Navigational Integration: a novel style to Integrate Autonomous Web Information Sources for Mobile Users

Wisut Sae-Tung Tadashi OHMORI Mamoru HOSHI
Graduate School of Information Systems
The University of Electro-Communications, Tokyo, Japan

Abstract

In today's computer internetworks, many information sources provide their contents via WWWs by embedding their client applications into Web pages. Let us call such an information source a *Web Information Source (WIS)*. In general, these WISs are maintained by multiple autonomous organizations. Usually, in this environment, mobile users find some WISs located in different organizations, and download client applications from those WISs to query their information. Typically, these WISs contain related information among themselves, but how to use these WISs in an integrated manner without destroying original style of usage of the WISs is still a challenging topic, because such WISs may be documents or dynamic ones using database invocations.

Motivated by this background, this paper proposes a novel style of integration of WISs for mobile end-users. Our integration style, called navigational integration, is based on the hyperlink concept of the WWW framework, and promotes utilization of database calls of each WIS. Furthermore we aim at the facility that mobile users, on their portable computers, can freely create a new client application to materialize the navigational integration; also, this creation must be done on the users' computers in a communication-disconnected mode. This paper describes the navigational integration and proposes a necessary language/architecture based on wrapper/proxy-executor/mediator(s).

1 Introduction

In today's computer internetworks, many information sources provide their contents via WWWs by embedding their client applications into Web pages. For example, ordinary

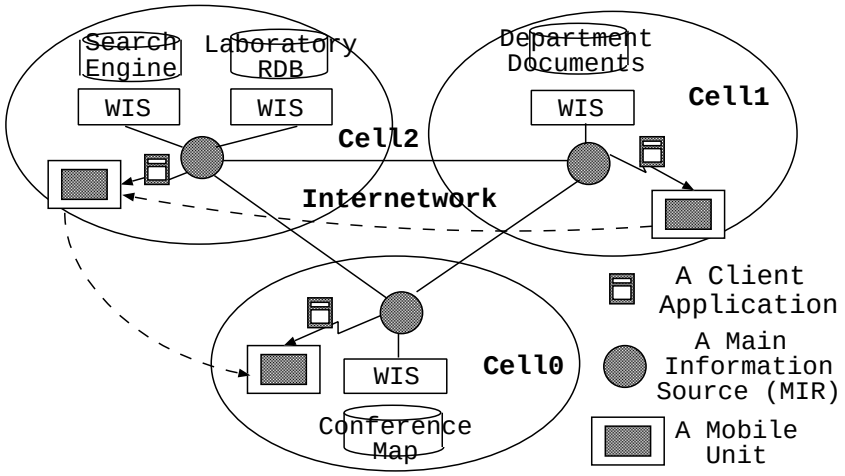

Figure 1: Web Information Sources in a mobile environment

HTML document servers, CGI-form texts with backend database servers, and java-based clickable maps with backend data services can be found everywhere in the Internet. Let us call such an information source *a Web Information Source* (WIS). In general, these WISs are maintained by multiple autonomous organizations. Let us call such an organization a *cell*. In this environment, mobile users move across cells and download client applications from WISs to query information. This environment is shown in Figure 1.

Typically, these WISs contain related information among themselves. Defining integration among these WISs by mobile end-users using an ad-hoc queries has become increasingly important in the near future for several reasons. Firstly, recent technologies make the size of computers smaller so that end-users can take and use them while moving [14]. Currently, using such devices to access information from public networks is a common way that can be seen everywhere. Secondly, a lot of public information services has already been provided through the WWW. Defining static view over these information sources for all need of users is impossible. Integration of these WISs requires user interactions based on ad-hoc queries, rather than the use of pre-defined views. Finally, representation of Web documents is complex, and so extracting data from Web documents and writing queries to recompose the data into a new representation (view concept of traditional database) is impossible for end-users. A tool for end-users should be provided to help them to generate such ad-hoc queries. The combination of these trends give a great demand for system that mobile end-users can freely issue ad-hoc queries and create new client applications to integrate several WISs with little technical knowledge as possible, without destroying original information and style of usage of the WISs.

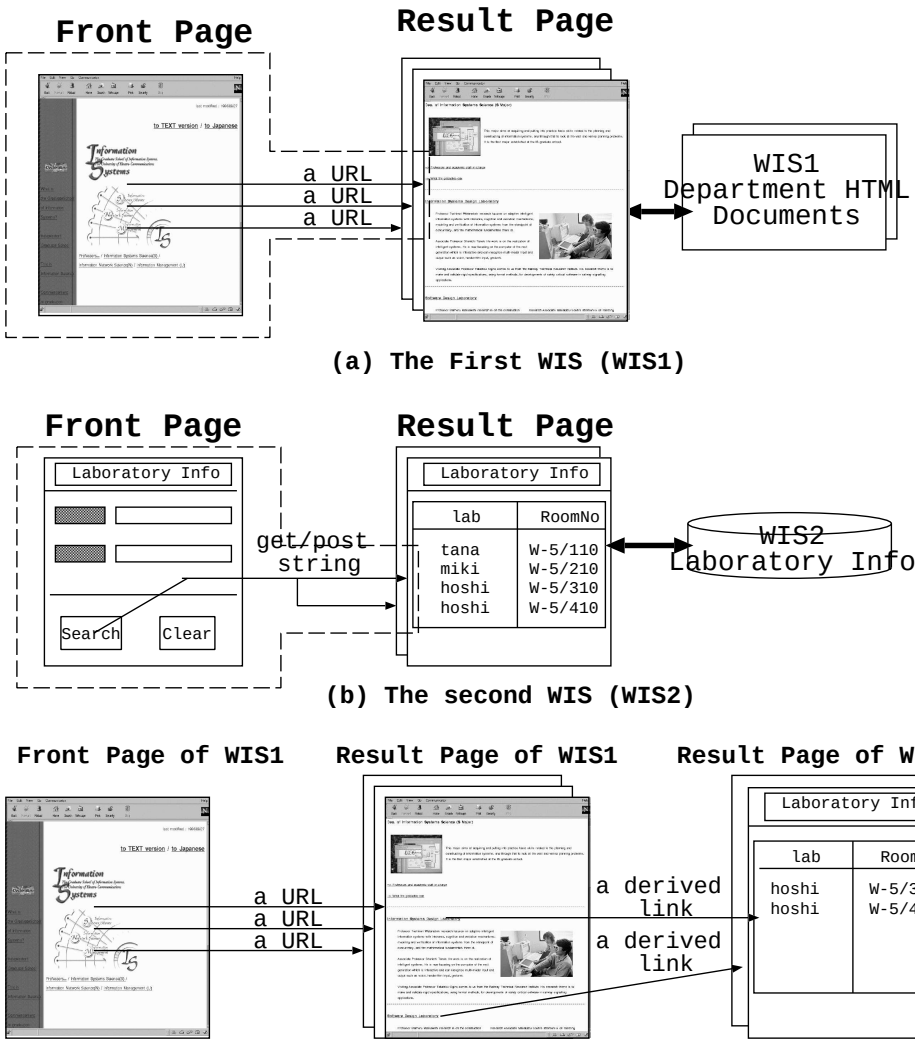

Figure 2: Navigational integration

This paper proposes an appropriate style of integration for mobile users who move from one place to another, find new WISs and generate a new client application to integrate those WISs by themselves on their mobile computers in a disconnected mode. Our integration style, called *navigational integration*[17, 18], is based on the hyperlink concept of the WWW framework and promotes utilization of filtering methods provided by each WIS. Previous works [2, 7, 9, 11, 12] discuss integration of heterogeneous information sources (including WISs) on a centralized site of developers but do not consider an appropriate style of integration for mobile users.

1.1 Example

To understand our navigational integration clearly, let us explain the navigational integration by using the example in Figure 2. Suppose that there are two WISs. In Figure 2-a, the first WIS (WIS1) is an ordinary WWW document server. It consists of a Web front-page and some HTML result pages (under a given DTD) describing departments; further, the front page includes URL links to these HTML-pages. In Figures 2-b, the second WIS (WIS2) is a CGI-based WIS having a front page through which a user can input condition parameters and activate a button link to retrieve a dynamic result page. Thus, the links on the front pages can be regarded as methods that invoke the information server of the WIS to generate the result pages. Then, the user wants to generate a new WIS (WIS3) from the original WISs by embedding *derived links* that invoke methods of the database server of WIS2 into the result pages of WIS1. In this example, the derived links pass the data (i.e., laboratory name) from the output page of WIS1 to WIS2 as the input conditions to get the result page of WIS2 (see Figure 2-c). When the user selects these derived links, he can navigate between contents of the web pages of two WISs.

The style of integration described above is called *Navigational Integration.* A newly-generated WIS (e.g., WIS3) can be regarded as a *navigationally-integrated* view among several original WISs. Then, our goal is to allow mobile end-users to define such a navigationally-integrated view[1] on mobile computers.

1.2 The Goal

To achieve our goal, note that only *client-parts* of WISs are downloaded to users' computers. A *client part* of a WIS consists of a front page and data-access functions of the WIS (e.g., as shown by the dotted areas in Figure 2-a and 2-b). Thus, the following ability is required: *by using the downloaded client-parts, mobile users can generate a new client-part which performs navigational integration among the WISs.* Further, this generation must be done solely on a mobile computer in a communication-disconnected mode.

Let us return to Figure 1 and show the overall processing steps taken by mobile users. In this environment, mobile users move across cells and collect client-parts of WISs. Next, they can disconnect their computers from networks and define navigational integration by using original client-parts on their mobile computers. Finally, they reconnect their

[1] By the term "define navigational integration", we mean to define a navigationally-integrated view.

mobile computers to the networks and execute new client-parts to perform navigational integration.

The paper is organized as follows. Related works are described in Section 2. Section 3 describes a system overview. Section 4 describes a system design. Section 5 describes an automatic semantic-conflict resolution that is a necessary mechanism for mobile users to define their integration. Section 6 describe enhancements of navigational integration to support general Web documents in terms of common data model, path expressions, and algorithm of conflict resolution, followed by concluding remarks.

2 Related Works and Our Contributions

In the past, there have been lots of works in the area of heterogeneous information integration [9, 12, 15] and Web site management systems [7, 11, 13]. The Web is one of information sources used in these works. In these works, integration is based on the notion of *view* in the traditional database; data from existing Web sites are extracted, integrated and restructured to be presented in different perspectives. They restrict end-users to access to the server that defines the static view. Other works propose query languages [3, 5, 6] for manipulating the Web in the database style. However, it is too complicate for end-users to learn and use them directly. The research on the design of a system architecture and tools for end-users (especially for mobile end-users) to create a new client application to integrate the WISs is not investigated insufficiently.

Motivated by this background, navigational integration of our approach (shown in Figure 2) is concerned with the process of embedding service invocation of one WIS into the result pages of another WIS. This style of integration is suitable for the navigating and browsing information the WWW framework. Extracting data from original Web sites and restructuring them into another presentation that is a complicated task need not to be specified by end-users. Thus, the query language to define navigational integration is easy for end-users to learn and understand. Moreover, to define navigational integration more easier, we also provide a tool for generating a new client application embedded with the query command defined by mobile end-users.

The conflict resolution [2, 4, 10, 13, 16] is one of important problems in integration systems. The previous researches based on a wrapper/mediator architecture solve this problem by defining data conversion rules in the middleware *mediator* in a server. How-

ever, such a static manner cannot be applied to our approach that allow end-users define their own integrations. We extend the current integration system by allowing independent development of definitions of WISs and their data conversions. Moreover, these (necessary and sufficient) information are downloaded and exploited by the mediator on mobile computers to resolve semantic conflict and generate a new client application in a network-disconnected mode. The details of system architecture and semantic conflict resolution will be described in Section 3 and 5, respectively.

Our contributions are in (1) a new style of information integration, called *navigational integration*, for mobile end-users in the WWW environment, (2) a new system architecture providing mobile end-users with an ability for defining and executing navigational integration, and (3) dynamic semantic conflict resolution by downloading necessary information from WISs and performing resolution on mobile end-users' computers in a disconnected mode.

3 System Overview

3.1 Model of a WIS

To apply a WIS to our approach, it must be a two-part structure, consisting of a backend WWW server and a client application embedded in its Web page as shown in Figure 3. The client application must be divided into two parts: a front page part (FP) and a data access part (DA). The FP part receives input from a user while the DA part sends the input data to the backend server and receives a result page. Thus, the DA part consists of a server-access method and specification of structure of data in the result page.

WISs of ordinary HTML documents and WISs with CGI programs satisfy the above model, but a Java applet satisfies our approach if its methods implemented for the FP part and those for the DA part are separated.

3.2 Our Approach to Define Navigational Integration

To integrate information among several WISs, a common data model is required for wrapping WISs into a uniform manner. Based on the model of a WIS shown in Figure 3, the pre-processing is that the FP part and the DA part of a WIS must be wrapped into a common data model. We use the object-oriented (OO)[1] data model to describe structure of data and methods of these two parts. Thus, we can use a concept of *path expressions*[8]

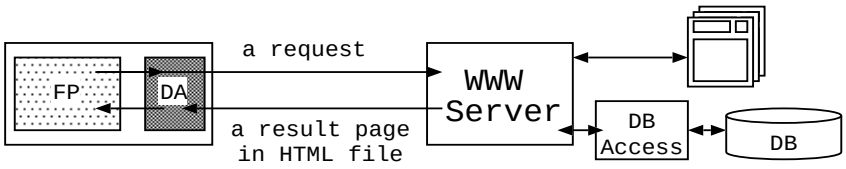

Figure 3: Components of a WIS

to specify data elements and can invoke server-access methods through *interface methods*.

After WISs are wrapped into the common data model, now let us consider the way to define navigational integration in a declarative form and the way to generate a new client-part for executing the navigational integration.

To define navigational integration in a declarative form, we use a query command whose WHERE clause specifies predicates that define how to express derived links in the navigational integration. Consider navigational integration from a WIS S to another WIS R. In general, the navigational integration from S to R can be defined by a WHERE clause which has two parts giving derived links. The first part gives a derived link from the front page of S to a result page of S. This part is written as an invocation of the server-access method of the interface definition that wraps S. (e.g., see the WHERE clauses of Fig.4-a or 4-b.) This method is invoked with variable(s) that will be bound with constant values later. On the other hand, the second part gives a derived link from a result page of S to a result page of R. This part can be written in two different modes. Figure 4 illustrates these two modes with their WHERE clauses. These modes are as follows:

(i) The first mode (Figure 4-a) is to embed the server-access method of R as derived links into the data elements in the result page of S. In each link, the data element of S is passed as argument values of the method. Through this server-access method, the native method understandable by R is called to retrieve a result page.

(ii) The second mode (Figure 4-b) is to embed the server-access method (of R) and equality predicates as derived links into the data elements of the result page of S. In each link, a result page of R is retrieved (by the first mode) and then another data element in the result page of S is related with an element of the result page of R.

In this way, navigational integration is described by a query command. The next step is to build a new client-part. To do this, the query command must be embedded into the front page of S. In case that S is an ordinary document, the variables in the first

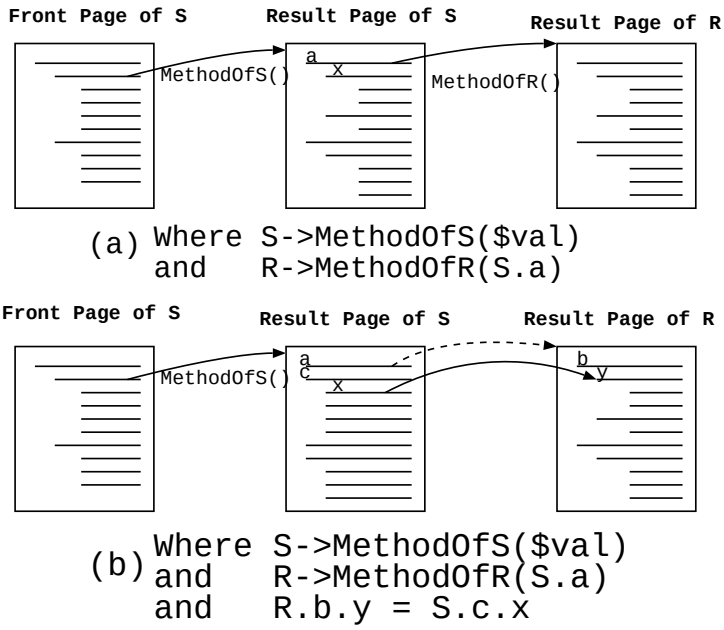

Figure 4: Two modes of navigational integration (written by WHERE clauses)

predicate in the WHERE clause are bound with constant values of data-elements on the front page of S. In case that S is a CGI-form WIS, the variables are bound with the parameters in the CGI-form. Thus, users can execute the query command through the front page of S. In the next subsection, we describe a system architecture for executing the query and its processing steps.

3.3 System Architecture and Integration Process

To explain a system architecture and the cooperation between its modules for executing navigational integration, let us use Figure 5. In this figure, there are three cells. All cells share a common knowledge called *domain hierarchy* for automatically resolving semantic conflict of data between cells (see detail in Section 5). Each cell has a *main information resource* (MIR) as a yellow-page server. In addition, a *wrapper* module that wraps a WIS is placed on the MIR. The client-part of each WIS is wrapped into interface definitions described in Section 4.1. In the existing integration systems[12, 13], they place a mediator module on a server for: (i) resolving semantic conflict among data of WISs, and (ii) executing a user's query. In contrast, we handle these two works by using two different modules. For the first task, we place a *mediator* module on the mobile computer side. Our *mediator* uses the downloaded client-parts of WISs, helps a mobile user to solve semantic conflict among data, and define navigational integration in a query command. Further,

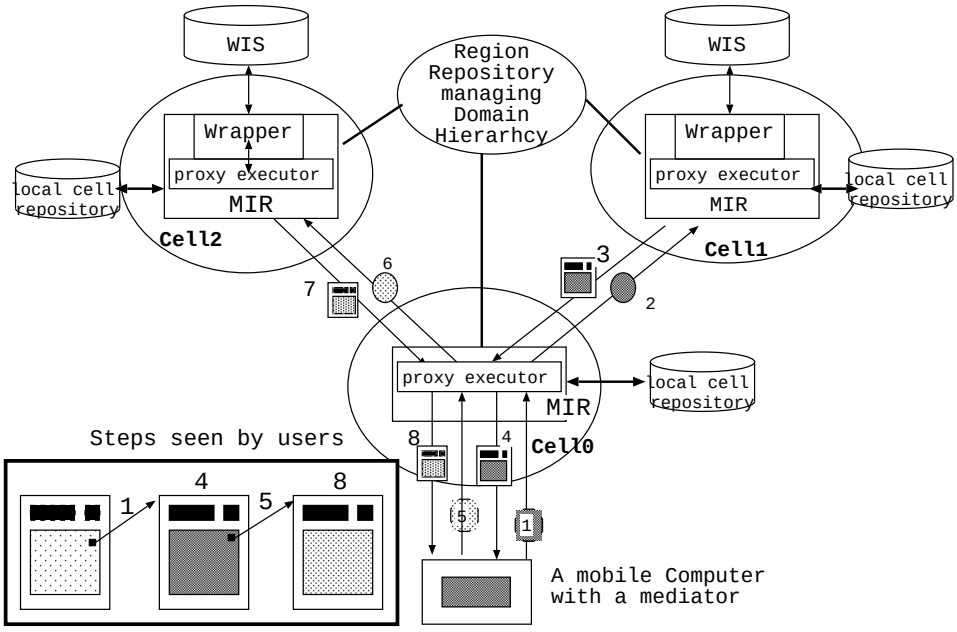

Figure 5: Processing of navigational integration

it automatically embeds the query command into the front page of WIS1. For the second task, we place the module called *a proxy executor* for performing this task on the MIR.

Then, a user can execute navigational integration by loading the new client into his browser. The steps of processing are shown in Figure 5. By selecting derived links, the user can send the query command to a proxy executor. This proxy executor controls cooperation between other MIRs in different cells. Exactly speaking, it communicates with the other proxy executors of the MIRs that manage the WISs specified in the query command to fulfill its work. However, from the users' view, the processing steps of navigational integration proceed in a sequence shown in the left lower box of Figure 5.

4 System Design

Based on the framework in the previous section, this section first describes a common data model for wrapping a WIS. Then, we describe path expressions and a query language for expressing the navigational integration.

4.1 Common Data Model

The common data model we use is a variant of the object-oriented (OO) model. As a unique property, we add a concept called *domain hierarchy*, which will be described later in Section 5.

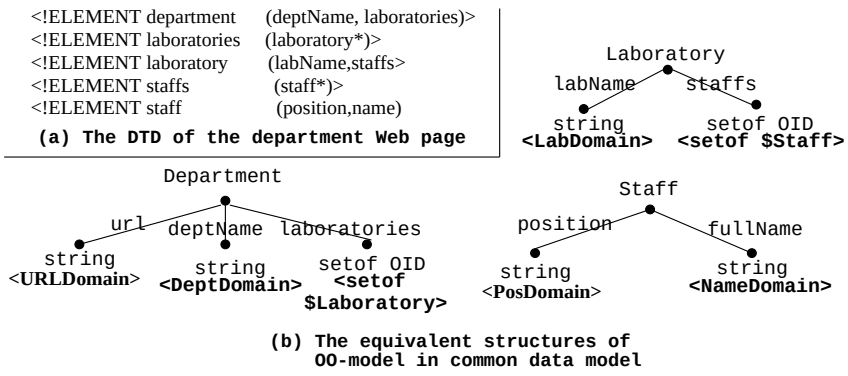

Figure 6: A DTD and an equivalent schema

In our policy, a web page is regarded as a nested structure of objects. Further, in order to model a function-embedded WIS such as a CGI-page, we give methods to this structure. Such a structure of objects with methods is called an *interface definition* in our system. Then, we wrap the service of a WIS by a set of interface definitions. (Nested data occurring in a web page are represented as relationships between interface definitions via multi-valued attributes.)

Let us show how to wrap the DA part of the first WIS of Figure 2-a. This WIS is assumed to be a structured document, whose DTD is given in Figure 6-a. Then, Figure 6-b describes an equivalent schema in our data model and Figure 7 shows one part of the interface definitions wrapping the first WIS.

In order to explain interface definitions, let us use the interface definition of *Department* shown in Figure 7. This interface definition has three attributes: *url*, *deptName* and *laboratories*. The second attribute is a single-value of string and its domain is *DeptDomain*. The third attribute is a multi-value keeping OIDs of instances of a Laboratory interface. Using such a multi-valued attribute, we can represent nested structure of documents. The first attribute *url* is an additional element keeping the URL of the original document page. It does not appear in the DTD shown in Figure 6-a.

As shown above, the data-types in our system are atomic data or OID(s) (as an object or a set of objects). *Domain* must be assigned for all atomic data. Here, a *domain* in our approach refers to a pool of values that have an exact data-format and meaning.

In the method part, there are two types of methods; a **public** type and a **private** type. Only *public* methods are invoked explicitly. In this example, *getByURL()* method is the way to retrieve an original document having a given URL.

Some of public methods can have a keyword *static* in their method signatures. Roughly

```
database ISInformation
address  http://HOST1/cgi-bin/execute.pl
interface  Department
body
    url             URLDomain,
    deptName        DeptDomain,
    laboratories    setof Laboratory,
method
public:
  static {DepartMent} getByURL(URLDomain $url);
private:
  OID  new(string $deptName,OID $lab)
implement
 sub new {
     my $package  = shift;
     my $deptName = shift;
     my $labOID = shift;
     my $this;
     $this = DepartMent->newEmptyObject();
     $this->{deptName} = $depName;
     push(@{$this->{laboratories}},$labOID);
     bless $this;
     return $this;
 }
  ...
endInterface
```

Figure 7: An interface definition for Department

speaking, a static (and public) method indicates a class method to retrieve a result page of a WIS by invoking a native method and extract data from that result page to create objects. The other methods are used for filtering such generated data-objects.

4.2 Path Expressions and Query Language

A path expression has the form

$$t_0 \rightarrow \{A_1\} \rightarrow ... \rightarrow \{A_n\}$$

where t_0 is a variable representing an object instance of a top-level interface definition (T_0) in a Web page. Further, $\{A_i\} \rightarrow \{A_{i+1}\}$ means that the domain of A_i is defined as **setof** T_{i+1} where T_{i+1} has an attribute A_{i+1} (see the example of the interface definition in Section 4.1). The result of the path expression is the set of objects or values that can be reached from t_0 via the specified attribute chain. For example,

$$D \rightarrow \{laboratories\}[*] \rightarrow \{staffs\}[0] \rightarrow \{fullName\}$$

is interpreted to get all full names of the first staffs in all laboratories in the department where D is a variable representing an object instance of the interface $Department$. The $[j]$ is used to specify the object ordered j and $[*]$ is used to specify all objects in a multi-valued attribute. For convenience, this $[*]$ can be omitted.

Here we are ready to demonstrate how to define a navigational integration by a query command. Assume that a mobile user finds the two WISs (WIS1 and WIS2 in Figure 2-a and b) and wants to define navigational integration of them (WIS3 in Figure 2-c). The following script is a query command to describe navigational integration from a given (single) department page of the first WIS (whose url = **$URLx** and is bound with a constant value later) to the second WIS:

```
select *
from   D in Department, L in LabBuilding
source Department of ISInformation
       on http://HOST1/cgi-bin/execute.pl,
       LabBuilding of uecdbWeb
       on http://HOST2/cgi-bin/execute.pl
where  Department->getByURL($URLx)
and    LabBuilding->getByLaboratory(
         &L:Lab2JLab(&D:LabD2Lab(
          D->{laboratories}->{labName})))
```

For example, in the above SQL-like statement, *getByLaboratory* is the method defined in the interface definition of the second WIS; this method is the major function of this WIS. As described earlier, data in a Web page can be specified using path expressions. The path expressions $D \rightarrow \{laboratories\} \rightarrow \{labName\}$ extracts the laboratory names appearing in the department Web page. Due to the semantic conflict between laboratory data in WIS1 and the required argument of the method *getbyLaboratory*() of WIS2, the LabD2Lab and Lab2JLab functions are inserted to resolve this conflict in order that data of WIS1 are converted into data of WIS2. For instance, a laboratory name 'Information System Design Laboratory' is converted to a leader professor's name 'Watanabe' (Japanese always use a leader professor as a laboratory name). We will describe how to detect the semantic conflict and its resolution by mediators on mobile computers in the next section.

To help a user issue this request easily, we implemented a user interface of the mediator as shown in Figure 8. This mediator explores information of the client-part downloaded from cells and interacts with the user through this interface (without typing) to generate a query command. After the user complete his requirement, the mediator automatically embeds the query command into the front page of the first WIS. This is easily done in case of ordinary Web pages or CGI-form pages by replacing the execution tag of HTML source with the query command. In case of a Java applet, it requires the template to generate a Java program that inherits the original applet and overrides the methods that act as the DA part of the first WIS.

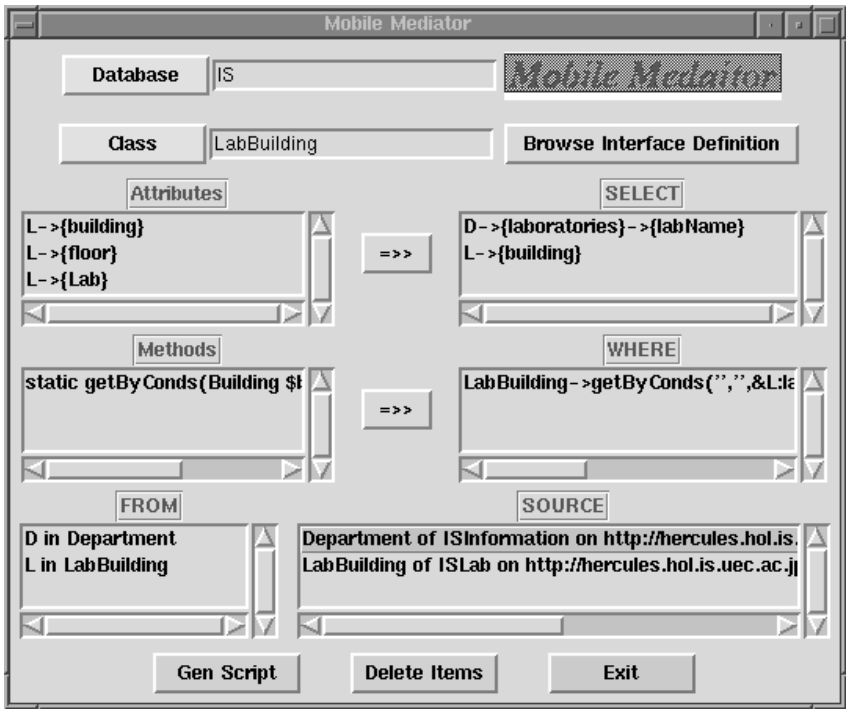

Figure 8: A graphic user interface of a mediator on a mobile computer

As a result, the above SQL-like script can execute navigational integration (of the Data-Access parts) with resolving semantic conflicts.

5 Semantic Conflict Resolution and Script Generation

To provide mobile users with ability for defining navigational integration, the semantic conflict among data of different WISs must be resolved automatically. To achieve this ability, we propose a *domain hierarchy* (DH), which is maintained among all cells in a given network. Here, a *domain* in our approach refers to a pool of values that have an exact data-format and meaning. Figure 9 shows an example of the domain hierarchy, which is maintained by two cells. The domain hierarchy consists of two components: (i) multiple local domain hierarchies (LDHs); each LDH is maintained individually in each cell (shown as the local cell repository in Figure 5), and (ii) a common domain hierarchy (CDH); this is shared by all cells (shown as the region repository in Figure 5). When multiple cells share the same CDH, these cells are called a *region*. The domains in each LDH are defined for describing format/meaning of data used in that cell while the domains in the CDH are defined as a set of standard data formats/meaning for data exchanging

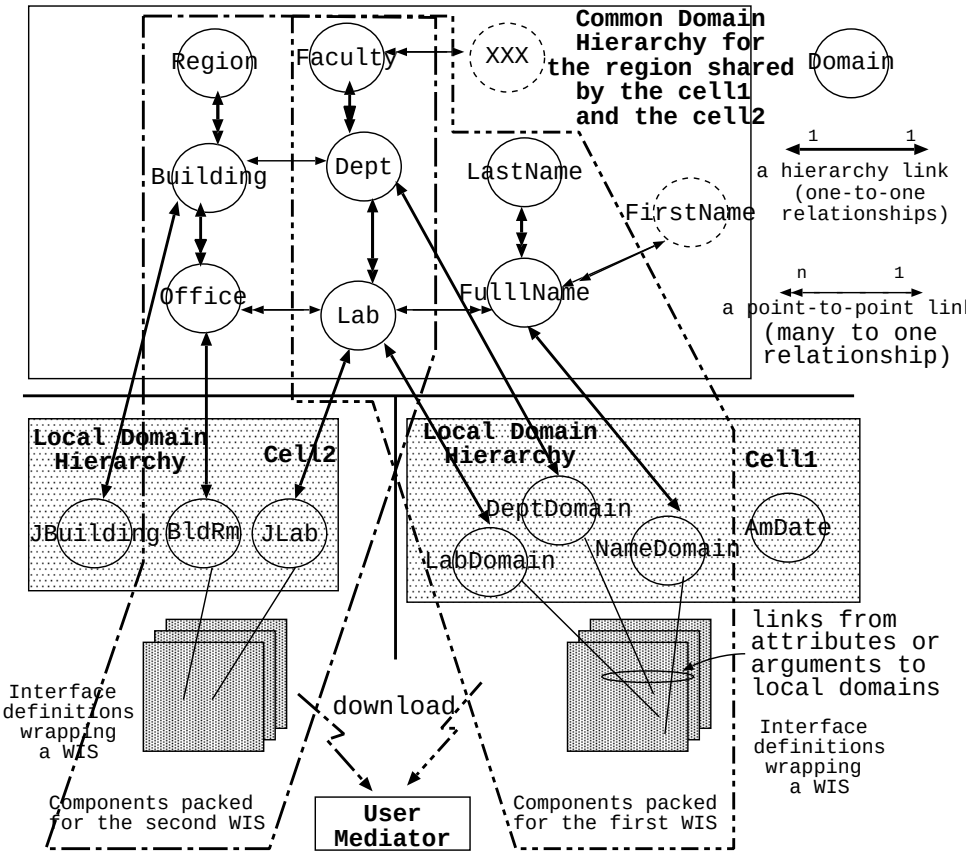

Figure 9: Domain hierarchy and downloading mechanism

among data of those cells.

As described earlier, all WISs are wrapped into a unified form called *an interface definition*. Attributes and arguments of access methods are mapped to the domains in the local domain hierarchy of the cell (see Figure 7). Moreover, domains of each LDH are linked to the domains in the CDH for providing semantic convertibility among data of WISs in different cells in the same region.

Construction: Now, we describe how to build a domain hierarchy. Domains are organized into a hierarchical structure, where a type of hierarchical relationship is a one-to-one relationship (data-format conversion), or an is-a relationship, or a part-of relationship. Therefore, for any given domain D, its parent domain is equal to, is more general than, or contains D. The *hierarchy links* (depicted by bold links in Figure 9) represent such hierarchical relationships.

As a result, for any given domain D, D has a hierarchical relationship with its parent. However, D may have other relationships with some domains D_i' $(i = 1, ..., n)$. These relationships are described as *point-to-point links* (depicted by fine links in Figure 9) between

D and D_i'. Therefore, D can have one hierarchy link to its parent and several point-to-point links to D_i'. (In implementation, each link is described by conversion functions for converting semantics between the two associated domains.) We refer to these links (with their conversion functions) as the *domain rule* of D.

Mediator and Script Generation: As described in Section 1, a mobile user must download software components describing services of WISs from their cells into his mobile computer. This *software package* of a WIS consists of (i) a front page of the WIS, (ii) interface definitions of the WIS, (iii) domain rules of the domains used in these interface definitions, (let us refer to these domains as X), and (iv) the domain rules of the ancestor domains of X. Therefore, as a result of downloading, a sub-part of the DH is maintained in the user's mobile computer. The download processing is shown in Figure 9. In this figure, the domains and interface definitions surrounded by the dotted line are packed and downloaded into the mobile computer.

The mediator on the user's computer uses the downloaded sub-part of the DH, and it resolves the semantic conflict between two data of different WISs. To do so, the mediator must find conversion paths between domains of the conflicting data, and choose the shortest one as a result.

In our scheme, a conversion path between two domains is assured to exist if there is: (i) a domain that is an ancestor of both domains; or (ii) only one point-to-point link between ancestors of both domains.[2] For example, in Figure 9, the conversion path from the domain *LabDomain* to the domain *JLab* is LabDomain $\rightarrow$ Lab $\rightarrow$ JLab. The conversion functions along the conversion path are inserted to convert the semantics between them.

Conversion paths are not always legal. If a conversion path causes a many-to-many relationship between a start domain and a destination domain, unexpected data will emerge. Such a conversion path is regarded illegal. The mediator must consider this restriction when finding a conversion path.

According to the above algorithms, the mediator in our system automatically resolves semantic conflict and generates an appropriate query. After that the mediator automatically embeds the query into the front page of the first WISs to generate a new client application for executing navigational integration, even in a communication-disconnected mode.

[2] In general, if a domain X has an ancestor who knows a point-to-point link to Z and if the same condition holds for another domain Y, then X and Y are reachable from each other.

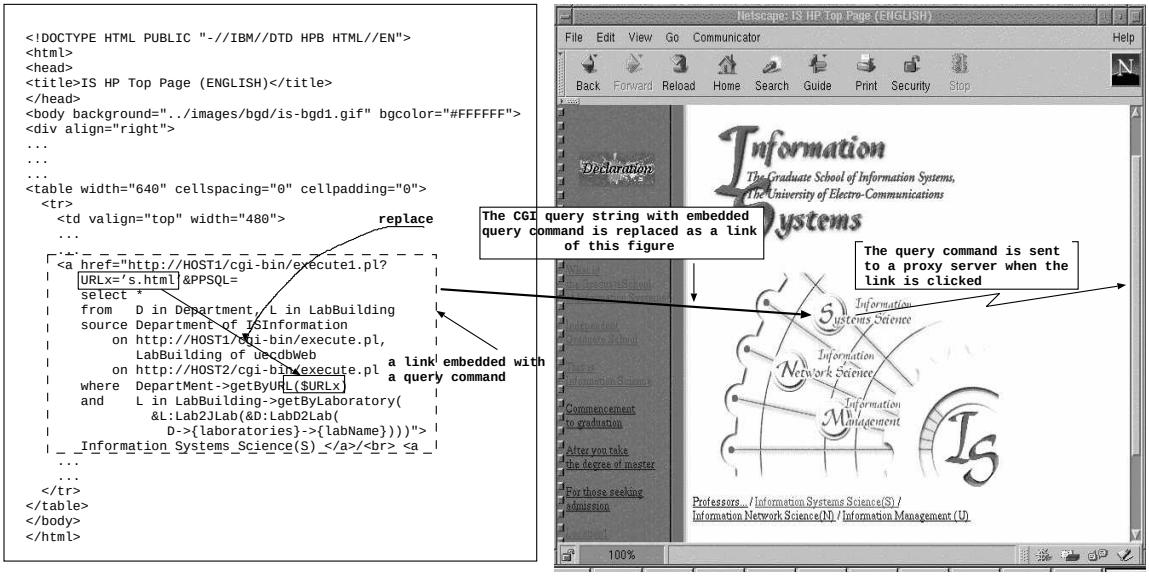

Figure 10: Code of the new client application (explained in Figure 2-c and Section 4.2)

We want to emphasize that all of these processing are performed by the interaction between the mediator and the user through the GUI shown in Figure 8. The interaction is simple. When the user selects the downloaded packages of WISs, the mediator presents a list of all path expressions and methods of them. To define navigational integration, the user simply specifies matching between path expressions of elements of one WIS and either arguments of methods or path expressions of elements of another WIS to form condition predicates described in Section 3.2. Thereafter, the mediator automatically resolves semantic conflicts and generates a new client application.

To understand the implementation of navigational integration more clearly, let us show an example of a new client application created by the mediator. Figure 10 is the source code of such a client-part and its browsed image; this is the client-part of the WIS shown in Figure 2-c, whose query command was explained in Section 4.2. As shown here, generally, the query command is embedded in the front page as a parameter of a query string of CGI or as a hidden field of a CGI form. The variables in the query command are bound by values of other parameters. Figure 11 shows the source code of the first result page of the client application (the middle page in Figure 2-c and that of the step 4 in Figure 5). Tags of derived links are generated by the proxy executor. The proxy executor analyzes the query command sent by the user into several sub-queries, sends one sub-query to the first WIS, receives the result pages from the WIS, and embeds CGI-execution tags to form derived links. These tags will be used to activate the next sub-query to proceed to the

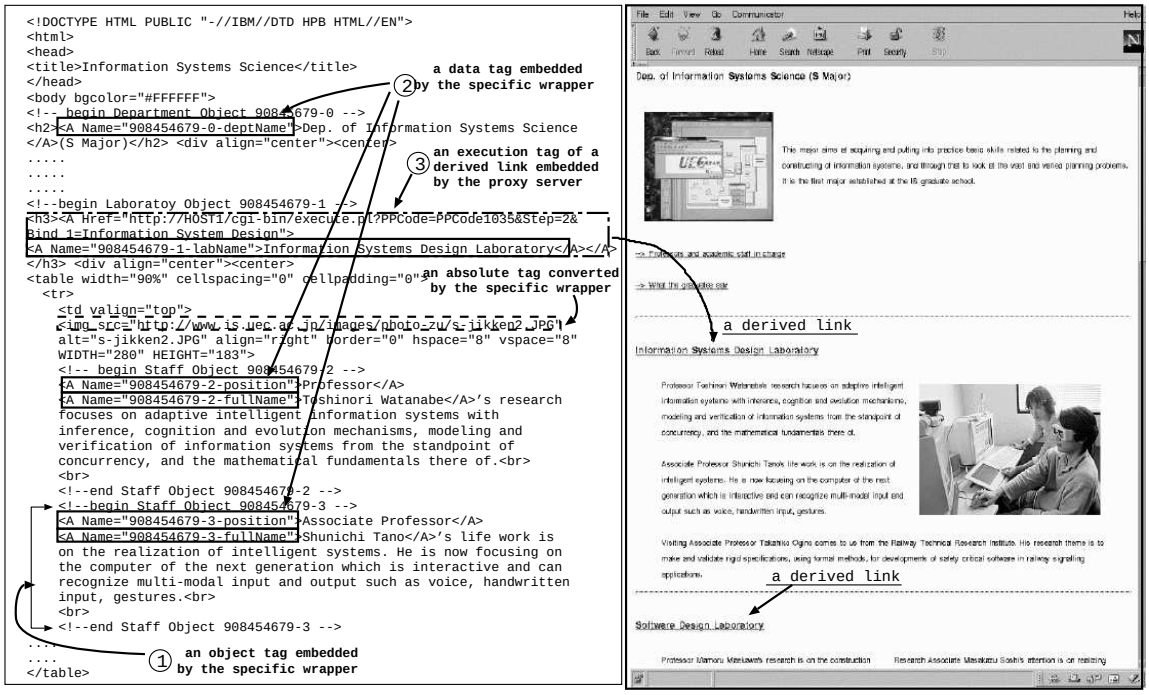

Figure 11: Code of the result page showing tags of derived links

next step of navigational integration.

6 Completions of Navigational Integration

So far, we have developed and implemented a system for mobile end-users for generating a new client application to materialize navigational integration. The common data model is used to represent data in documents in the Web. Path expressions and a query language are proposed for mobile end-users to described navigational integration between several WISs discussed in Section 4. Moreover, domain hierarchy for resolving conflict between autonomous WISs have also discussed in Section 5.

In this section, we present completions of navigational integration for WISs, where schemas of Web document including three properties:

(i) a heterogeneous domain: This is used for data that can be defined as string capable to represent in several domains. For example, contact address may be represented by e-mail in one place and by room number in another.

(ii) a heterogeneous structure: This is used for data that can be defined as alternative complex structure. For example, staff data may be represented by a teacher object in one place and by a visitor object in another.

(iii) a concatenation domain. This is used for data that is composed of several atomic

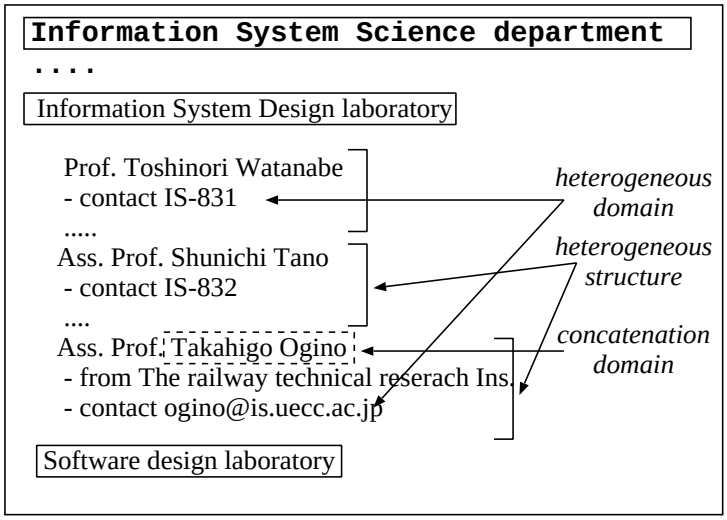

Figure 12: A Web document with heterogeneous properties

domains and requires to used it as one unit. For example, name is composed of first name and last name.

An example of document with the above properties is shown in Figure 12.

To support the effect of these properties, the enhancements of system in terms of common data model, path expressions and algorithms of semantic conflict resolution are discussed.

6.1 Enhancement of Common Data Model

To describe how to model data including the properties described above, let us start with how to define a domain of an attribute in an interface definition. The attribute declaration has the following form:

$$attributeName \qquad Domain.$$

The $attributeName$ is an attribute name defined in an interface definition and $Domain$ is a domain of $attributeName$. Let $AtomicDomain$ be a name of domain of atomic data and $IntfName$ be a name of an interface definition as described in the Section 4. In addition, Let the '$|$' symbol indicate an OR constructor and the ',' symbol indicate a sequence of appearance. For example, $A|B$ means that either A or B must appear. A, B means that A must appear, followed by B. To support the properties described above, we enhance the syntax of $Domain$ in such a way that $Domain$ is expressed by the following variations:

1. AtomicDomain

2. (AtomicDomain | AtomicDomain | ...)

3. (AtomicDomain, AtomicDomain, ...)

4. **setof** IntfName

5. **setof** (IntfName | IntfName | ...)

6. (AtomicDomain | AtomicDomain | ... | **setof** (IntfName | IntfName | ...))

The variations 1 and 4 are the simple domains for supporting structured documents described in Section 4. The variation 2 supports the heterogeneous domain property. The variation 5 supports the heterogeneous structure property. The variation 6 supports combination of the heterogeneous domain and the heterogeneous structure properties. The variation 3 supports the concatenation domain property.

6.2 Enhancement of Path Expressions

To allow domain specifying, we have enhanced path expressions. The enhanced path expression has the form

$$t_0 \rightarrow \{A_1\} \rightarrow ... \rightarrow \{A_n\} < Domain >,$$

Where t_0 is a variable representing an object instance of a top-level interface definition in the Web page.

For instance, if we want to specify the contact data whose domain is *EmailDomain*, we can specify as follows:

$$D \rightarrow \{laboratories\}[*] \rightarrow \{staffs\}[1] \rightarrow \{contact\} < EmailDomain >.$$

Path expressions for specifying data of sub-domain in a concatenation domain is the same as the path expressions of heterogeneous domain. For example, if we want to specify the first name of the first staff of all laboratories, we can specify as follows:

$$D \rightarrow \{laboratories\}[*] \rightarrow \{staffs\}[1] \rightarrow \{fullName\} < FirstName >.$$

The next step is to specify navigational integration in terms of path expressions in query conditions. To describe navigational integration, we let the mediator use a *restricted disjunctive normal form* (restricted DNF) [3] in the where clause of a query command.

Assume that a user wants to write a query command to define navigational integration from a WIS S to a WIS R. Then, we express the condition clause by the following restricted DNF style:

[3] We call the term *restricted DNF* because each sub-condition must begin with a method invocation and each derived link is generated when all statements in a sub-condition are valid.

$$((R \rightarrow methodOfR(pe_{S1}) \textbf{ and } (pe_{R11} = pe_{S11}) \textbf{ and } ...)$$
$$\textbf{or}$$
$$(R \rightarrow methodOfS(pe_{S2}) \textbf{ and } (pe_{R21} = pe_{S21}) \textbf{ and } ...)),$$

where pe_{Xi} ($X \in R, S$ and i is a number) is a path expression to specify an element (attribute) of a WIS X.

Remember here that the algorithm of semantic conflict resolution in Section 5 deals with only atomic domains. Therefore, as far as a user can decompose a query condition of navigational integration into *expressions on atomic domains*, through the GUI shown in Figure 8, the user can interact with the mediator to resolve semantic heterogeneity automatically for building such a query (of a restricted DNF expression).

6.3 Enhancement of Algorithm of Semantic Conflict Resolution

Earlier, we have described the enhancements of our common data model, path expressions and queries for navigational integration given in restricted DNF. However, such a query condition (which consists of expressions on atomic domains) will be sometimes too complex. A user must explicitly specify such a query, and it might be a difficult work. Thus, we require an ability that the mediator can generate a query condition of navigational integration by way of a short-cut form. To achieve this requirement, By interact with the user through the GUI, the mediator should use (in default) an enhanced algorithm of semantic conflict resolution among data of domains to handle the following cases: (i) heterogeneous domains, (ii) concatenation domains, and (iii) heterogeneous domains including atomic data and objects.

(1) Semantic Conflict Resolution for Heterogeneous Domains

Consider navigational integration from an element x of WIS S to another element y of WIS R where the domains of both x and y are heterogeneous domains. Our problem is to determine the most reasonable conversion paths between domains of both elements. Below, we present the scheme that is applicable to our problem.

Suppose that domains of x and y are A and B, respectively. Let us firstly consider the case that A and B are heterogeneous domains, where A consists of a set of atomic domains $(A_1|A_2|...|A_n)$ and B consists of a set of atomic domains $(B_1|B_2|...|B_m)$. Then, conversion paths from A to B can be computed as a set of best paths from $A_i(i = 1, ..., n)$ to $B_j(j = 1, .., m)$ for all values A_i and B_j as follows:

$$
\begin{array}{rcl}
P_{A_i,B_j} & = & \{a | a = path(A_i, B_j), \\
 & & i = 1, .., n \ and \ j = 1, ..., m\} \\
 & \text{and} & \\
ALLPATH(A, B) & = & \{b | b = bestPath(P_{A_i,B_j}), \\
 & & i = 1, .., n \ and \ j = 1, ..., m\},
\end{array}
$$

where a set of conversion paths (P_{A_i,B_j}) between A_i and B_j is computed by a binary function called *path()*. In addition, the best conversion path is computed by a unary function called *bestPath()*. After that, all paths will be used to create the conditions in the disjunctive normal form (or conditions). That is, the mediator tries to find the best conversion paths for all subdomains of A to the all subdomains of B.

Example: Let us consider conversion paths between two domains A and B shown in Figure 9, where

$$
\begin{array}{rcl}
A & = & BldRm|JBuilding, \\
B & = & LabDomain|DeptDomain.
\end{array}
$$

The best domain conversion paths from all subdomains of A to all subdomains of B can be computed as shown in below table.

From	To	Relationship	Length
JBuilding	LabDomain	one-to-many	4
JBuilding	DeptDomain	one-to-one	3
BldRm	LabDomain	many-to-one	3
BldRm	DeptDomain	many-to-one	4

In principle, for two given heterogeneous domains, the plausible conversion paths are the best paths of combination of all subdomains between them. All of them can be the answer.

However, from a practical viewpoint, it will be often convenient to simplify the answer. To do so, the mediator may filter the answer again by way of the priority and the length of the paths.

Roughly seeing table , all paths in this table are legal but the second path and the third path seem better than the others. In this example, the DeptDomain is selected as end-point domain in the conversion path of the domain JBuilding because it has stronger relationship (one-to-one) than the first path. In case of the BldRm, the LabDomain is selected because the conversion path is shorter than the fourth path.

The filter process is suitable when the receiver WIS is a CGI-form WIS. On the other hand, in case that the receiver WIS is an ordinary HTML document, we cannot predict the best paths between two heterogeneous domains because the best paths depend on not

only the relationships between subdomains but also their data. For example, some items of *Building* data in the sender may have no item of department data but has a lot of items of relevant laboratory data. In this case, all combinations of conversion paths are preferable to be chosen.

(2) Semantic Conflict Resolution for Concatenation Domains

The semantic conflict resolution for data of a concatenation domain can be regarded as an extension of string matching in conventional database system. The strategy to this problem is to find the conversion paths between sub-domains of a sender and those of a receiver.

Example: Let us consider a conversion path between elements x and y whose domains are A and B, respectively, and

$$\begin{aligned} A &= FName, LName \\ B &= JFName, JLName, \end{aligned}$$

where $FName$ and $JFName$ are first names written in English and Japanese, respectively. $LName$ and $JLName$ are last names written in the same manner. The domain conversion paths from all sub domains of A to all sub domains of B can be computed as follows:

$$\begin{aligned} &FName \rightarrow ... \rightarrow JFName \\ &LName \rightarrow ... \rightarrow JLName. \end{aligned}$$

The mediator tries to find conversion paths from all sub-domains of the sender to all sub-domains of the receiver. Thus, the conditions should be written as follows:

$$\begin{aligned} &R \rightarrow \{y\} < JFName > = f(S \rightarrow \{x\} < FName >) \\ &\textbf{and} \\ &R \rightarrow \{y\} < JLName > = g(S \rightarrow \{x\} < LName >), \end{aligned}$$

where $f()$ and $g()$ are conversion functions for resolving semantic conflicts.

Algorithm: Here we generalize the above example to an automatic procedure. Suppose that domains of x and y are A and B, respectively. In addition, A and B are concatenation types, where A is $(A_1, A_2, ..., A_n)$ and B is $(B_1, B_2, ..., B_m)$. The algorithm of semantic conflict resolution between two concatenation domains can be given by the following steps:

1. Create two lists called $listA$ and $listB$ for domains A and B, respectively. The $listA$ consists of $(A_1, A_2, ..., A_n)$ and the $listB$ consists of $(B_1, B_2, ..., B_m)$.

2. Until both lists get empty, do (a)-(c):

   (a) Start with the first element from $listA$.

   (b) Match with the domain(s) in $listB$.

   (c) If a conversion path is found, insert the pair of matched domains into a result list, and remove the matched domains from $listA$ and $listB$. If any conversion path is not found, the execution fails, and exit with error.

3. Generate a restricted DNF condition from the result list.

(3) Semantic Conflict Resolution for Heterogeneous Domain Including Atomic Data and Objects

Next, we consider the semantic conflict resolution between domains of two attributes where the domain of each attribute is declared as the union of an atomic domain and a structure type domain. Suppose that a domain of A is declared as $(J \mid \textbf{setof}\ K)$ and a domain of B is declared as $(M \mid \textbf{setof}\ N)$. J and M are atomic domains. K and N are names of interface definitions. In this case, the mediator automatically resolves all semantic conflicts by matching the atomic subdomain of A to the atomic subdomain of B as follows:

$$I1 \rightarrow ... \rightarrow \{A\} < J >= f(I2 \rightarrow ... \rightarrow \{B\} < M >),$$

where $I1$ and $I2$ are the variables of objects of a top interface names that lead to the attributes A and B, respectively. The mediator does not resolve semantic conflict between two heterogeneous structures.

In case that users want to describe navigational integration through the heterogeneous structures, we assume that the users must indicate what pairs of domains they want to match. As a result of this indication, the query condition is given by

$$I1 \rightarrow ... \rightarrow \{A\} < J > \ = \ f(I2 \rightarrow ... \rightarrow \{B\} < M >)$$
$$\textbf{or}$$
$$I1 \rightarrow ... \rightarrow \{A\}[*] \rightarrow \{P\} = g(I2 \rightarrow ... \rightarrow \{B\}[*] \rightarrow \{Q\}).$$

Note that the first condition clause is automatically generated by the mediator when the user specifies to match the attribute A to the attribute B whose domain is the atomic domain M. In contrast, the second condition clause is generated when the user explicitly specifies to match the attribute P of the sub-object in the attribute A to the the attribute Q of the sub-object in the attribute B. The function $f()$ and $g()$ are conversion functions for resolving semantic conflicts.

7 Concluding Remarks

In this paper, we have proposed navigational integration as a new style of integration for mobile users in the WWW framework. The navigational integration is a process to embed invocations of services functions of one WIS as derived links into a result page of another WIS. We have also proposed the system architecture consisting of wrappers, proxy executors and mediators for supporting the navigational integration. The mediator in our approach is installed in a mobile computer, performing functions of defining integration, resolving semantic resolution and generating a new client application to materialize navigational integration. The proxy executor is installed in a remote server, cooperate with wrappers to perform execution of requests sent from mobile end-users.

For resolving semantic conflict among autonomous WISs, a distributed repository mechanism called *domain hierarchy* is proposed. Our common data model allows us to represent Web data including the heterogeneous properties and to relate the data with the domain hierarchy. Algorithm for downloading necessary and minimum information from cells has been proposed. The algorithm that is a modification of shortest paths computations for resolving semantic conflict has been also proposed. Then, mobile end-users use the mediators on their mobile computers to exploit downloaded information to generate new client applications for executing navigational integration. We have also developed a prototype of our system for various WISs, including WWW document servers, CGI with database servers, and clickable maps.

We should emphasize that our approach can be generally applied to existing integration systems. The domain hierarchy concept is added to provide knowledge of data conversion between autonomous information sources while mediators on end-users' side allow end-users to be able to define their integration.

For future works, we are extending our system to support multi-level result pages[4] . Moreover, we are considering the way to filter Web contents by using a hierarchical structure and tag information of objects in the contents of the Web pages. We believe that the presented work in a promising first step in the direction for mobile end-users to integrate information from heterogenous WISs in the open-access WWW environment.

[4] The term multi-level result pages refers to the consequence of result pages that are generated by WWW server

References

[1] A. Kemper and G. Moerkotte. "Object-oriented database management: applications in engineering and computer science." Prentice Hall Publishers, 1994.

[2] D. Florescu, A. Levy and A. Mendelzon: "Database Techniques for the World-Wide Web: Survey". *SIGMOD Record*, 27(3): pp. 59-74, September 1998.

[3] D. Konopnicki and O. Shmueli. "Early experiences with W3QS - A WWW Information Gathering System." In *the 19th IEEE Convention of Electrical and Electronics Engineers in Israel*, 1996.

[4] E. Sciore, M. Siegel, and A. Rosenthal. "Using semantic values to facilitate interoperablity among heterogeneous information systems." In *ACM Transactions on Database Systems*, 19(2), pp. 254–290, June 1994.

[5] G. Arocena, A. Mendelzon, "WebOQL: Restructuring Documents, Databases and Webs" , In *Proc. of 14th. Intl. Conf. on Data Engineering (ICDE 98)*, Florida, 1998.

[6] G. O. Arocena and A. O. Mendelzon: "Applications of a Web Query Language." In *Proceedings of the 6th International WWW Conference*, Santa Clara, California, April 1997.

[7] M. Fernandez, D. Florescu, A. Levy, D. Suciu.: "A Query Language for a Web Site Management System", *SIGMOD Record*, 26(3): pp. 4-11, September 1997.

[8] M. Kifer, W. Kim, Y. Sagiv. "Query Object-Oriented Databases", In *Proc. of the ACM SIGMOD Conf*, pp. 393-402, California, USA, 1992.

[9] M. Tork Roth and P. Schwarz.: "Don't scrap it, wrap it! a wrapper architecture for legacy sources", In *Proceeding of the 23th VLDB Conference*, Athens, pp. 266-275, Greece, 1997.

[10] M.W. Bright, A.R. Hurson and S. Pakzad. "Automated Resolution of Semantic Heterogeneity in Multidatabases." In *ACM Transaction on Database Systems*, Vol. 19 No. 2, pp. 212–253, June 1994.

[11] P. Atzeni, G. Mecca, P. Merialdo, "To Weave the Web", In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB'97)*, Athens, Greece, pp. 206-215, August 1997.

[12] S. Chawathe et al. "The TSIMMIS project: Integration of heterogeneous information sources." In *Tech. Rep of 100th DBS of Information Processing Society of Japan*, pp. 7-18, Tokyo, Japan, 1994.

[13] S. Cluet et al. "Your mediator needs data conversion!." In *Proc. of the ACM SIGMOD Conf. on Management of Data*, Washington, USA, pp. 177–188, 1997.

[14] T. Imielinski and B. R. Badrinath.: "Data Management for Mobile Computing." In *SIGMOD Record,* 22(1): pp. 34–39, March 1993.

[15] V.S. Subrahmanian et al.: HERMES: A Heterogeneous Reasoning and Mediator System, available at http://www.cs.umd.edu/projects/hermes/publications/authors/all.html

[16] W.Kim, I. Choi, S. Gala, and M. Scheevel. "On resolving schematic heterogeneity in multidatabase systems." In *Distributed and Parallel Databases* 1(3): pp. 251–279, July 1993.

[17] W. Sae-Tung. "Integration of Autonomous Web Information Sources by Mobile Users". *Ph.D. Thesis*, The University of Electro-Communications, 1999.

[18] W. Sae-Tung, T. Ohmori, and M. Hoshi. "Navigational Integration of Autonomous Web Information Sources by Mobile Users" In *1999 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM 1999)*, Victoria, B.C, Canada, pp. 270–275, August 1999.