineX.

Figure 3 shows the steps (1-12) to materialize a query of navigational integration. There, the proxy executor nearest to the user receives a query (step 1), and controls related wrappers and other proxy executors. These servers materialize one Web page at a time according to user's requests of link traversal. Namely, in response to one request of link traversal, an appropriate result page is generated, and derived links are embedded in the page according to the query command, and then that page is replied back to the user's browser.

3 Implementation overview

This section describes the interface definitions of WISs and query commands that define navigational integration. Domain rules and heterogeneity resolution techniques on PDAs are explained in [3, 4].

In our study, a WIS is modeled by a *client-part* and a backend database server. A *client-part* refers to the part downloaded from a WIS into a user's browser. That is, a client-part contains two components: i) the user-interface part (i.e., a Web-page visible to users), and ii) the data-access part (which is a function to call a backend server method and to receive new pages as results). We wrap these components individually by Interface Definitions (IDs). An ID represents one class (= an object-oriented relation). One instance (= a tuple) of this class represents one Web page (in default). Its class method represents a page-generation function.

An ID has the following syntax:

database RepositoryName
address URLofProxyExecutor
class IDName
body
    Attribute DataType [,Attribute DataType]
method
public:
    MethodSignature [;MethodSignature]
private:
    MethodSignature [;MethodSignature]
implement: ...
endclass

The above ID tells that a class *IDName* is maintained in a repository *RepositoryName* at a MIR of url *URLofProxyExecutor*. This ID has a list of (*attribute*, *data_type*) pairs, denoted by (A_1 T_1, A_2 T_2, ... , A_n T_n). A_i and T_i are an attribute and its data type, respectively. A tuple (= a Web page of this class) has an instance value belonging to these (*attribute*, *data_type*) pairs. The data type can be an *atomic domain* [3] or can be constructed from data types. The constructors are: `listof` *IDName*, another *IDName* (in these cases, *IDName* represents a substructure of a Web page), `logical OR` and `concatenation` (for semi-structures of a Web page), or `link` to another Web page. Methods have a form *methodName*(D_1 AR_1 , ... , D_n AR_n), where D_i is the data type of an argument AR_i. A public static method is used to generate pages of this class under given parameters. These methods can be used in query commands.

As an example, consider a CGI-based WIS whose front page is a CGI-form and its data-access part is a call to a database server. Consider that this WIS provides a ranked list of presentation programs held in a building, ranked by the time and location of a user.

Then, the following is the ID wrapping this front-page. (This ID represents the front page and its generation function):

```
database ISBuilding
address http://HOST1/cgi-bin/WrapperExecutor.pl
class BldGuideF
body
  submit    link FloorLabInfo, # invoke getBystaticMethod of
                               # the FloorLabInof ID.
method
public:
          # a method to generate the front page.
  static getBldForm(ResearchField $researchField,
                    string $direction );
private:
implement:   ...
endclass
```

[3] An *atomic domain* is a pool of atomic data-values which have exact formats and meanings.

The following is the ID wrapping the data-access part (= a class of result pages):

```
database ISBuilding
address http://HOST1/cgi-bin/WrapperExecutor.pl
class FloorLabInfo
body
  lab         JLab, # atomic value whose domain is Jlab
  floor       int,  # atomic value whose domain is integer
  sessions    listof Session, # list of objects of Session ID
method
public:
            # a method to generate result pages.
  static getByGuideConds(ResearchField $rsf, int $floor,
                         string $direction,Time $time);
private:
  # extract parameter from the query string and invoke
  # the method getByGuideConds. This is invoked by the
  # wrapper of this class when the double arrow (=>) part
  # in a path expression (X->a=>Y->z->..)  is evaluated.
  # As a result, a result page of FloorLabInfo is generated.
  #
  static getByStaticMethod(String $queryStr);
implement:   ...
endclass
```

A data-field in a Web page can be specified by a *path expression*, which has the form:

$$t_0 \rightarrow \{A_1\} \rightarrow ... \rightarrow \{A_i\} \Rightarrow IDName \rightarrow ... \rightarrow \{A_n\}$$

where t_0 is an instance object variable describing this page. The single arrow ($\rightarrow$) denotes a substructure within a single Web page. The double arrow ($\Rightarrow$) followed by $IDName$ means a link to a different Web page of $IDName$.

Next, we describe a query command defining navigational integration. It has the following syntax:

from ObjVariable in IDName [,ObjVariable in IDName]
source IDname of RepositoryName on URLofWrapperExecutor
[,IDname of RepositoryName on URLofWrapperExecutor]
where IDName→methodName($arg_1, ..., arg_n$)
[**and** IDName→methodName($arg_1, ..., arg_n$)].

Here, arg_i can be a constant or a path expression which specifies a data-field of a previous Web page. A query is executed by evaluating `where`-clause conditions in the listed order. A unit step of the evaluation is i) to evaluate a double arrow in a path expression, or ii) to evaluate IDName→methodName(). At each of these steps, a corresponding Web-page generation method is invoked by a proxy executor, and thereby a new result page is generated; furthermore, in this page, derived links are embedded at those data-fields which are specified by the remaining path expressions related to this page. (Derived links are implemented as CGI-calls to a remaining subquery stored in a proxy executor).

As an example, consider the following query:

from B in BldGuideF, D in Department, A in AltaSearch
source BldGuideF of ISBuilding on http://HOST1/cgi-bin/Proxy.pl,
Department of ISInformation on http://HOST2/cgi-bin/Proxy.pl,
AltaSearch of NewAltaVista on http://HOST3/cgi-bin/Proxy.pl
where BldGuideF→getBldForm("","")
and Department→getByKeyWord(B→{submit}⇒FloorLabInfo→{lab})
and AltaSearch→getByKey(D→{laboratories}→{staffs}→{fullName}).

This command defines a navigational integration of three WISs. The first WIS is a CGI-based one whose IDs have been described above. This WIS has a CGI-form front-page (wrapped by `BldGuideF`), and it provides, as a result-page (wrapped by `FloorLabInfo`), a ranked list of presentations of laboratories in a building. (This list is ranked by the current time and location of a user). The second WIS is a Web document server whose result-pages (`Department`) are about laboratories of departments in a university. The third WIS is the Altavista Web site, whose result-pages (`AltaSearch`) are the results of searching. Then, this query works according to the mode (i) of Fig.2. That is, this query is materialized as a new WIS whose front page is that of the 1st WIS, but its first result-page (which is the same as that of 1st WIS) contains, at the data-fields of `lab`, new function-invocation links to result-pages of the 2nd WIS; these result-pages (of the 2nd WIS) are retrieved by getByKeyWord() method from the 2nd WIS under the data-values of `lab`; furthermore, these next-level result-pages also contain new derived links at the staffs' `fullName` fields; these links are to call AltaVista search engine under the data-values of `fullName`. In this way, this query works as a single WIS resulting from the navigational integration.

In IDEAS 2000 presentation, we will demonstrate how this example works as a single WIS, and will show how a mobile user can build such a query solely on a PDA. Our system is described in details in [3, 4].

References

[1] S. Chawathe. et al. The tsimmis project: Integration of heterogeneous information sources. *Tech. Rep of 100th DBS of Information Processing Society of Japan*, pp.7–18, October 1994.

[2] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the world-wide web: Survey. *SIGMOD Record*, 3(27), pp.59–74, September 1998.

[3] W. Sae-Tung, T.Ohmori, M.Hoshi. Navigational integration of autonomous web information sources by mobile users. *Proc. IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing*, pp.270-275, August 1999.

[4] W. Sae-Tung, T. Ohmori, and M. Hoshi. Integration of Web Information Sources by Mobile Users: Navigational Style of Integration and System Architecture. to appear in *Proc. ADBIS-DASFAA 2000 (Proceedings of Challenges)*, September 2000.