Chapter 1

Introduction

1.1 Background and Motivation

In today's computer internetworks, most organizations use the World-Wide-Web (WWW) [30] as a standard media for providing their information to public users. In many cases, information services are provided through the WWW, they are represented in form of hypertext documents that range from simple static documents to dynamic documents created by backend database servers. For example, ordinary HTML document servers, CGI-forms text with backend database servers, search engines, and java-based clickable maps with backend data services can be found everywhere in the Internet. Even though most of information sources provide users with significant value when used in isolation, there is increasing need to integrate information and services of the multiple WWW systems in the public networks to provide more valuable information and services.

To integrate such information sources, many groups of academic and commercial researches have developed several integration systems. Most of them have proposed the solution for integrating fixed information sources on a centralized site. An *integrated view* over several information sources is defined by developers. Therefore, they restrict end-users to access only the static (and relational-like) database views defined in the centralized system. For example, data from several information sources such as per-

sonal bibliography and relational databases are collected, and are defined to generate a result information table to external world. End-users can only query information from these information sources through the static integrated view defined in the centralized site. This approach may appear to be adequate for use. However, no systems have proposed an appropriate way for mobile users to integrate information among autonomous information sources. For example, a user carrying his mobile computer enters into a place and meets multiple information sources that provide relevant information. Then, the first problem is how the user can integrate these information sources of the autonomous distributed organizations by himself. The second problem is what kind of the result of integration is suitable to the WWW framework and easy for the end-users to use and understand. These two problems have not been addressed by existing systems, which concentrate on providing tools for developers to create a centralized integration system.

Motivated by this background, this thesis proposes a new information-integration system for mobile users in the WWW environment. Our goal[37, 38, 39, 40] is to realize an environment where mobile users move across networks, find/download some necessary information of information sources from different autonomous distributed organizations, and define an integration of information/services from the WWWs on their mobile computers in a disconnected mode. For the WWW system based on the hyperlink concept, we propose a new style of integration of such Web-based information sources for mobile users. The integration in our framework is the process to embed *derived links* into Web pages of one information source for calling services of another information source. In addition, defining integration on the mobile computers in a disconnected mode saves the battery power of mobile computers and communication

cost in the mobile environment[31].

1.2 An Example of Integration of Our Approach

To understand the integration of our approach more clearly, in this section, we will give an example illustrating the results of integration, and an overview of system environment.

In the WWW system, client applications of the information sources are embedded into Web pages. To access information from the information sources, users use browsers to download these Web pages and execute these client applications. Throughout this thesis, we use the term *a Web Information Source* (WIS) to refer to such an information source providing its services through the WWW system. *A Web Client Application* (WCA) refers to a client application of a WIS embedded into a Web page.

Consider WISs in a university that provide information about the university. Figure 1.1 shows WISs that can be found in the university: (i) an ordinary HTML document server providing department information, (ii) a CGI-form text with a backend database server providing information of laboratories, (iii) a search engine of the university, and (iv) a java-based clickable map with a backend database service. Mobile users can access information from these WISs by downloading their front pages consisting of the client applications of the WISs. The front page includes links to access its result pages. Thus, these links can be regarded as methods that invoke the database server of the WIS to access the result pages. Suppose that a user enters into the university, meets these WISs, and wants to integrate information between two WISs. The integration in our framework is the process of embedding those derived links which invoke methods of the database server of the second WIS into the result pages of the first WIS. Figure

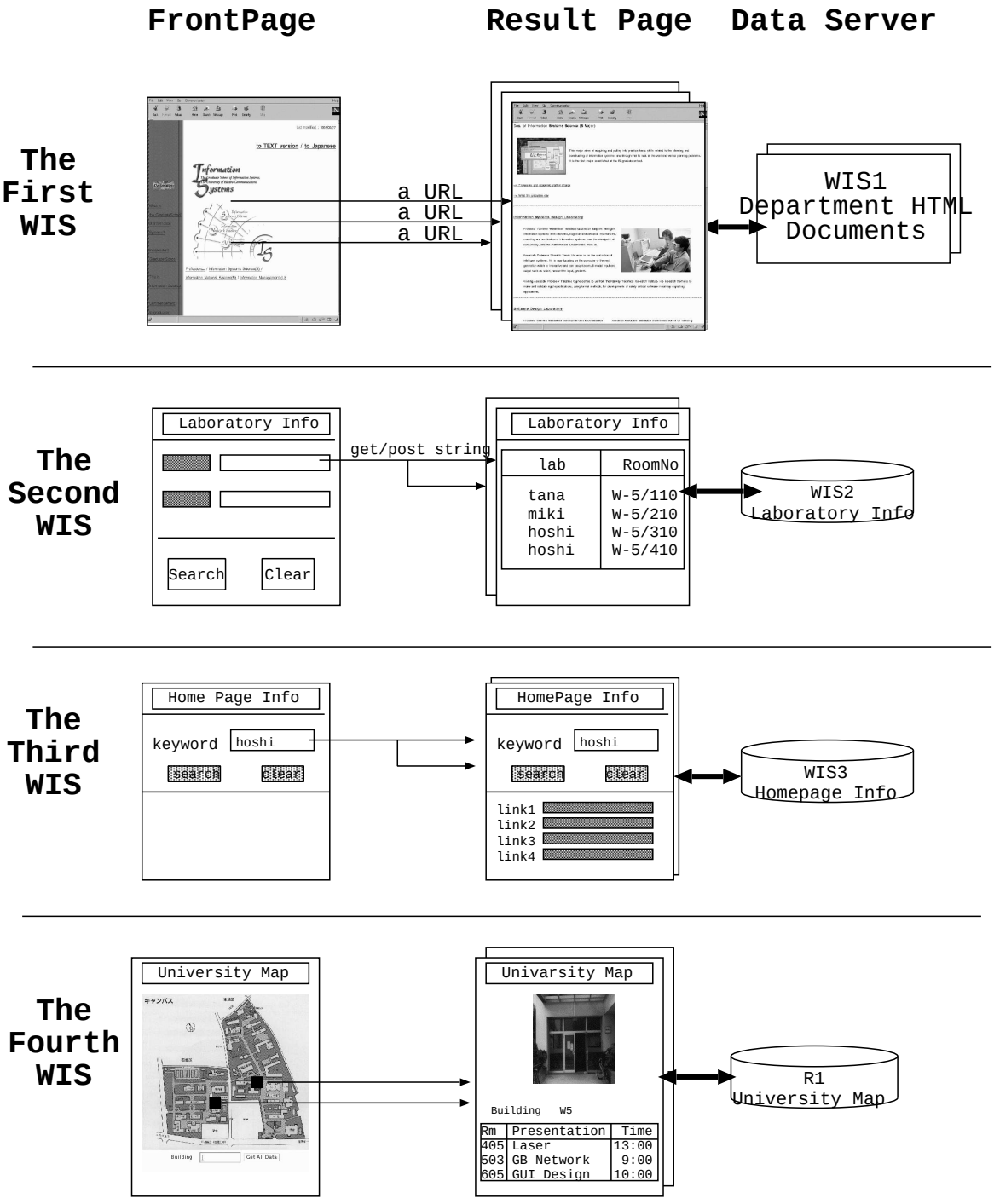


Figure 1.1: WISs in a university

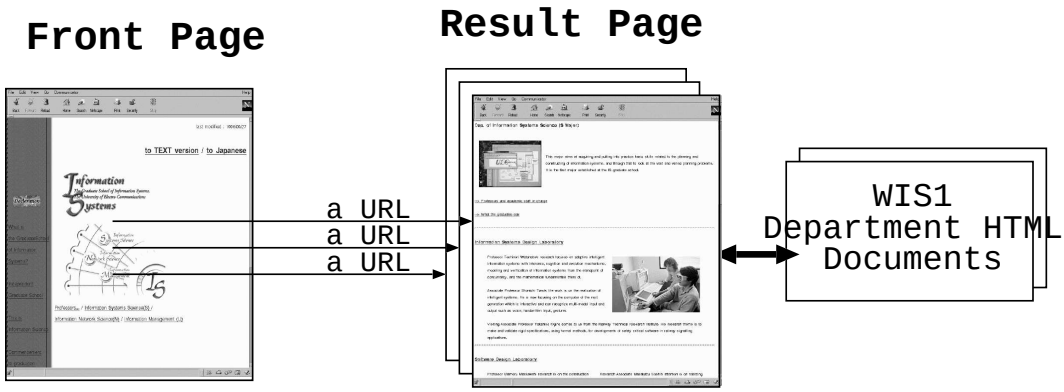
1.2 shows an example of integration of our framework. Figure 1.2-(a) shows the first WIS that is an ordinary HTML document server providing department information in the university. Figure 1.2-(b) shows the second WIS that is a CGI-form text with a database server. The user can input conditions into the form to query information of laboratories in the university. Then, Figure 1.2-(c) shows the result of the integration of our framework. The derived links that are used to invoke the server-access method (method of CGI program) of the second WIS are embedded into the result page of the first WIS. We call such an integration of WISs *a Navigational Integration.*

Unlike the traditional integration systems, the navigational integration has several advantages shown as follows:

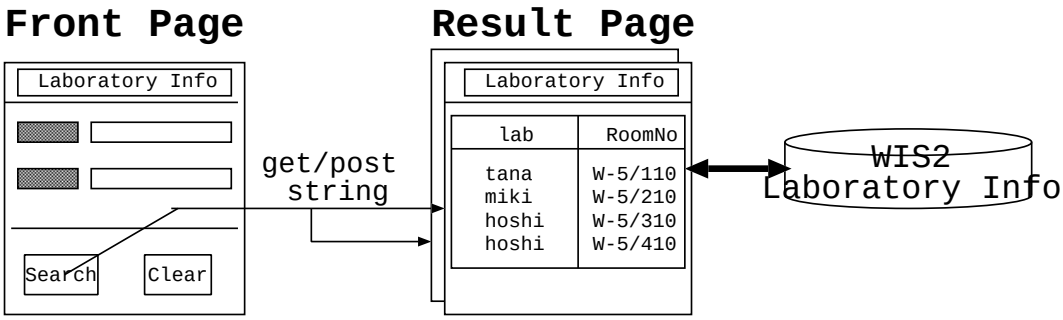
- The navigational integration is the process of embedding service invocations of a WIS as derived links into the result page of another WIS. From the users' view points, it is a natural manner in the WWW framework.

- When applied to WISs that provide filtering capability, the navigational integration can utilize the benefit of the data filtering of the WISs. Users can retrieve, process, filter information at the WISs, and receive only the relevant information. This operation not only retrieves only required information but also reduces a volume of data transmitted between mobile computers and WISs.

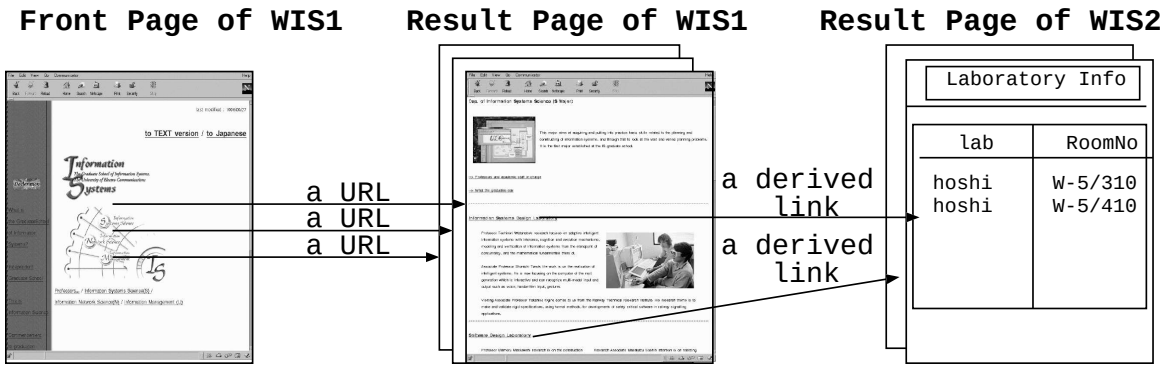
- The navigational integration framework also provides semantic conflict resolution between different WISs. At least, such a heterogeneity must be resolved automatically. This ability is necessary for a system where end-users need not to recognize such knowledges of the system[45]. Detail of semantic conflict resolution will be described in chapter 5.



(a) The First WIS (WIS1)



(b) The second WIS (WIS2)



Service invocation of the WIS2 is embedded as a derived link in the result page of the WIS1

(c) Navigational Integration (WIS3)

Figure 1.2: Example of navigational integration

This thesis proposes a system architecture that allow mobile end-users to be able to define navigational integration among WISs on their mobile computers. Our assumed environment is shown in Figure 1.3. In general, these WISs are maintained in multiple autonomous organizations. The WISs are independently implemented and connected to the network for providing information and services to public users. We call such an organization *a cell*. In the above example, a cell may be a department or a laboratory located in a building in the university. In addition, a network is assumed to be made of many cells. We call such a network *a region*. Each cell has a host server as a yellow page server providing an access to its maintained WISs. We call such a host server *a Main Information Resource* (MIR). In this environment, we propose a scenario where mobile users move across cells in a region and frequently disconnect computers from the networks as shown in Figure 1.3. While moving, users connect their computers to the networks, and download necessary information from MIRs of cells. As a result, to reduce the communication between mobile computers and the MIRs, we aim to realize an environment where mobile users can generate new client applications from the downloaded information in order to integrate those WISs on their mobile computers while they are in a disconnected mode.

1.3 Technical Problems and Principle of Our Solutions

To realize the described framework to support mobile users for integrating autonomous WISs, the following technical problems must be solved:

- *Formalization of navigational integration*: Unlike the integration of existing integration systems, the navigational integration concentrates on how to embed

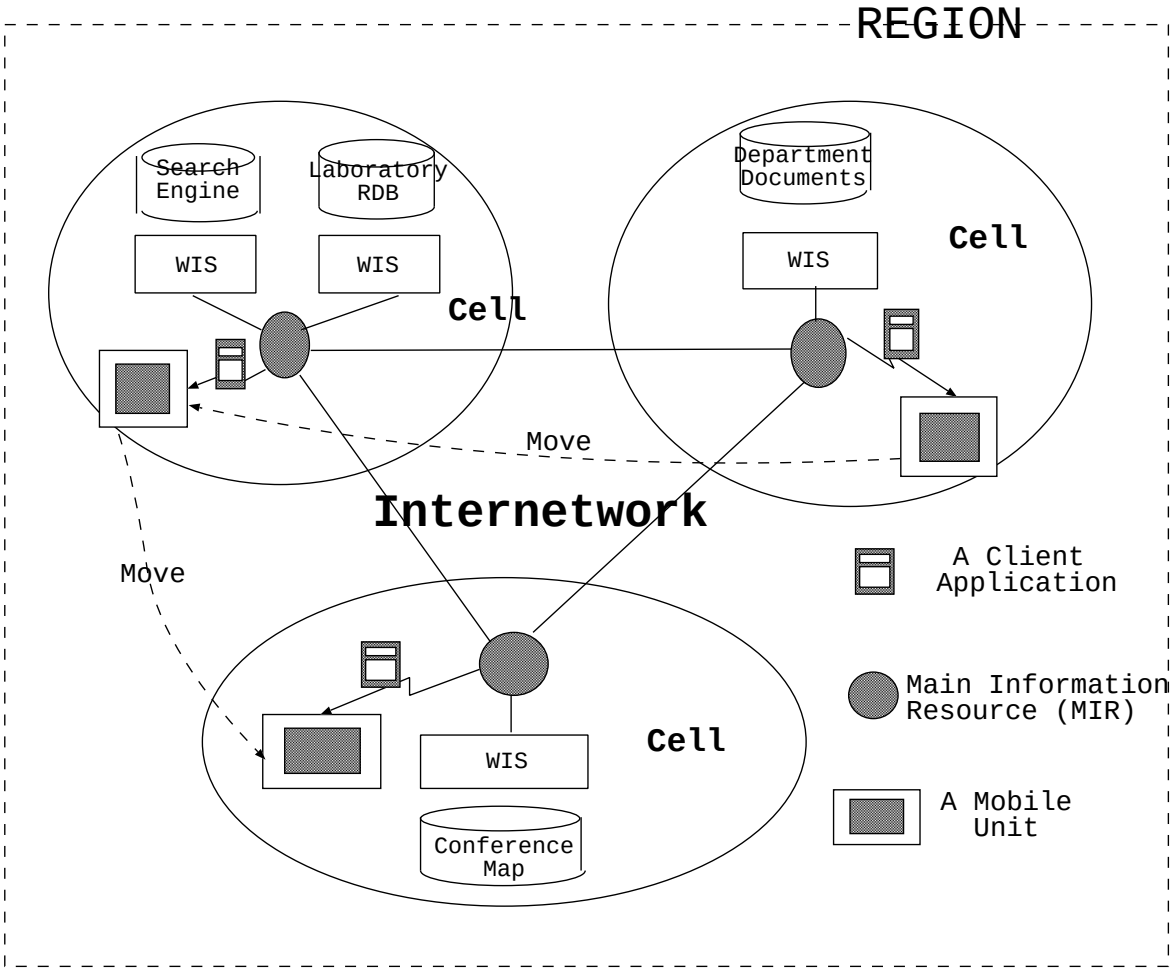


Figure 1.3: Web Information Sources in a mobile environment

access methods of one WIS into Web pages of another WIS. The traditional database integration concentrates on defining views on data by programmers. Data are extracted from existing sources, and then they are restructured and finally represented into pre-defined forms. This concept cannot be applied directly because the navigational integration is based on invoking access methods of WISs using the concept of hypertext links. Moreover, the integration is defined by a mobile user, not by a programmer. The process of defining integration must be easy to understand and use.

- *Semantic heterogeneity*: The term *semantic heterogeneity[8]* refers to the fact that the same data may be represented in different forms in different systems. Similar to the existing integration systems, data from one system must be converted into the different form used in other systems. However, the existing solutions are performed in an ad-hoc manner through static descriptive rules defined by programmers. In contrast, the navigational integration is defined by mobile users. Thus, the automated semantic conflict resolution is required.

- *Mobile environment*: As described earlier, mobile users often disconnect their mobile computers from the networks due to power battery saving and communication cost. The requirement is to provide an ability for mobile users to define integration on their mobile computers in a disconnected mode.

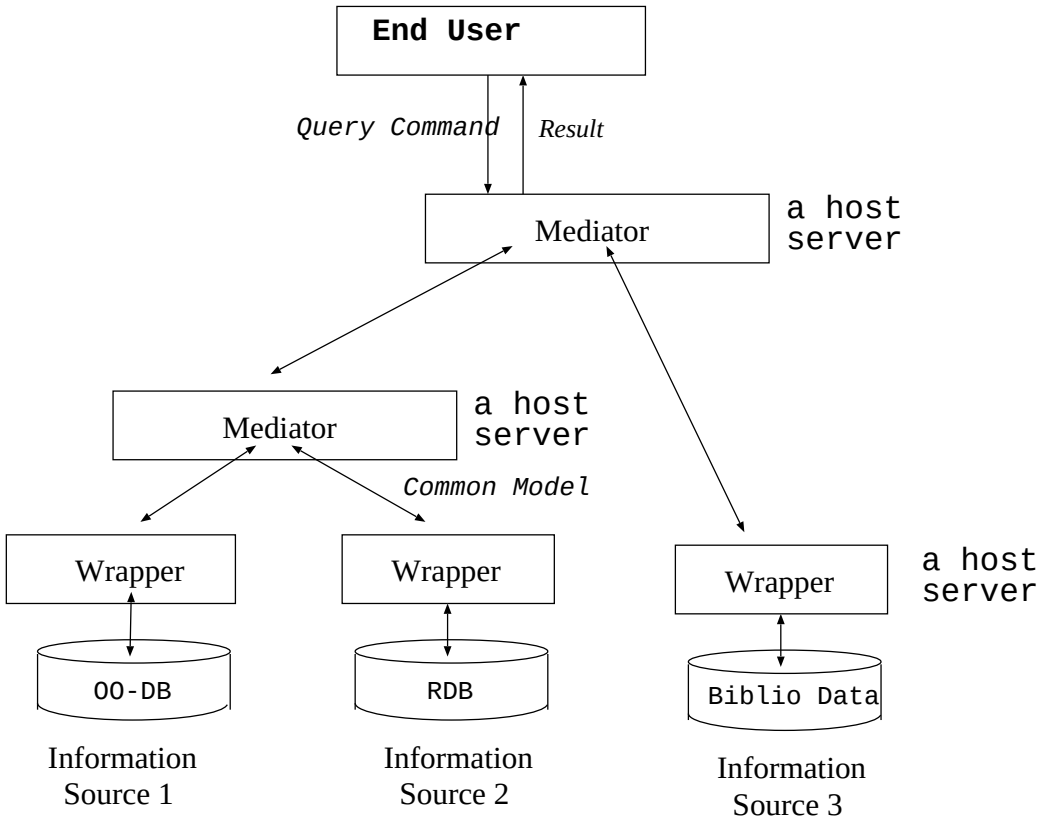
Several works [14, 19, 27, 28] have proposed the frameworks for restructuring and integrating information from *fixed information sources* and providing views as *static integrated information* via middle layer sources. The hierarchy layer architecture containing *wrapper*[42] and *mediator*[41, 43] modules is an outstanding approach proposed by many researches. TSIMMIS[27] project is one of the well-known researches. In

this framework, the *wrapper* extracts data from the original information source and converts them into data in a common data model. In addition, the *mediator* combines, integrates or refines data from wrappers, providing users with a more desired view. These frameworks propose the development environment for developers to create a centralized integration system. Therefore, it restricts end-users to access only the static views defined in the centralized system. Users in this approach have no freedom to select information sources in the distributed and mobile environment.

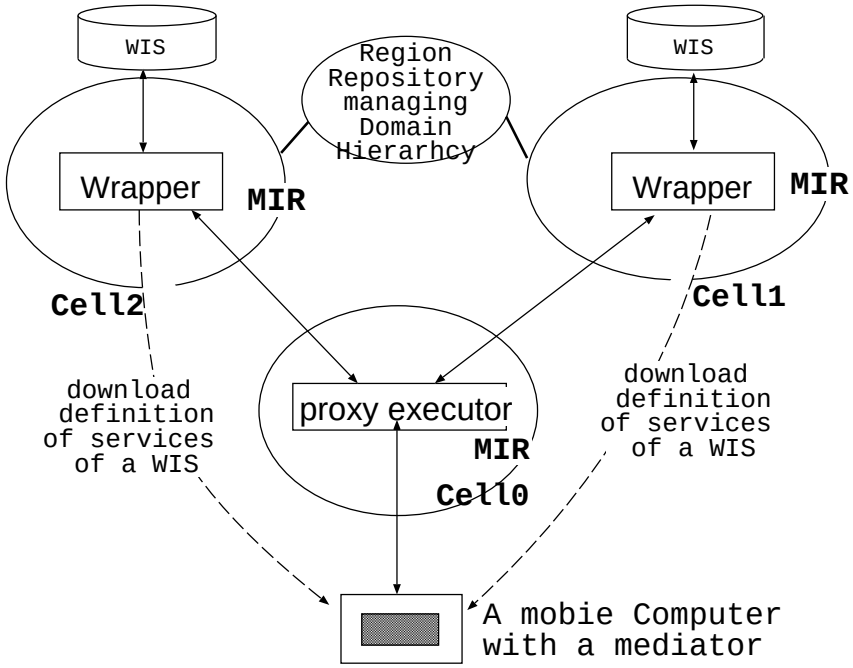
In contrast to these frameworks, our framework provides mobile users with a mechanism for integrating autonomous WISs on their mobile computers in a disconnected mode. The traditional approach cannot be applied directly and needs some modifications. A new solution requires rearranging the system architecture of the existing integration systems. It must also include some new enhancements to solve the restriction in mobile environment and the heterogeneity in autonomous distributed organizations.

Figure 1.4 illustrates the difference between the system architecture of the existing approach and that of our approach. In the exiting approach shown in Figure 1.4-(a), each information source is wrapped by a module called *wrapper*. The wrapper extracts data from its information source and converts them into data in a common data model. An integrated view among multiple information sources is defined in a host server that has a module called a *mediator*. Even though each module can be installed in any host server, end-users can retrieve information through only the mediator. End-users cannot combine these information sources by themselves.

In contrast, the system architecture of our approach (shown in Figure 1.4-(b)) allows mobile users to select the target information sources. Each cell shares the common knowledge called a *domain hierarchy* for automatically resolving semantic conflict of



(a) A system architecture of the existing approach



(b) A system architecture of our approach

Figure 1.4: Comparison of system architectures

data between cells. Each cell has a main information resource (MIR) as a yellow-page server and still places the wrapper module on the MIR. Rather than placing the mediator module on the MIR, we place the mediator module on the mobile computer side. The mediator downloads some necessary information of WISs from MIRs into the mobile computer, and it helps a user to define navigational integration. Thus, the mobile user can choose the WISs by themselves. The mediator of the traditional approach also performs the execution of query commands sent from end-users. In contrast, our approach places a module called a *proxy executor* for performing this function on the MIR. The navigational integration can be performed by sending a query command through this proxy executor.

1.4 Points of Originality

Based on the solution principles described in section 1.3, the points of originality of this work can be listed below:

- A new style of information integration for the WWW framework called *navigation integration* [38, 39]: This is a new style of generating a new WCA to integrate information from several WISs for mobile users.

- A new distributed repository called a *domain hierarchy*: Unlike the centralized repository of the existing integrations systems, semantic conflict among information provided by independent organizations can be resolved automatically while each organization still has the right of self-management.

- Downloading algorithm for acquiring necessary information from information sources: Using this algorithm, the only necessary information can be downloaded and be maintained in the users' mobile computers so that the mobile users can

use the mediators on their mobile computers to generate new customized client applications for integrating several WISs in the disconnected mode.

1.5 Organization of Thesis

This thesis is organized as follows. **Chapter 2** reviews related works in query languages for Web, integration systems based on a mediator/wrapper model, and Web site management systems. This chapter describes details of common data models and important modules in existing integration systems.

Chapter 3 firstly describes general requirements for achieving our goal. Thereafter, this chapter describes the basic idea of how to implement navigational integration. This is followed by a component model of a WIS to explain the requirement of the WIS in this framework. We then outline an integration model, an execution process of navigational integration and several components of our system architecture.

Chapter 4 begins with a common data model to support structured documents whose structures can be described as DTD (Document Tag Definition) without "or" structure. Then, it is followed by path expressions for specifying data and methods on the basis of our common data model. A general definition of navigational integration is described based on the common data model and the path expressions. To define the navigational integration by mobile users, a query language in terms of the path expressions is proposed. After that, this chapter defines functions of components on a MIR that execute the query command for performing navigational integration. Overall behavior and cooperation of the components are given in the steps of processing procedures.

Chapter 5 presents details on a new approach called a *domain hierarchy*; it is an autonomous and distributed domain hierarchy management system for resolving semantic conflict of information among independently-developed WISs. Based on the domain hierarchy, this chapter proceeds with a download algorithm for packing necessary information of WISs into mobile computers. Using the downloaded information, an algorithm for resolving semantic conflict is proposed. With this algorithm, the mediator can resolve semantic conflict between different WISs of autonomous cells in a disconnected mode. In addition, this chapter explains how to define domains for general Web documents, and then describes optimized execution plans determined by the mediator.

Chapter 6 describes the system prototype of our approach. Firstly, we start with the prototype configuration that we implemented. Secondly, domain hierarchy of a region for a university is implemented. Thirdly, an applicability of system is described. It includes the range of WISs to which our approach can be applied and the efficiency and effectiveness of the proxy server. Finally, efficiency and effectiveness of the mediator is discussed.

Chapter 7 describes enhancements for supporting semi-structured data. This chapter describes the particular properties of semi-structured data that do not appear in the well-structured data described in chapter 4. To support such properties, this chapter describes the enhancements in the following issues: (i) the common data model, (ii) the path expressions, and (iii) algorithm of semantic conflict resolution.

Finally, **chapter 8** concludes this thesis.

Chapter 2

Related Works

Recently, the WWW has become one important data repository [30], providing access to sources whose data can be represented in different forms: structured data, semi-structured data, or unstructured data. Therefore, the database technology[3] to the problems of querying and managing large volume of data has been applied to solve the problems in the WWW framework[5]. This chapter reviews significant bodies of recent researches using the database technology to address the information integration and management on the WWW framework. We will focus on the following works: (i) query languages for the Web, (ii) heterogeneous information integration systems, and (iii) Web site management systems.

2.1 Query Languages for the Web

Query languages [6, 10] are developed to apply SQL-like queries to Web document. At the first time, many query languages propose to describe the Web with different data models. For example, WebSQL [10] proposes to model the Web using the relational data model [4]. Web3QL [6] uses the object-relational data model to describe the Web so that it can use external programs similar to users-defined functions. These languages treat Web pages as atomic objects. We can see that the query languages for the Web

adopt the well-known data models, e.g., the relational model, the object-relational model or the object-oriented model, as their common data models to handle data in the Web pages and provide simple data manipulations on them. However, they do not distinguish between internal structure and the external links. Moreover, they do not provide the ability to create new complex structures as a result of a query.

Recently, new query languages for the Web are proposed to solve the above problems. WebOQL [9] is one of such languages. The data model in WebOQL is the hyper-tree. Hyper-trees are ordered arc-labeled trees with two types of arcs, internal and external. Internal arcs are used to represent structured objects and external arcs are used to represent references (typically hyper-links) among objects. Figure 2.1, from [9], shows a hyper-tree describing a Web document called *csPapers* that contains descriptions of publications from several research groups.

The WebOQL is a functional language. There are many operators to process hyper-tree so that it can be used as a tool to filtering and creating Web data. The detail can be found in [9].

However, the query languages for Web (WebSQL, Web3QL, WebOQL and so on) do not pay attention to the data in the internal structure. They pay attention to tags of HTML documents. Though the WebOQL provides ability for filtering and restructuring data in Web document, it is too difficult for end-users to understand and use. That is, it is suitable to be used as programming tools. For end-users, a simpler query language is necessary.

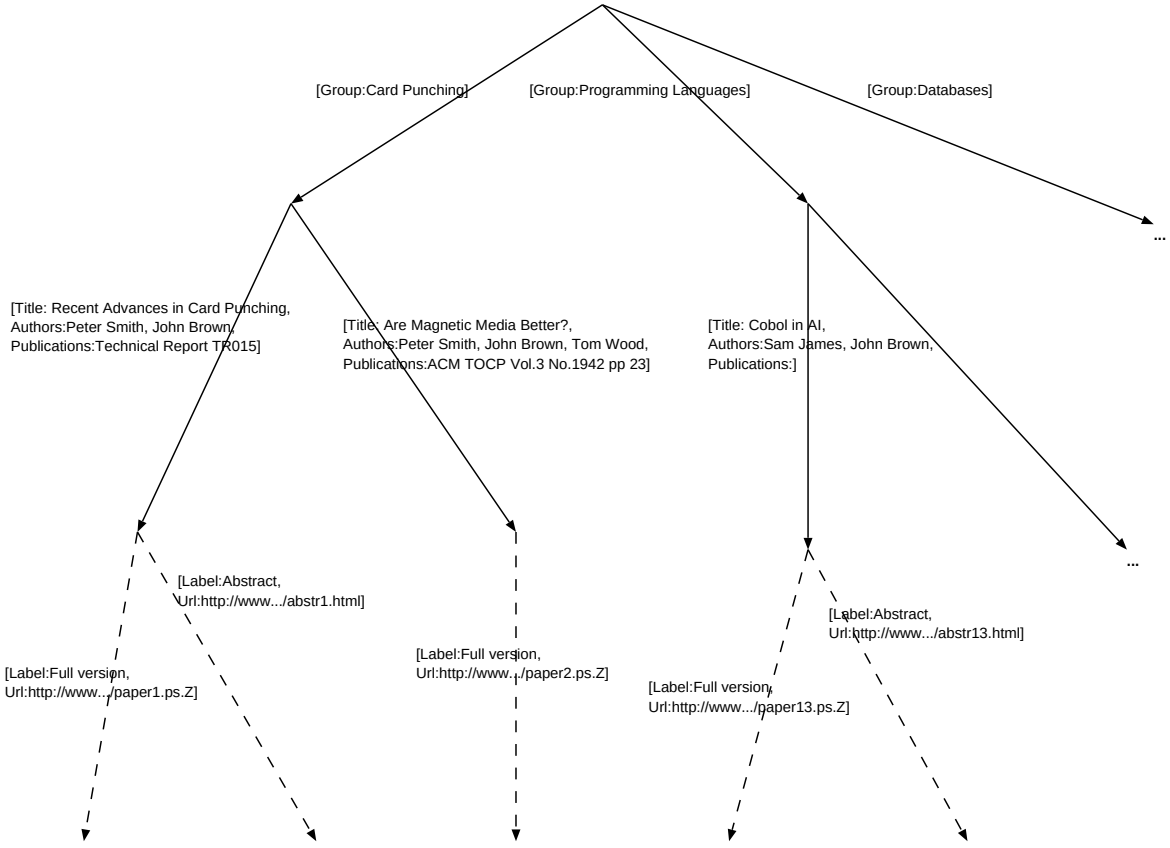


Figure 2.1: A sample of hypertree

2.2 Heterogeneous Information Integration

In the database research, several systems[11, 16, 19, 20, 26, 27, 34] have been proposed for integrating heterogeneous information sources such as relational databases, object-oriented databases, structured files, and so on. In particular, WWWs contain a large number of information sources and grow to be one of important information sources. All of these information sources represent their information in form of structured/semi-structured documents. Thus, information integration systems are also extended for supporting structured/semi-structured documents.

One problem that make information integration difficult is the heterogeneity of data among different information sources. To solve this problem, one solution is to build a front-end system that supports a single common data model, and proposes either a single query language or a user interface on top of the different types of information sources. The common system architecture of this approach to support the concept of the common data model and the user interface is shown in Figure 2.2. Unlike conventional database systems, there are two main modules in this architecture:

Wrapper: Because the integration system cannot directly communicate with a local system of each information source, a wrapper is proposed to do such a work. It translates a common query command [7] to a native command of each information source and translates the replied data into the common data model that can be further processed by the integration system.

Mediator: Because the goal of integration in this approach is to help users access information without knowing where the information is stored, the mediator provides a global view of information that is integrated from several sources. It receives a user's query, splits it into several sub-queries, sends the sub-queries to

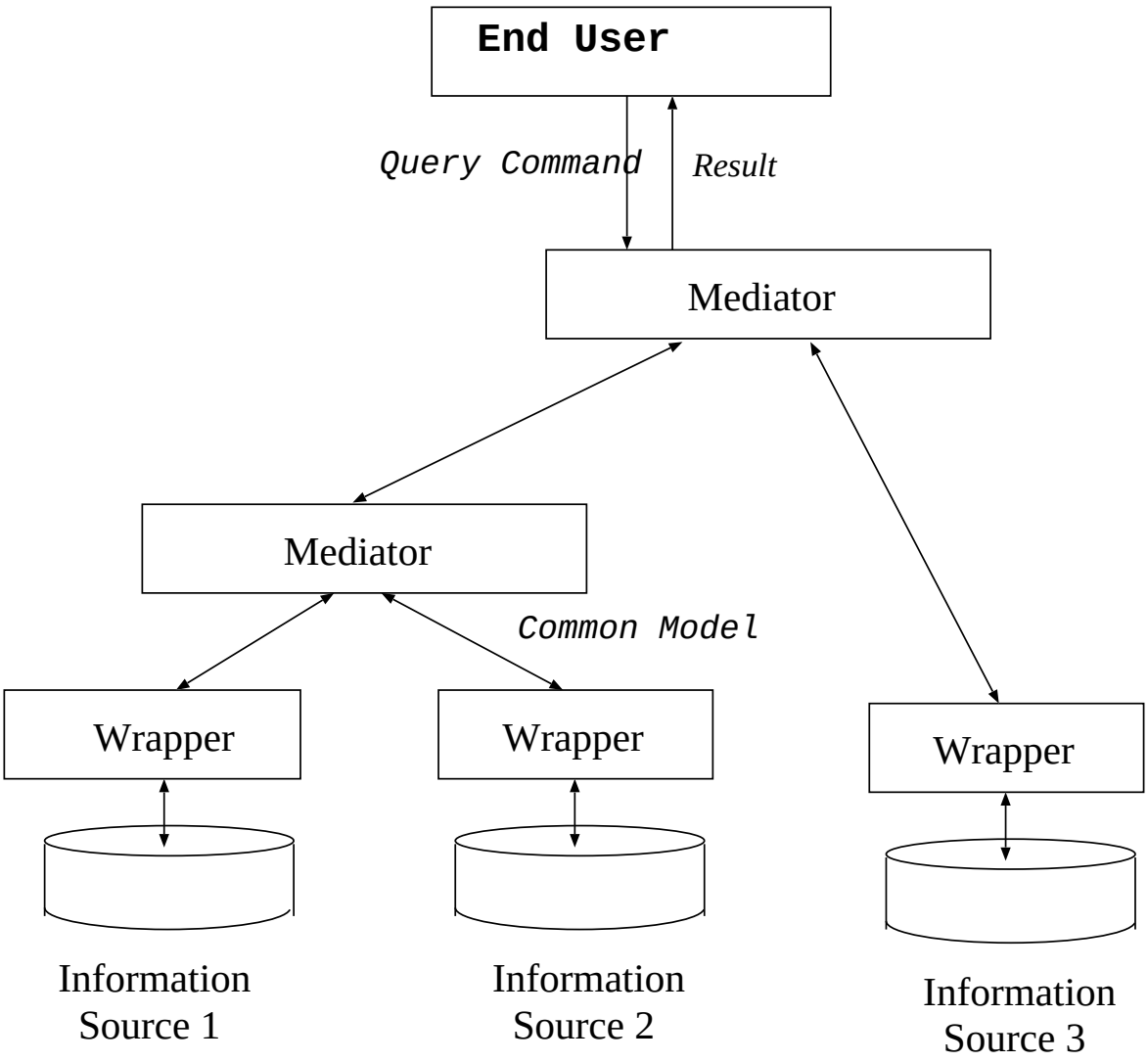


Figure 2.2: A common architecture of information integration system

the corresponding wrappers, receives answers from the wrappers, integrates the answers, and provide the user with a total result.

The following is the main issues in the work of the information integration systems.

Common Data Model: One important element in the integration system of heterogeneous information is the common data model. Several systems have proposed their common data models. However, from our observations, the common data models can be divided into two types: (i) a lightweight data model, and (ii) a heavyweight data model. The outstanding data model of the first type is the Object Exchange Model (OEM)[44] proposed in the TSIMMIS project of Stanford university while that of the second type is the object-oriented data model (OO). Because the OEM is the basic data model that is adopted and extended to be other data models by several integration systems, we describe it in more details.

Data represented in OEM can be though of as a graph, with objects as the vertices and labels on the edges. Entities are represented by *objects*. Each object has a unique *object identifier* (OID). Some objects are atomic and contain values of the basic atomic types, e.g., integer, real, string, gif, etc. All other object are complex; their value is a set of *object references*, denoted as a set of (label,OID) pairs. Figure 2.3 provides an example of an OEM graph.

OEM can easily model relational data, object-oriented data, hierarchical and graph data. Because OEM is not a strongly typed structure such as standard database models, it can be used to model semi-structured information sources. For example as shown in Figure 2.3, we can see that (i) a staff can have zero, one or more affiliates; (ii) an affiliate is an object in one place and a string in another place; (iii) advisor information is sometimes allowed to be omitted. The detail of semi-structure can be

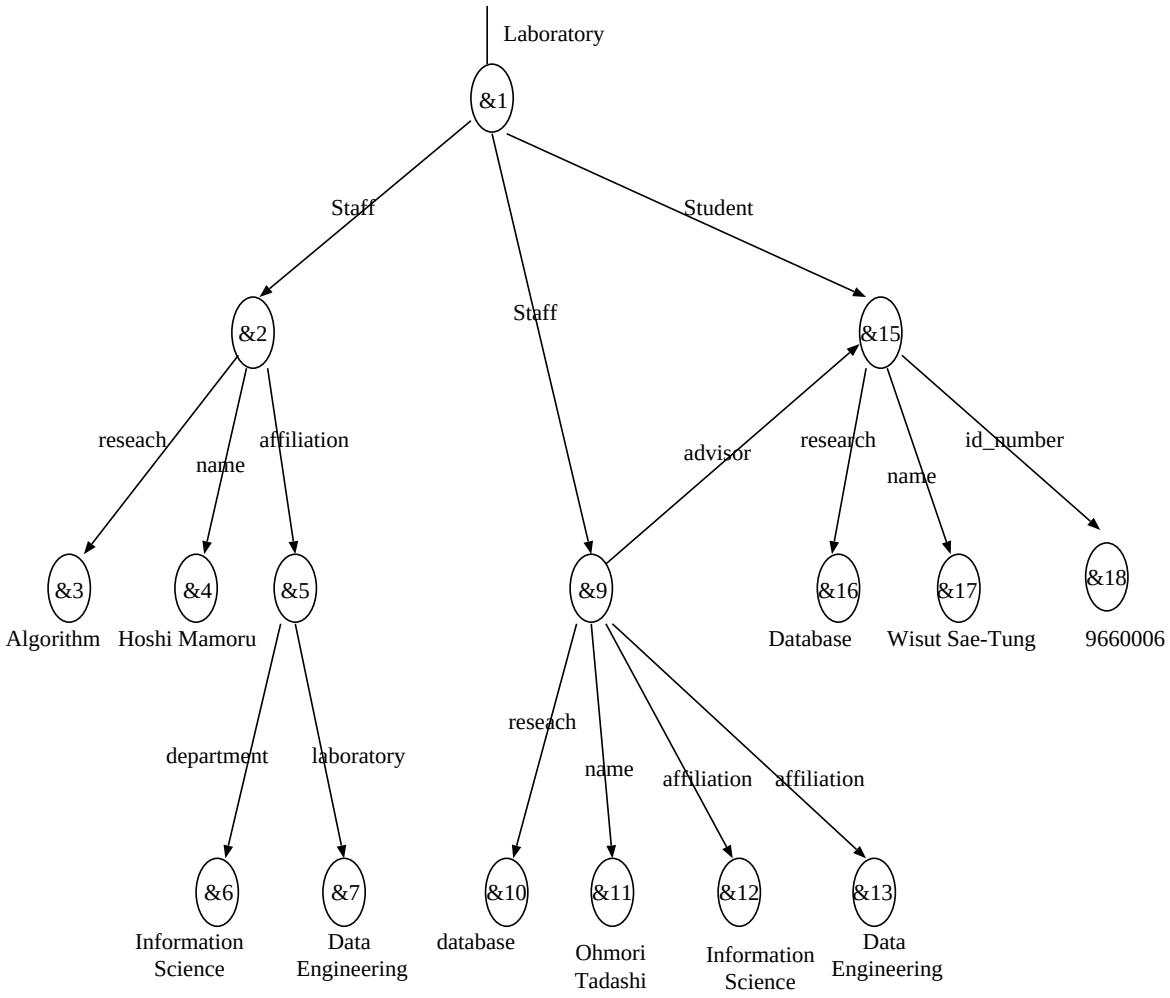


Figure 2.3: A sample of OEM graph

seen in [25].

The OEM is a schema-less data model. Because of its simplicity, it can be used to model any types of data models. However, its disadvantage is its heavy load to realize a query scheme upon it. It requires an ability to discover concise and accurate structural summaries of semi-structured information[24].

To query semi-structured data, TSIMMIS provides a query language called *Lorel* that uses the OEM as its common data model. It emphasizes the ability to query the schema of the data, and the ability to accommodate irregularities in the data, such as missing or repeated attributes, and heterogeneous records. However, this language is not developed specially for the Web and do not distinguish between graph edges which represent internal structure of Web documents and external links that interconnect them.

There are many projects applying the architecture of the wrapper/mediator [17, 19, 28]. Some projects [17] use common data models enhanced from the OEM while some projects [19, 28] use those adopted from the OO data model.

Wrapper: The role of a wrapper is to extract data from its underlying information source and to translate the extracted data into data of a common data model. One important research in building the wrapper is how to translate system queries into the native commands of the underlying information sources that may have limited query capabilities. Many projects [27, 35] solve this problem by providing a toolkit that allows developers to specify source descriptions and query reformulation. Another important research [19] concentrates on a wrapper-building task that should be able to evolve; this allows applications to access data from new sources as soon as possible while subsequent enhancements to the wrapper can transparently improve performance

by taking greater advantage of the repository's query capabilities.

Mediator: In general, a mediator [41] is a middle-ware system that provides an integrated view of a variety of heterogeneous information sources. It provides functions that locate, access and retrieve data from several wrappers that communicate with the underlying heterogeneous information sources. Creating an integrated view from the wrappers must solve many problems: schematic mismatch, semantic mismatch, sources selection, and so on. Another important issue that should be taken into account in building the mediator is a query optimization. The mediator should be able to determine the query that each wrapper can execute at its best full capabilities.

The concept of an integrated view allows end-users to query information without having any knowledge about the underlying sources. On the other hand, end-users are forced to use only the integrated view and have no freedom to select and create their own views.

2.3 Web Site Management Systems

In this section, we describe the Web site management system to which the database concepts and technologies have been applied. In contrast to the previous works which concentrate on integrating existing web sites, they concentrate on the process of creating Web sites. Web sites can be constructed either by starting with some raw data (stored in databases or structured files) or by restructuring existing Web sites.

The Web site builder must consider the following tasks: (i) extracting data from original sources, (ii) designing a site structure which illustrates relationships between pages in the site, and (iii) designing the representation of each page.

The common architecture is shown in Figure 2.4. First, it also requires a *wrapper*

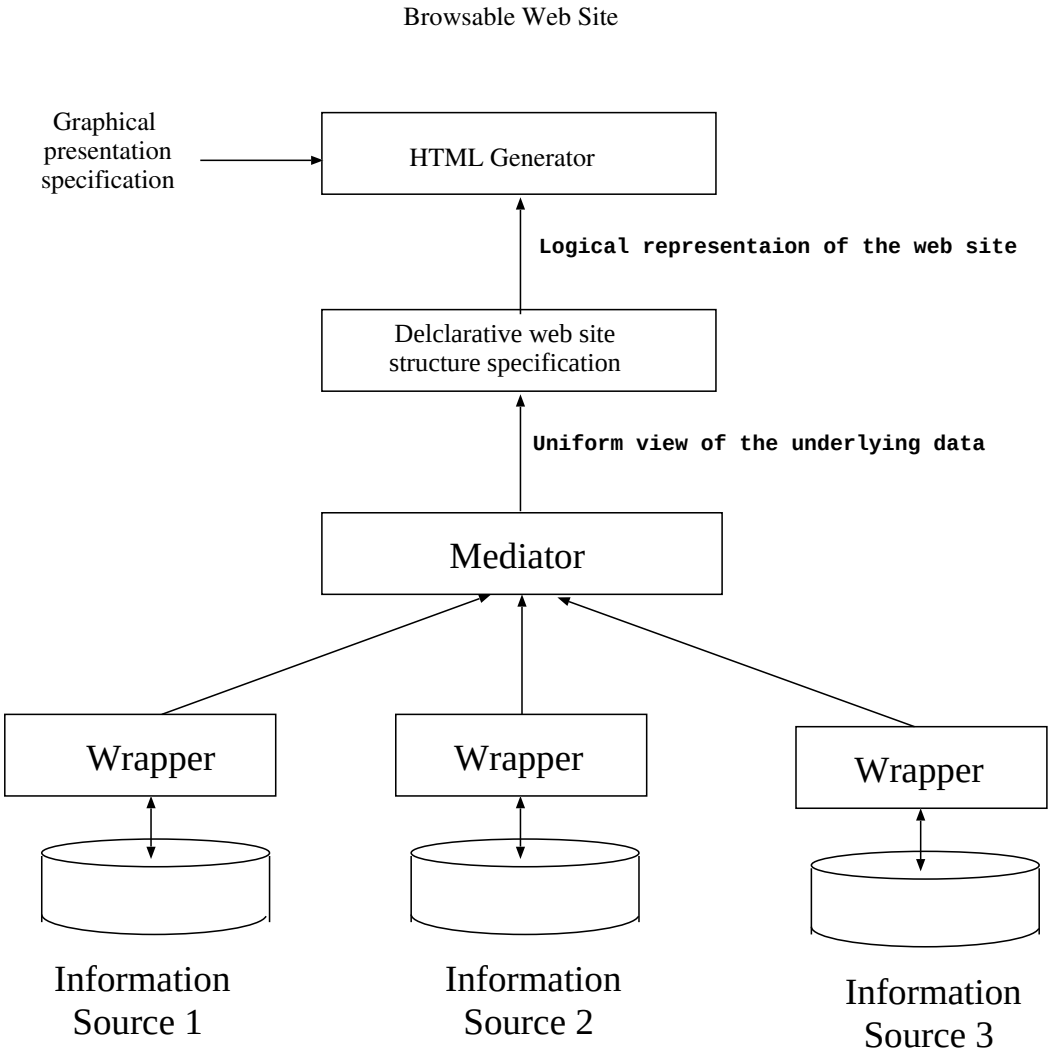


Figure 2.4: An architecture for Web site management system

to extract data and transform the data into a common data model. The data may be stored in databases, structured files, or existing Web sites. Second, the next step is to create a view against the extracted data using a *mediator*. After that, the next step in building a web site is to write an expression that declaratively represents the structure of the web site. The logical representation in the common data model of the Web site is created at this step. Finally, the logical representation of the web site is translated into the browsable form of HTML files using a *HTML generator*.

The system architecture of the Web site management system is not so different as that of the information integration system. Several techniques and concepts (e.g., the common data model, the wrapper and the mediator) are similar. STRUDEL [17] uses the labeled directed graph of OEM as its common data model. STRUDEL provides a query language, throughout the system, for integrating the raw structure (as mediator) and defining the structure of the Web site. ARANUES [23] uses a more specific common data model called *ADM* to model the Web including internal structures and external links of the Web page. However, data described in ADM are extracted and built as data of tables in a relational database, and it uses the relational database system as the mediator to create a uniform view. In addition, ARANEUS provides a language for creating web sites from data stored in the relational database. Unlike the heterogeneous information integration system, we can see that they provide a clear separation between the creation of the logical structure of the Web site and the specification of the graphical presentation of the site.

Chapter 3

Basic Idea and System Concept

In the previous chapter, we have discussed recent researches concerning about Web-data management problems. However, these researches concentrate on providing a framework for developers. None of these systems provide frameworks for mobile users. In this chapter, we describe problem statements, basic ideas for solving problems, and a basic model for our system framework.

3.1 Problem Statements and Strategies

Let us consider the environment as shown in Figure 1.3 again. In order that mobile users can freely manipulate and integrate information on several autonomous WISs, the following problems must be solved:

Problem 1: Unlike the relational model that data and integrated view are represented in simple table, integration of information on the Web requires more sophisticated representation.

Solution: Even though query languages for the Web are powerful enough to query and restructure web documents by extracting required data and re-composing them into a new structure, they are still too difficult for end-users to understand and

use. Therefore, this requires a new style of integration in the WWW framework. The integrated view should be based on relationships of Web pages with hypertext links.

Problem 2: In general, users do not know what kinds of data are provided by WISs and how to access them. Therefore, WISs should provide such information for users.

Solution: Several works solve this problem by creating uniform integrated view of information or logical representation of the Web site. This forces the users to use only the provided views. In our approach, schema and access methods of a WIS must be wrapped and downloaded into the users' mobile computers. After that, a system in the mobile computer uses and explores the downloaded information to help the user define integration.

Problem 3: In the process of generating integration, semantic conflicts among information from independently-developed organizations must be solved. Therefore, there should exist some mechanisms to support automated semantic conflict resolution.

Solution: Several works[20, 36, 41] provide tools to solve this problem but using these tools still requires knowledge about information. Moreover, semantic conflict resolution must be pre-defined in a centralized system. Therefore, the system architecture must be enhanced to support data conversion between independently-developed organizations.

As summation, we can provide a comparison of our approach with other dominant integration approaches by way of Table 3.1.

Table 3.1: Comparison between our approach and other existing integration approaches

Features	query languages for WWW	Heterogeneous Information Integration Systems	Web Site Management Systems	Our Approach
Explicit source selection	Good support	No support (predefined views)	No support (predefined views)	Good support
Automatic semantic resolution	No support (users must do by himself)	No support (resolved in predefined views)	No support (same as the second approach)	Good support
Scalability	Limited support difficult to use and understand	Limited support scale of conflict resolution= $O(n^2)$ where n in a number of systems	Limit support (same as the second approach)	Good support
architecture	Stand alone system	Centralized	Centralized	Distributed
Target	Only Web	Heterogeneous sources (RDB, OODB,and so on)	Same as the second approach	Hetrogeneous WISs
Integration definition	Defined by users	Defined by Developers	Defined by Developers	Defined by users

In this chapter, we discuss a new style of integration system of Web information sources suitable for end-users, especially for mobile users. This chapter also discusses necessary technologies required to achieve the new style of integration system. Section 3.2 describes basic idea of the new style of integration of the WWW framework called *navigational integration* and its concept of implementation. Section 3.3 describes a component model of WIS that shows constraints of WISs used in our framework. Section 3.4 describes an integration model in our framework. Section 3.5 describes the execution step of navigational integration. Section 3.6 describes our system architecture. Section 3.7 describes a region design for resolving semantic conflicts among different organizations. Section 3.8 summarizes this chapter.

3.2 Basic Idea

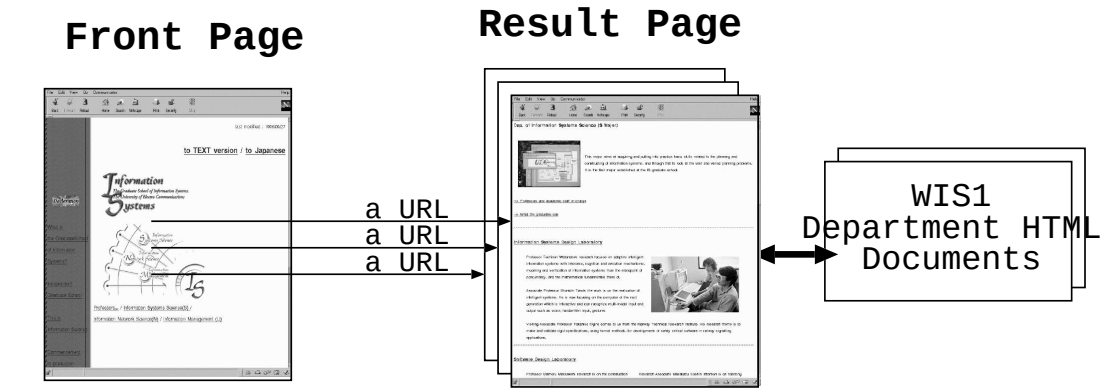
In this section, we overview a basic idea of navigational integration and its implementation. The formal definition of navigational integration will be described in chapter 4.

In general, in the WWW framework, links are important elements for users to find relevant data between two pages. Navigation through links is a core facility. From this point of view, integration in the WWW framework should be represented by links, which describe relationships among data appeared in two Web pages. Namely, a link should be regarded as an invocation of a server-access function of a WIS to get a result page. Thus, the integration of the WWW framework in our approach is set as a process of embedding new links into a Web page of one WIS in such a way that they invoke server-access functions of another WIS. We call this integration style *navigational integration*. Moreover, we call the new links generated in the navigational integration *derived links*.

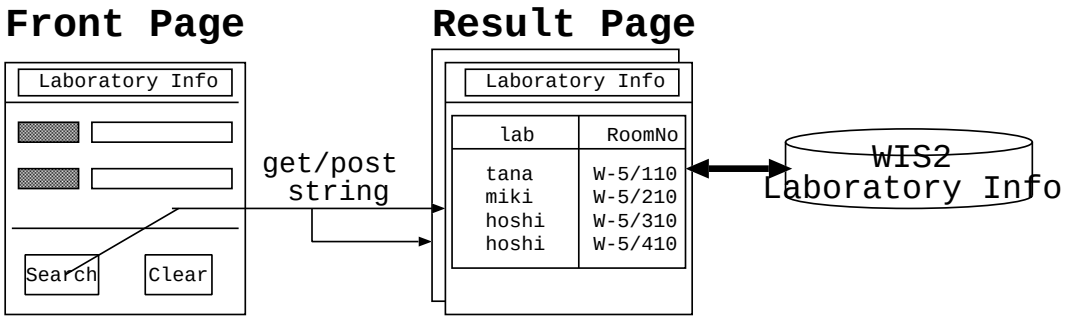
In the following, we explain an example of navigational integration and describe a basic idea to define it.

3.2.1 Example of Navigational Integration

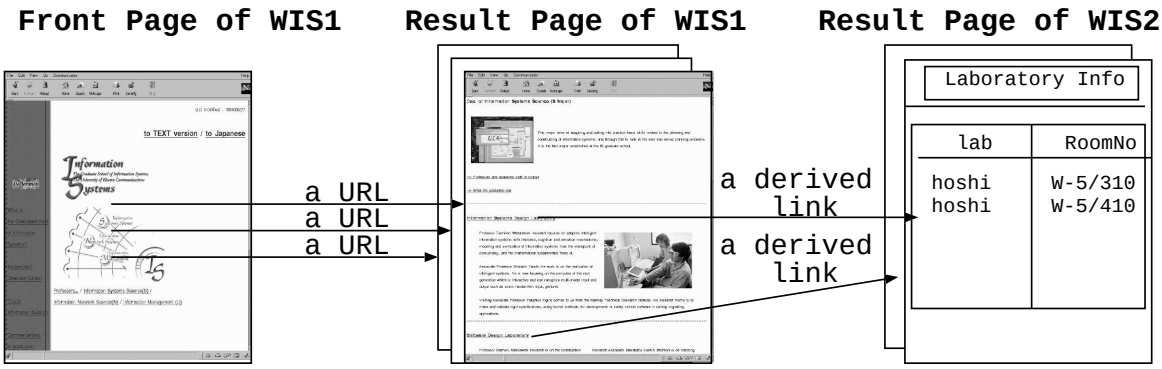
Let us consider an example in Figure 3.1. The first WIS (WIS1) shown in Figure 3.1-a is an ordinary WWW document server. Its WCA has a Web front-page including entry points that specify URL links to access some HTML-pages (under a given DTD) describing department information. The second WIS (WIS2) shown in Figure 3.1-b is a WWW server using a CGI program to access a database server. Its WCA has a CGI-form page as a front page through which a user can input condition parameters



(a) The First WIS (WIS1)



(b) The second WIS (WIS2)



Service invocation of the WIS2 is embedded as a derived link in the result page of the WIS1

(c) Navigational Integration (WIS3)

Figure 3.1: Navigational integration

to retrieve a dynamic result page. Then, the user wants to generate a new WCA from these original WCAs by embedding *derived links* between pages of these WCAs in such a way that it skips over unnecessary re-input operations. For example, the derived links pass the data (e.g., laboratory names) from the result page of the WCA of the WIS1 as the input conditions of the next WCA to get the result pages of WIS2 (Figure 3.1-c). When clicking these derived links, the user can navigate from contents of one web page to another page of two WISs as if they are provided from the same WIS.

3.2.2 Basic Idea to Define Navigational Integration

Figure 3.2 describes our assumed model of a WIS. In our approach, a WIS consists of both a backend server and a Web client application (WCA). We will concentrate on the WCA. We assume that the WCA is made of two parts: a front page part and a result page part. In the front page, there are entry points that perform as links, and the links can access a set of the result pages that are generated by its backend server. Thus, a link between the front page and its corresponding result page can be regarded as a function invocation to the backend data server. For instance, the function invocation may be: (i) a simple URL to access another HTML page, (ii) a URL that forms a CGI query string to invoke the database server to generate a dynamic HTML page, or (iii) a remote procedure call (RPC) to invoke a function (e.g., a java function or a JDBC function) on the database server.

We aim to generate a new client application for performing navigational integration. This can be done by the following steps:

Step 1 To integrate information among several WISs, we wrap the WISs into a uniform manner. Namely, data of WISs must be wrapped by using a common data model.

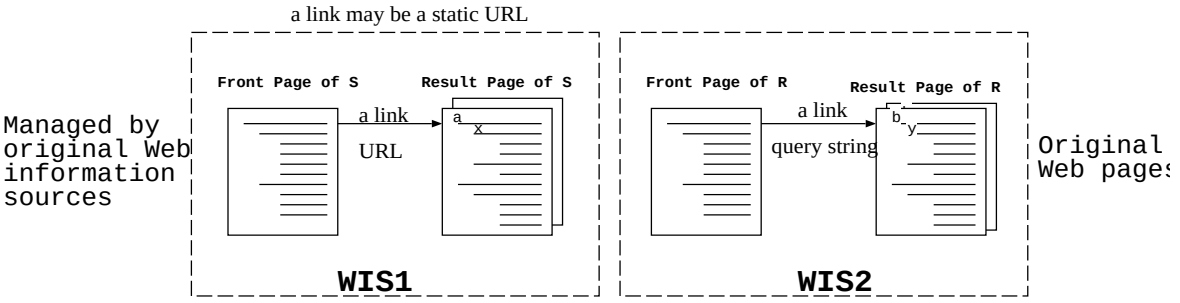


Figure 3.2: Model of a WIS

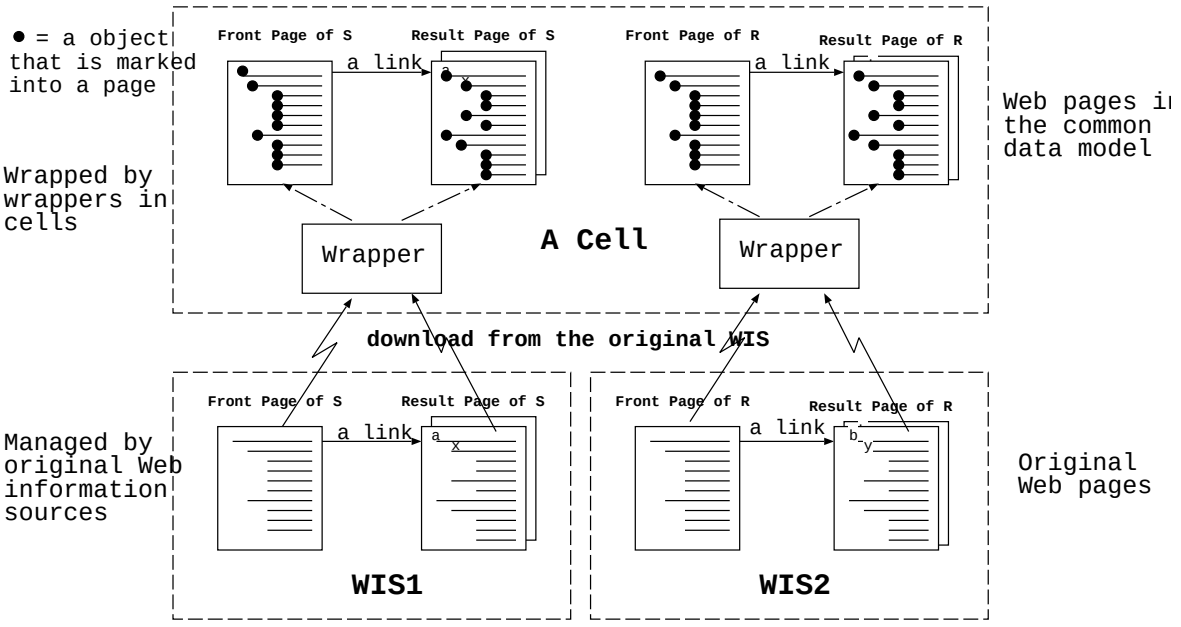


Figure 3.3: The Process of WIS wrapping by the integration system

Figure 3.3 shows the process of Web wrapping in our integration system. Our common data model is a variant of object-oriented model [1]. Each Web page is viewed as an object with a set of attributes. We will describe it in details in the next chapter. A front page and a result page of a given WIS are wrapped in order to abstract their representation with respect to the layout structure including records and nested structure. Based on the common data model, values in the page can be specified using the concept of path expressions. Moreover, the server-access function of the WIS such as a CGI-string query can be described by a method of the object class.

Through the process of wrapping, data on each page are extracted to generate objects. In addition, the positions of the data on the page are marked with identifiers of objects and their attribute names. Thus, the system can find the position of each value in the Web page for embedding a derived link in the next processing.

Step 2 Derived links must be able to be specified by users. The links are created at the positions of specified values. To do so, derived links are defined over the WISs of interest in a declarative manner based the common data model. We propose a query language to provide this ability.

We explain this idea by Figure 3.4. Consider two given WISs S and R. It contains two parts: (i) links from the front page of S to the result pages of S, and (ii) links from the result pages of S to the result pages of R. The first part can be described in declarative form by invoking the server-access methods of S with a constant value. The second part can be divided into two modes. Figure 3.4-a describes the first mode of the derived links from the result page of S to the result page of

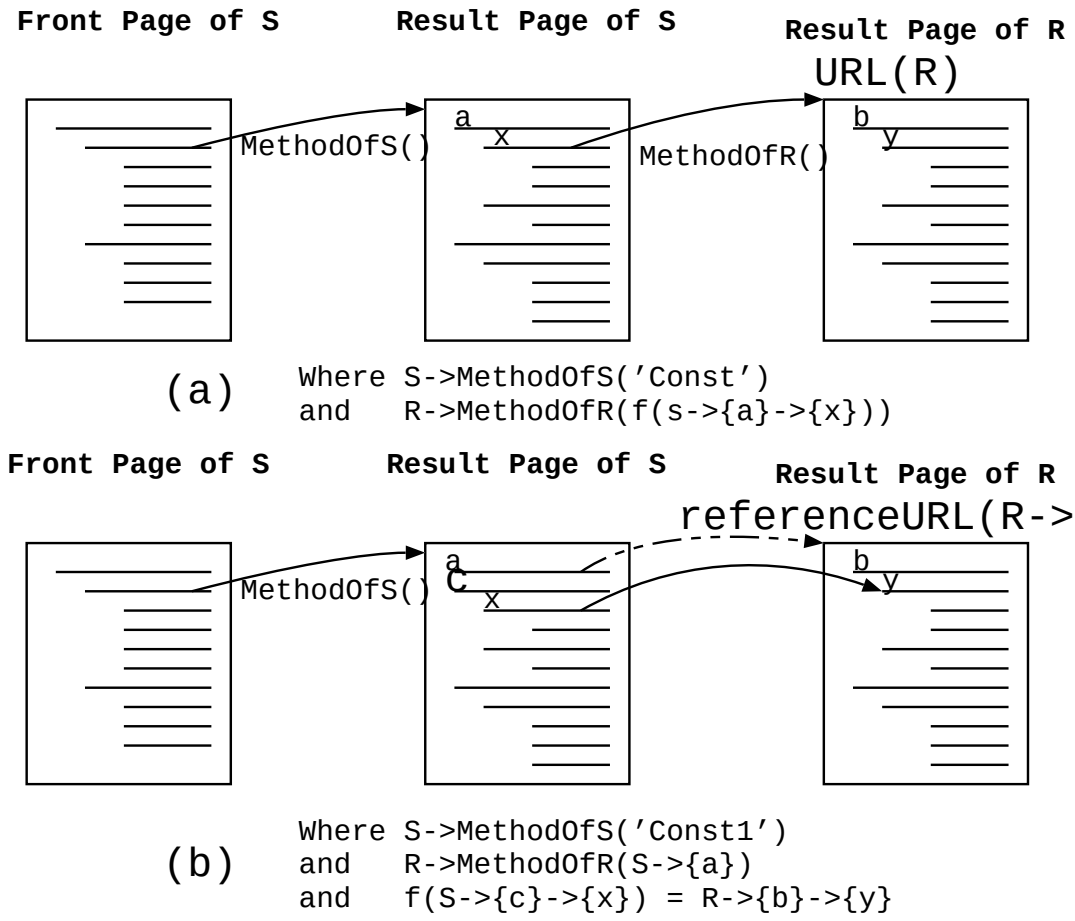


Figure 3.4: Modes of navigational integration

R. Such derived links can be described by invoking the server-access method of R with value of the S as its argument. Figure 3.4-b describes the second mode of derived links from the result pages of S to the internal part of the result pages of R. such derived links can be described by adding a clause of comparison between the value of S and the value of R.

Now, we describe formal description of derived links in our approach. In general, the derived link L can be expressed in a form: $y = f(x)$, where x is a value in the result page of S, y is a value in the result page of R, and $f()$ is a function expressing a relationship between x and y. In hypertext framework, the derived link that represents the relationship between two values of two pages can be written in two modes:

$$\begin{array}{lcll} \text{Link}(S \rightarrow \{x\}) & = & \text{URL(R)} & (1) \\ & \text{or} & & \\ \text{Link}(S \rightarrow \{x\}) & = & \text{referenceURL}(R \rightarrow \{y\}) & (2) \end{array}$$

$Link(x)$ is a function to create a derived link for the value x. From the above equation, Link($S \rightarrow \{x\}$) means a link created by the value x in the result page of S. $URL(R)$ means URL of the result page of R, and $refenrenceURL(R \rightarrow \{y\})$ means the URL of the internal part of the result page of R containing the data y. The equation (1) represents the first mode of navigational integration as shown in Figure 3.4-a while the equation (2) represents the second mode as shown in Figure 3.4-b.

In this way, navigational integration is described by a declarative query command. The last step is to build a new client-part to perform navigational integration.

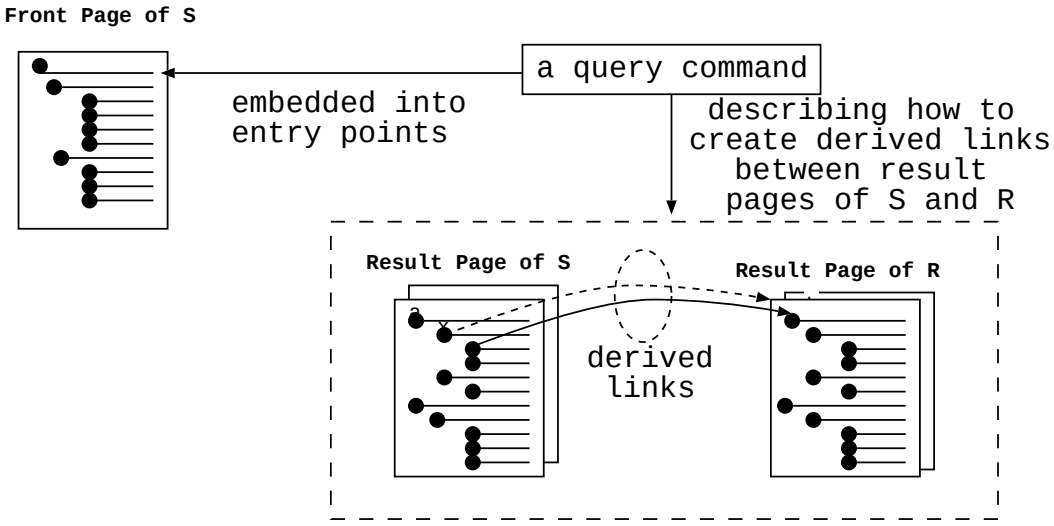


Figure 3.5: The process of generating a client application by a user

Step 3 A front page of the first WIS is created as the front page of the new customized WCA.

To do so, the query command that describes the derived links between result pages of multiple WISs is embedded into the entry points of the front page of the first WIS as shown in Figure 3.5.

To summarize our idea, first, WISs must be previously wrapped based on the common data model; this process is prepared in each cell. Next, based on the common data model, a user generates a query command that specifies what WISs to be accessed and how to create derived links between them. Finally, the generated query command is embedded into the front page of the first WIS to generated a customized WCA.

So far, we have described the basic idea to generate a new client part for performing navigational integration. However, there are two problems remained to be solved. The first problem is how to generate this new client part on mobile computers in a disconnected mode. The second problem is *semantic conflicts* among data of different

organizations. The following sections will overview how to solve these problems.

3.3 A Component Model of WIS

To fit a WIS in our approach, the WIS must satisfy the following constraints. In general, a WIS can be regarded as a two-part structure, consisting of a backend server and a Web client application (WCA). The backend server can be either an ordinary WWW document server or a WWW server with backend database. The WCA is a client application that is embedded into a Web page and sent to be executed on a user's browser. In general, a WCA can be divided into two parts: *a FP part* (Front Page) and *a DA part* (Data Access). The FP part is responsible for gathering information from a user as input data and sending the data via its *entry points* (e.g., links in a HTML page or "Send" button in CGI-form text) to the DA part. The DA part is responsible for passing the user's input as conditions, accessing and receiving a result page from the information source. Thus, the DA part consists of two elements: (i) server access functions and (ii) specifications of data returned as the result page from the backend server. The characteristics of its WCA must satisfy the following constraints:

- To be able to control these two parts from the outside, the DA part must be distinctly separated from the FP part (as shown in Figure 3.6.) This figure shows three examples of WCAs to which our approach can apply: an ordinary Web page, a CGI-form HTML and a WCA written in the Java language [13]. In general, the ordinary Web document and the CGI-text form satisfy this constraint. However, in case of a Java applet, a graphical user interface and a server access (a DA part) of the applet must be implemented in individual methods. For example, an applet whose method controls both a mouse event and gets data on GUI cannot

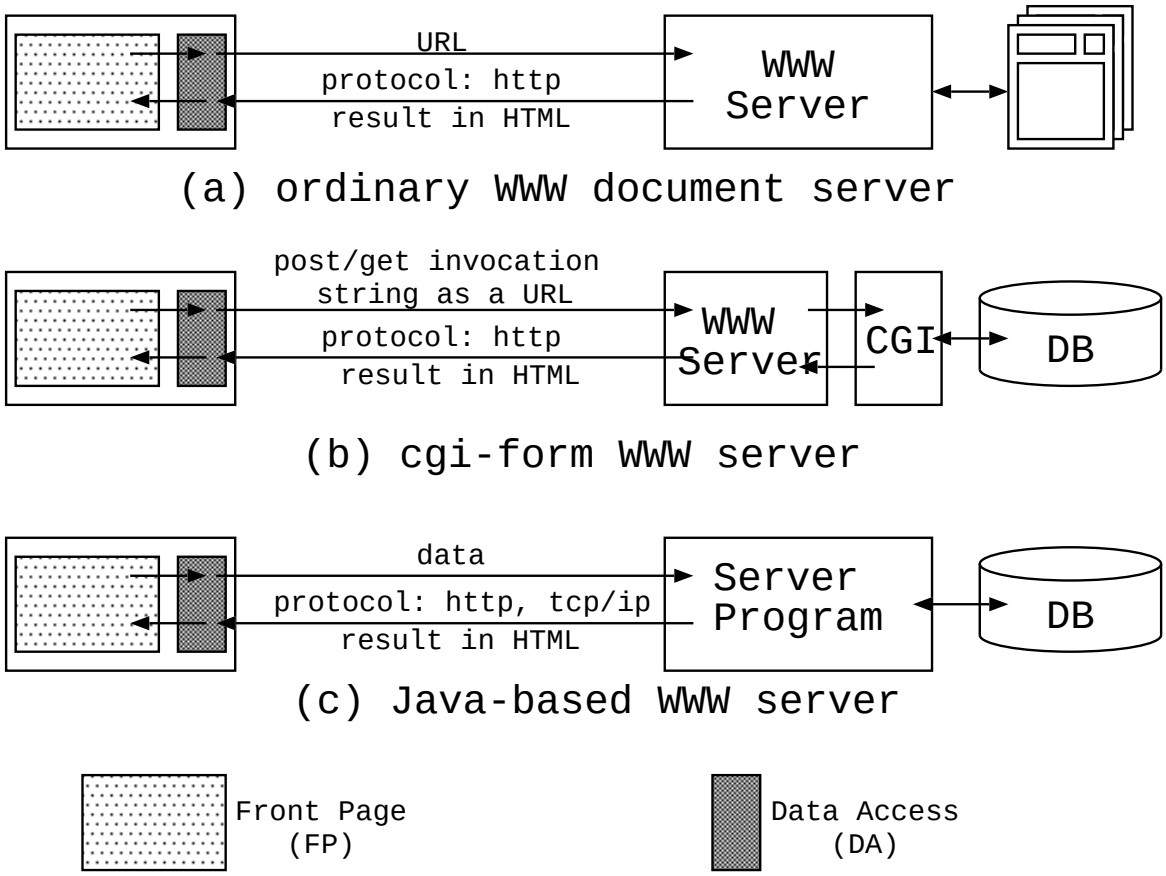


Figure 3.6: Components of WISs

be fit in our approach.

- The FP part and the DA part must be able to be modified. The navigational integration requires the ability to update the entry points of the FP part and to embed additional codes in the result page of the DA part. Similar to the first constraint, the ordinary Web document and the CGI-text form satisfy this constraint because their codes are written in HTML format.

  In case of a Java applet, the java applet that satisfies the first constraint also satisfies the update-able FP part because we can use the inheritance coding technique to overwrite methods in the original java applet. Thus, a java applet that can be fit in our approach must satisfy the first constraint and returns a HTML page as a result page.

We can summarize that an ordinary Web document and a CGI-text form satisfy our approach in all cases but java applets satisfy our approach in some cases.

3.4 An Integration Model

To provide a mobile user with an ability to create his own WCA to integrate information from several WISs, the process of our integration model can be summarized as shown in Figure 3.7. First, to present users a uniform interface, all services of WCAs must be wrapped in the common form called *interface definitions*. Therefore, the FP and the DA of each WCA must be wrapped by interface definitions. Next, the user downloads the interface definitions (from cells that manage these WISs) and registers them into his mobile unit. Next, using a *mediator* module on the mobile unit, the user generates a query command for performing navigational integration of these WISs from the DA interface definitions. By embedding the query command into the entry points of the

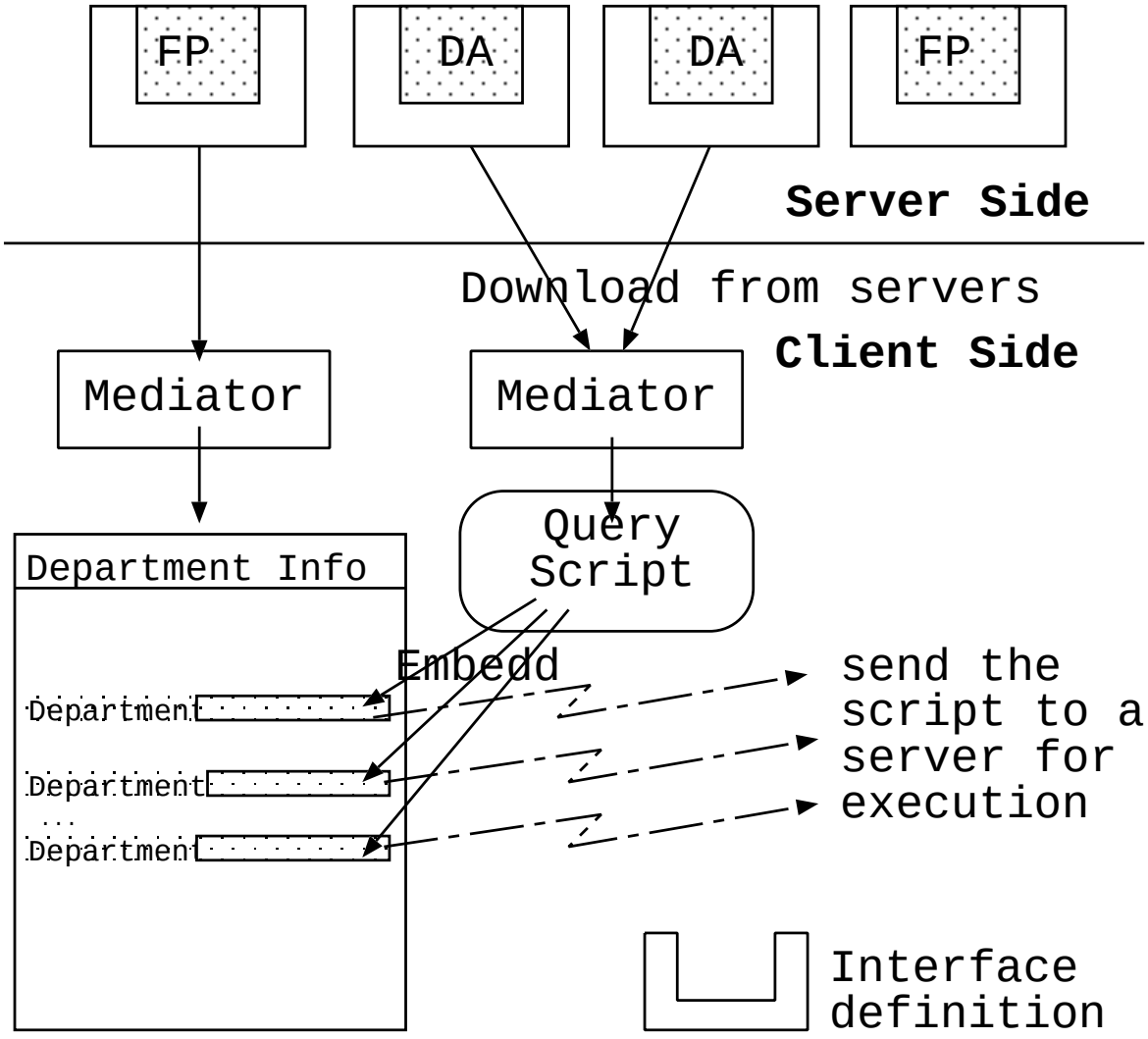


Figure 3.7: The integration model

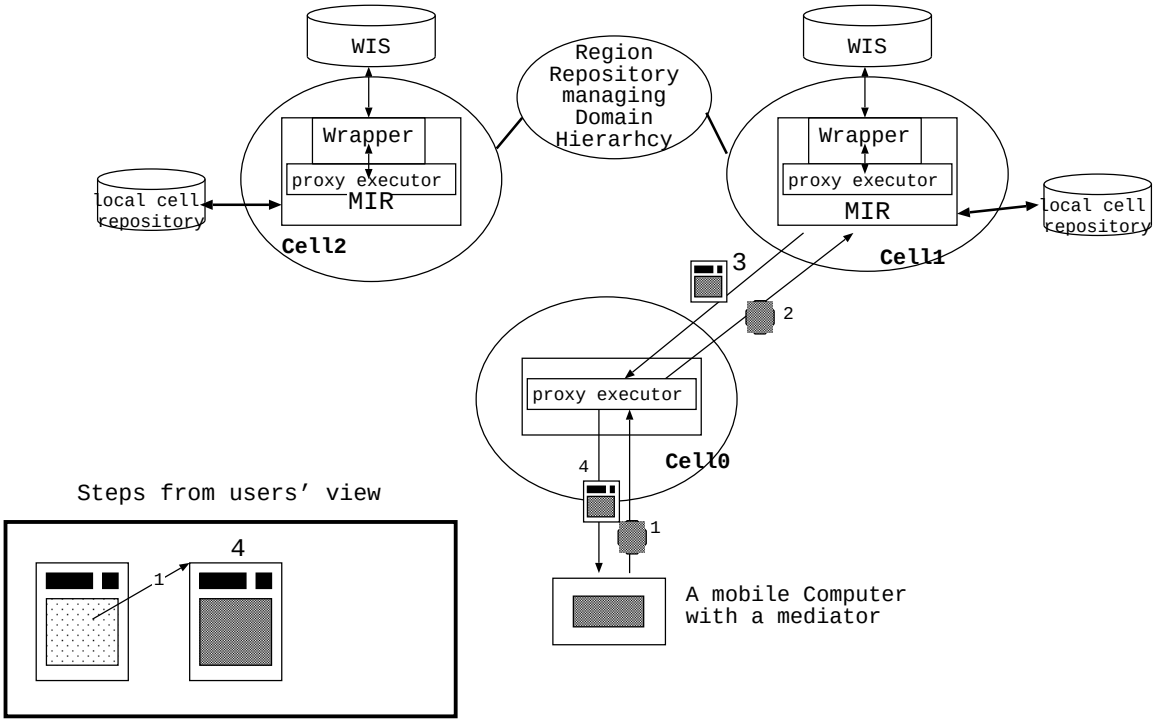
FP of the first WCA, the user can generate a new client application. By loading this new client application into a browser, the user can send the query command to be executed at a remote site through the FP of the WCA of the first WIS. Note that this building is done without network connection.

3.5 Execution Model

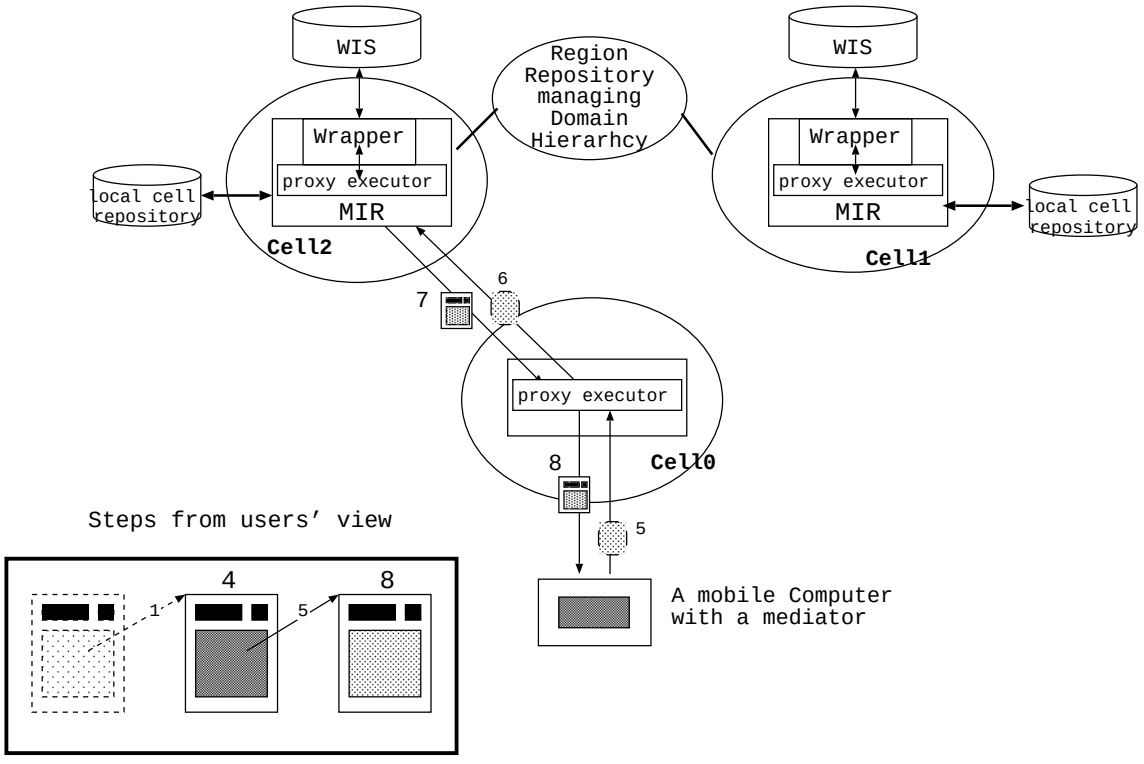
To continue from the previous section, in this section we will describe an execution model of the navigational integration. To understand the execution in the navigational integration more clearly, we use the Figure 3.8 to explain the execution steps in the navigational integration. The integration in this example uses two WISs. The first step shown in Figure 3.8-(a) is the navigation from the front page to the result page of the first WIS. The second step shown in Figure 3.8-(b) is the navigation from the result page of the first WIS to the result page of the second WIS. The sub-figure shown in the left side of each figure is the result that is seen by the user on his browser.

As described in chapter 1, each cell has a main information resource (MIR) as a yellow-page server. The MIR in each cell manages several WISs; each WIS can be accessed through its wrapper and has one proxy executor to execute requests from mobile users. In addition, each cell has a local cell repository that maintains interface definitions of its maintained WISs as well as the local domain hierarchy defining semantic information of data used in its cell. All cells in the same region also share a region repository that manages the domain hierarchy information for resolving semantic conflicts among them.

Figure 3.8-(a) shows the execution of the navigation from the front page to the result page of the first WIS. First, the user sends the query command to any MIR (a MIR of



(a) The Execution Step in Navigational Integration (1)



(a) The Execution Step in Navigational Integration (2)

Figure 3.8: Execution step in navigational integration

cell0 in this figure) by clicking the link that the query command was embedded (step 1). At this MIR, the proxy executor then receives the query, registers the command into its system as an executable object, analyzes the query into several sub-queries, and sends a sub-query to the MIR of the first WISs managed in cell1 (step 2). Next, the proxy executor of the cell1 executes the sub-query by sending it to the corresponding wrapper. The wrapper gets a result page from the WIS, and extracts data in the page to generate objects in the common data model. After that, the wrapper embeds the objects into the result page for marking positions of data. The result page is sent back to the MIR of cell0 again (step 3). Next, the proxy executor receives the result page, embeds the derived links for activating the next step of the navigational integration into the position of data marked earlier, and sends the result page back to the user again (step 4). From the user's view point, he sees only the step (1) and (4).

Figure 3.8-(b) shows the execution of the navigation from the result page of the first WIS to the result page of the second WIS. Similar to the first step, first, the user clicks the derived links for retrieving the relevant information he wants. The embedded command in that link is sent to the proxy executor of cell0 again (step 5). Next, the proxy executor of cell0 restarts the execution that it has registered at the first step according to the parameters sent with this request. Then, it sends the request to the next MIR in cell2 (step 6). Likewise the execution in cell1, cell2 returns the result to the MIR of cell0 (step 7). Finally, the proxy executor returns the result page of the second WIS to the user (step 8).

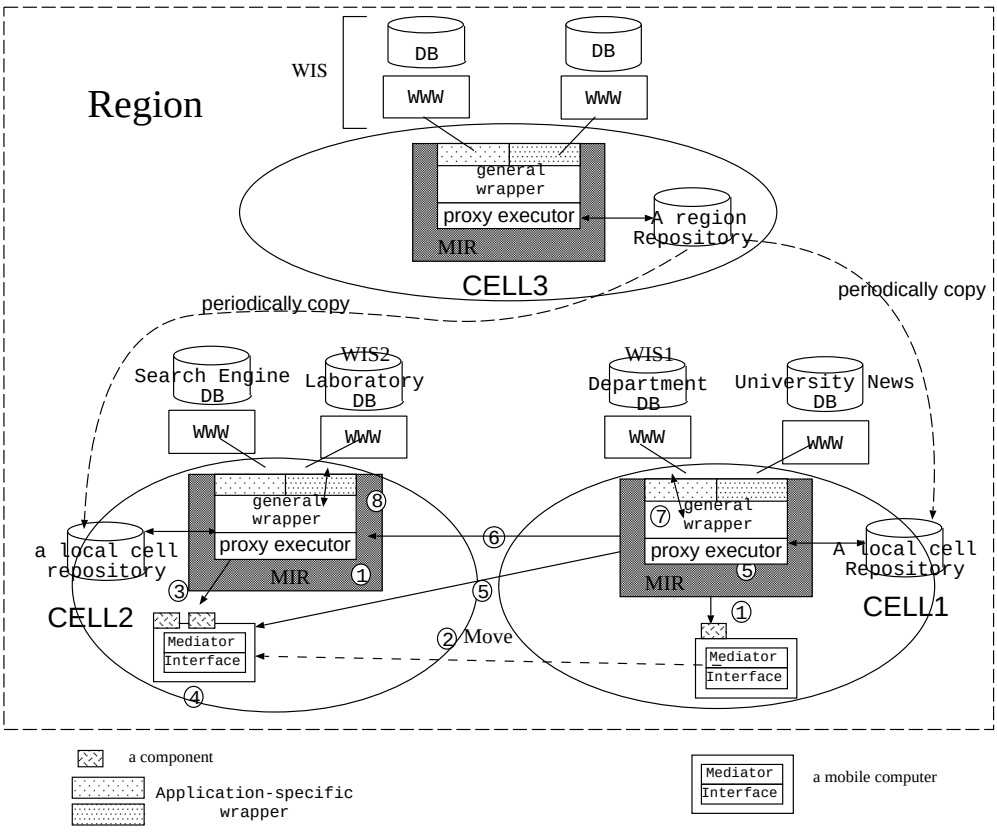


Figure 3.9: A system architecture

3.6 Components in System Architecture

In chapter 1, we have described the system architecture employed in our approach briefly. In the previous section, we have described the execution of the navigational integration using the component modules in the rough system architecture in the chapter 1. Now, we will describe the system architecture and its components of our integration approach. Formal description is in chapter 4. Figure 3.9 shows the system architecture that implements the navigation integration. It consists of the following components:

Web Information Sources: Existing Web information sources are shown as disk-shaped diagram with the WWW system in Figure 3.9. They provide their information through their Web client applications. These components are already existing in the networks. We need not reimplement them.

To inform users of service information, the client application and some necessary information for information integration must be wrapped in interface definitions. We will describe the interface definition in chapter 4.

A Region repository: To allow data from autonomous cells to be exchangeable, individual cells must agree on the meaning of their data. The region repository maintains information of the global domain information that all cells in the same region agree on standard domains in their region for resolving semantic conflict problems between data of different cells. The details will be described in chapter 5.

In this system architecture, the master version of region repository is maintained in one MIR of the region and its copies are distributed to all cells of the region periodically.

MIR (Main Information Resource): A MIR acts as a yellow page server of a cell to assist mobile users in finding and integrating services of the information sources. It contains three component modules as follows:

Wrapper: This module consists of a general wrapper and several WIS-specific wrappers. Each WIS-specific wrapper is implemented corresponding to the standard interfaces defined in the general wrapper. With the interface definition, the WIS-specific wrapper converts a query sent from a proxy executor into a native command of the corresponding WIS, executes to retrieve a result page, converts the data in the result page into objects represented in a common data model, and embeds the objects into the result page of the WIS for marking the positions of data.

The wrapper module does not necessarily reside in the same server as the Web information source. The wrappers reside in a MIR.

A proxy executor: To control overall execution, the proxy executor is also installed into a MIR. It performs many operations: receiving and analyzing query commands from users, executing queries based on the plan generated by the mediator on mobile computers, communicating with its wrappers and other proxy servers, embedding derived links for navigational integration and so on. Note that the proxy executor performs some parts of tasks performed by the mediator of the traditional integration systems.

Local Cell Repository: This module maintains information of the cells such as (i) interface definitions of managed WIS, and (ii) a local domain information for resolving semantic conflict problems among data of different cells. The details will be described in chapter 5.

Mediator and User Interface: These two components are installed in a user's mobile computer and work together as a front-end system to generate a customized WCA. We propose a query language for mobile users to describe their integration requirements (chapter 4). The mediator explores information in downloaded software packages and communicates with the user through the user interface to help the user generate a query command (chapter 5). The mediator finally embeds the query command into the front page of the first WIS. Therefore, using the new front page, the user can send the query command to remote server for executing navigational integration.

The overall behavior of the system is shown in Figure 3.9. A mobile user moves across two cells from CELL1 to CELL2 and downloads software components (interface definitions and some necessary information) of WIS1 (in CELL1) and WIS2 (in CELL2) as shown in step 1-3. Next, by the indication of the user, the mediator on the user's computer uses the downloaded software components in the mobile computer to resolve semantic conflicts (the details will be later in the chapter 5) between WIS1 and WIS2, and it generates a query command for navigational data-access from WIS1 to WIS2 (step 4). The mediator also automatically embeds this script into the copy of the top page of WIS1. The user can use it as a new customized WCA. We should note that the copy of the top page of WIS1 has been already downloaded with the components. Thus, the mediator can do this embedding step on the mobile computer in the disconnected mode. As a result, the user can navigate from the contents of WIS1 to those of WIS2 by clicking derived links that we have described in the previous section (step 5-8 in Figure 3.9).

3.7 Region Design for Semantic Conflict Resolution

Before ending this chapter, let us explain a common problem of integration of heterogeneous information sources called semantic conflict. We also outline how to solve this problem in our approach.

As described earlier, the navigational integration of WISs is defined as a process of creating *derived links* between their Web pages. A link is used to pass data of one page as a condition to access another page. In general, the data of two pages of different WISs may not directly be matched due to semantic conflcit problems. Because WISs are managed in autonomous distributed cells, the data exchange between different cells must be established.

To permit information from autonomous distributed cells to be exchangeable, each cell must agree on the meaning of its data. Such a work is not easy. Currently, several works [27, 28, 34] provide tools for solving this problem by specifying integration in a descriptive way. However, such a static specification is an ad-hoc manner. New information sources may be added into an existing system. In addition, the existing system may change its specification of data. It is difficult to identify the impact of these changes to the existing integration system.

To solve this problem, we propose the multilevel data representation called *a Domain Hierarchy* (DH). The *domain* in our approach refers to a pool of values that have an exact data-presentation and meaning. The DH consists of two parts: *a Common Domain Hierarchy* (CDH) and *a Local Domain Hierarchy* (LDH). The CDH provides standard vocabularies to express the domain definitions used as a pool of exchange domains for all cells. On the other hand, the LDH is a list of the different kinds of domains used in each autonomous cell. The LDH defined in each cell is independent.

By mapping the LDH of each autonomous cell to the CDH, the information of domains in each cell can be exchanged together through the common domain hierarchy. Therefore, we can build a large-scale integrated system in a dynamic environment without inhibiting scalability of overall system and an autonomous management of each cell. We call a system that consists of several autonomous cells capable of exchanging information together *a region*. We will discuss the detail of the domain hierarchy in chapter 5.

3.8 Summary

In this chapter, we have described basic problems and strategies for solving these problems for the integration system of Web information sources by mobile users. The navigational integration is proposed as an integrated view for the WWW framework. To do so, the WISs must conform the constraint as shown in section 3.3. Section 3.4 describes the integration model of our approach. A FP part and a DA part of a given WCA must be wrapped by a unified form called an *interface definition*. The interface definitions of several WCAs are downloaded from cells and registered into a user's mobile computer. Using the mediator on the mobile computer, downloaded information is explored and the user can generate a query command and embed it into the front page of the first WCA. Section 3.5 describes the execution step in the navigational integration. Section 3.6 introduces a system architecture and its integration processing. Semantic conflict problem among information from different sources is also discussed briefly in Section 3.7.

Chapter 4

System Design and Implementation

This chapter discusses the design and implementation of the integration system in our approach. Section 4.1 presents a common data model. Section 4.2 describes path expressions based on the command data model. Section 4.3 describes the definition of navigational integration. Section 4.4 presents a query command and navigational integration in terms of path expressions. The mediator will be represented in Section 4.5. Section 4.6 describes a wrapper and a proxy executor. Section 4.7 summarizes this chapter.

4.1 Common Data Model

To access and extract data from a WIS, its WCA must be described uniformly in a common data model. The data model that we use is a variant of the object-oriented (OO) model. As a unique property, we add a concept called *domain hierarchy* to resolve semantic conflict among data from different WISs. It will be described later in Chapter 5.

As described in the previous chapter, data of a given WCA must be extracted and turned into objects of our common data model. In our policy, a web page of WCA is regarded as a nested structure of objects. Further, in order to model functions

embedded in WCA such as a CGI-page, we give methods to this structure. Similar to the object-oriented concept, *objects* are the basic things for dealing with the *data*. All objects have *identifiers* and are distinguishable. A group of objects with the same characteristics is described in terms of an *interface definition*. The attributes defined in the interface definition may be single-valued or multi-valued. The single-valued attribute corresponds to string data, a link to another Web page or an identifier of an object. The multi-valued attribute contains a set of identifiers of objects. Nested data occurring in the WCA can be represented as relationships between interface definitions via multi-valued attributes.

To support the semantic interchange, we extend the interface definition with a concept of *domain*. Here, a *domain* in our approach is a pool of values that have the exact representation and meaning. The domain can be divided into two types: a system domain and a user-defined domain. The system domain is interpreted by the system in default while the interpretation of user-defined domain is specified by a designer. We attach the domains to single-valued attributes and arguments of the methods described in the interface definitions. We will describe its details later.

In general, contents of a WCA consist of several elements to construct its global schema. Consequently, its global schema can be represented using a set of interface definitions. Each interface definition represents the schema of each element and its attributes represent its possible components. Nested data within a WCA that is considered as relationships between interface definitions can be mapped through multi-valued attributes.

To describe the content of elements of a schema and its properties, the interface definition has the following form:

database DatabaseName
address URLofMIRofCell
Interface InterfaceName **inherit** ParentIntf
body
attribute Domain [,attribute Domain]
method
public:
MethodSignature [,MethodSignature]
private:
MethodSignature [,MethodSignature]
implement
MethodImplementation
endInterface

where "Domain" and "MethodSignature" take the form:

DomainName | **setof** IntfName | **refto** IntfName
and
[**static**] returnValue methodName([D1 ARG1,...,Dn ARGn])

,respectively.

DomainName specifies a name of domain. $D_1, ..., Dn$ are DomainNames, and $ARG_1, ..., ARG_n$ are arguments. Therefore, attaching a domain name to an attribute or an argument of a method means that its value has meaning and representation corresponding to that domain.

IntfName is a name of an interface definition. An attribute that its domain is **refto** followed by Interface name means that its value keeps an OID (object identifier) of

an object belonging to that interface. In addition, an attribute that its domain is **setof** followed by interface name means that its value keeps set of OIDs of objects belonging to that interface. These **refto** and **setof** are used to represent one-to-one and one-to-many relationships between objects of two interfaces, respectively.

As shown above, the InterfaceName specified by the **Interface** is the name of the interface definition being defined. Both IntfName and InterfaceName are names of interface definitions but they are not necessarily distinct. Its URL of MIR and its database name are specified by the **address** and the **database** clauses, respectively. To access data, requests must be sent to this address.

An interface definition wraps a WCA and allows users to manipulate objects using its methods. Its structural representation is defined in the **body** clause. The methods are listed in the **method** clause. Methods are divided into two types: **public** and **private**. For users, our tool uses these two identifiers to export the list of public methods and to hide the list of private methods. Because the WCA provides its own native-access method (e.g., through the form that requires inputs), we separate the access methods defined in the interface definition into two types: (i) one is transformed to native-access method of the WCA, and (ii) the other one operates on the objects of the common data model. We use the **static** keyword to specify a **public** method that is used as the first type. The **public** method returning a boolean value is used as the second type. The **implementation** clause contains the coding of methods. The code program in this part is written in a Perl program [15] because our interface definition is extended from it.

To understand the data model and how to construct the schema of a give WCA more clearly, consider a web page of a department of a university shown in Figure

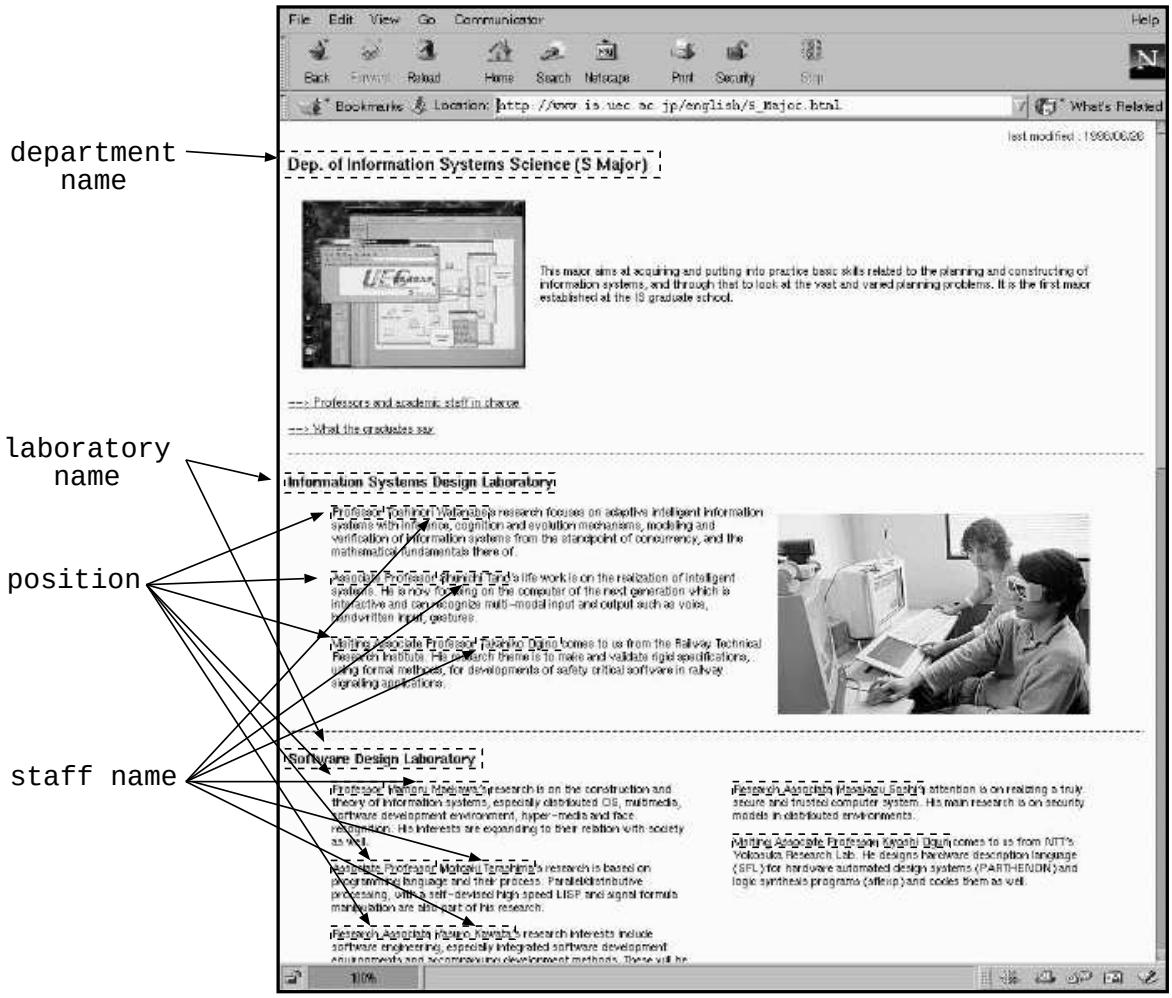


Figure 4.1: The department homepage

4.1. It is the result page of the first WCA of Figure 1.2-a. In this example, it has three elements of schemas: *Department*, *Laboratory* and *Staff*. One department has several laboratories and each laboratory has several staffs. The DTD (Document Type Definition) of this page written in a tree form is shown in Figure 4.2-a. The equivalent structure described in our common data model is shown in Figure 4.2-b.

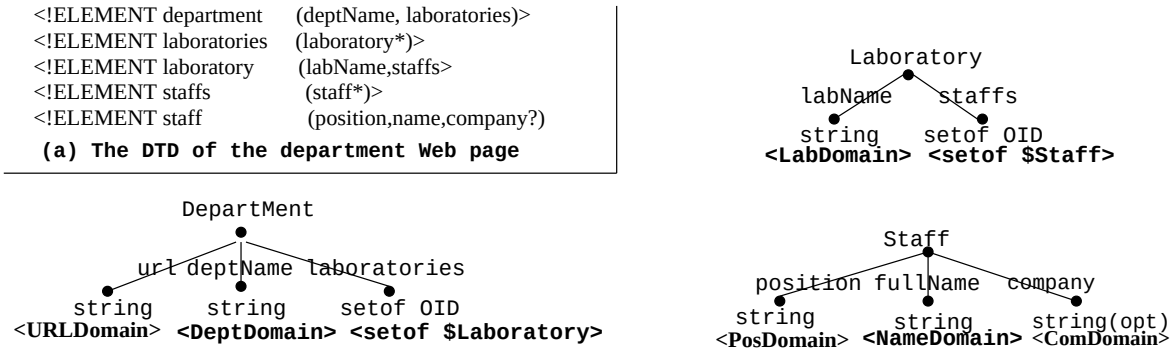
Figure 4.3 shows the interface definitions wrapping the DA part of the first WCA. In order to explain interface definitions, let us use the interface definition of *Department* (in Figure 4.3-a). This interface definition has three attributes: *url*, *deptName* and *laboratories*. The first attribute *url* keeps the URL of the original document page. This attribute is added by a designer. It does not exist in the DTD. The second attribute is a single-value of string keeping a name of a department, and its domain is *DeptDomain*. The third attribute is a multi-valued attribute keeping OIDs of instances of a Laboratory interface. Using such a multi-valued attribute that represents one-to-many relationship, we can represent a nested structure appearing in a document.

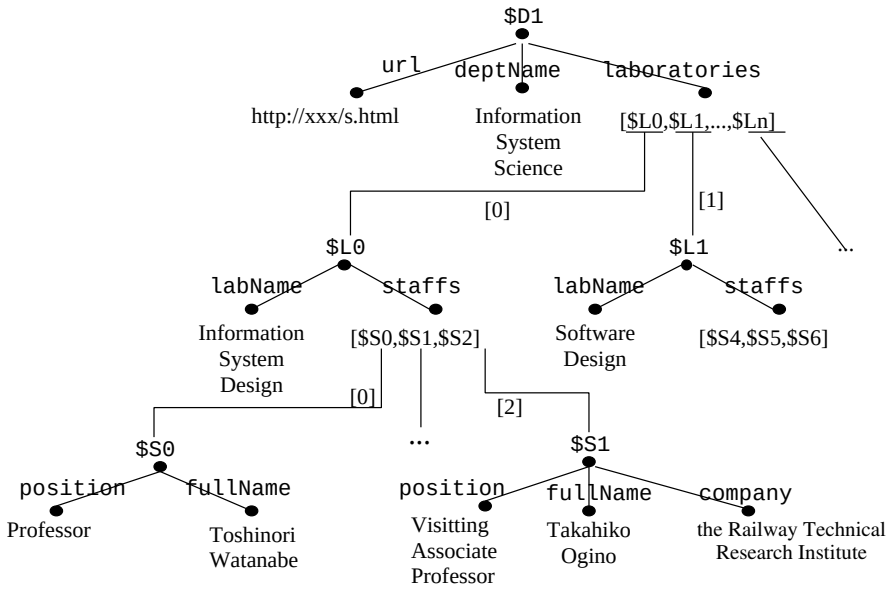
To support an irregular structure that some elements can be omitted, an optional attribute can be defined. For example, the *staff* interface definition in Figure 4.3-c shows the *company* optional attribute. This attribute may have a value in case that a staff is a visitor invited from a company.

As shown above, the data-types in our system are only atomic data and OID. *Domain* must be assigned for all atomic data.

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

Another example shown in Figure 4.4 is the interface definition of the DA part of






```
# object representation of Department
$d1->{deptName} = 'Information System Science'
$d1->{laboratories}[1]->{labName} = 'Sotware Design'
$d1->{laboratories}[0]->{staffs}[0]->{fullName} = 'Toshinori Watanabe'
# GetByURL is class method of Department
Department->getByURL('http://xxx/s.html')
$d1->printDetail() # PrintDetail() is instace method.
```

(c) instances in the common data model

Figure 4.2: Schema and instance

```
database ISInformation
address  http://HOST1/cgi-bin/execute.pl
interface  Department
body
    url             URLDomain,
    deptName        DeptDomain,
    laboratories    setof Laboratory,
method
public:
  static string getByURL(URLDomain $url);
private:
  void printDetail();
  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
```

(a) The "Department" Interface Definition

```
database ISInformation
address  http://HOST1/cgi-bin/execute.pl
interface  Laboratory
body
   labName      LabDomain,
   staffs       setof Staff,
method
public:
  ...
endInterface
```

(b) The "Laboratory" Interface Definition

```
database ISInformation
address  http://HOST1/cgi-bin/execute.pl
interface  Staffs
body
   position    PostDomain,
   fullName    NameDomain,
   company     ComDomain optinal,
method
...
endInterface
```

(c) The "Staff" Interface Definition

Figure 4.3: The interface definitions of a Department Web page

```
database uecdbWeb
address http://HOST2/cgi-bin/executor.pl
class LabBuilding
body:
   lab          JLab,
   roomNo      BldRm,
method:
public:
 static {LabBuilding} getByLaboratory(JLab $lab)
...
```

Figure 4.4: Interface definitions for describing a CGI-form text

the CGI-from texts WCA shown in Figure 3.1-b. Its schema is the relational model, and thus can be described by one interface definition.

Let us return to Figure 4.2-c. From this figure, for a given interface definition, we can represent its object instance and method invocations as follows:

Object identifier(OID) Each object has its own OID. For example, $\$D1$ denotes an object identifier of the interface definition $Department$. (The object identifiers shown in this presentation denote logical symbols, not the actual system identifier, which are not accessible by the user.)

Attribute A *text* in $\rightarrow\{text\}$ is used to refer to an attribute value. For example, $\$D1 \rightarrow \{deptName\}$ refers the value held by attribute *deptname* of $\$D1$. In general, values may be either an atomic data (e.g., 'Information System Science') or a set of sub-objects (e.g., [$S0,$S1,$2]) depending on the declaration in its interface definition.

Element number A subscript of multi-valued attribute is enclosed in square brackets $[n]$. Thus, an individual element in a set value can be specified using $[n]$, where n

is the element number. For example $\$D1 \rightarrow \{laboratories\}[0]$ refers to the first laboratory of $D1.

Invocation of an instance method The instance method can be invoked using the notation OID→instance-method (e.g., $D1 →printDetail()).

Invocation of a static method The static method can be invoked using the notation InterfaceName → static-method (e.g.,Department → getByURL('s.html')).

4.2 Path Expressions

Path expressions [18] is a high-level notation for expressing navigation through the object composition. By wrapping a Web document into our common data model, we can use the path expression to specify an element in a nested structure of the Web document. Path expressions may be single valued or set valued. Let us give the definition of path expressions used in our framework as below.

Definition 1. Let $t_0, ..., t_n$ be (not necessarily distinct) interfaces. Suppose that t_{i-1} is described by

interface t_{i-1} **is**

$$[..., A_i \qquad \textbf{refto } t_i, ...].$$

That is, t_{i-1} is the interface with an attribute A_i, where A_i is a single-valued attribute that keeps an OID of object of t_i. Then, a **path expression** on t_0 is given by $t_0 \rightarrow \{A_1\} \rightarrow ... \rightarrow \{A_i\} \rightarrow ... \rightarrow \{A_n\}$ $(1 \leq i \leq n)$. The result of the path expression is the set of objects (or values) of the attribute A_n that can be reached via

the specified attribute chain containing the single-valued attribute $A_1, ..., A_i, ..., A_{n-1}$.

□

Definition 2. Let $t_0, ..., t_n$ be (not necessarily distinct) interfaces. Suppose that t_{i-1} is described by

interface t_{i-1} **is**

$$[..., A_i \quad \textbf{setof } t_i, ...].$$

That is, t_{i-1} is the interface with an attribute A_i, where A_i is a multi-valued attribute that keeps OIDs of objects of t_i. Then, a **path expression** on t_0 is given by $t_0 \rightarrow \{A_1\} \rightarrow ... \rightarrow \{A_i\}[j] \rightarrow ... \rightarrow \{A_n\}$ $(1 \leq i \leq n)$, where j specifies the j-th element in a set of multiple-valued attribute A_i. The result of the path expression is the set of objects (or values) of the attribute A_n that can be reached via the specified attribute chain containing the multi-valued attribute $A_1, ..., A_i[j], ..., A_{n-1}$. □

The following is an example of path expressions from Figure 4.2-c with interface definition shown in Figure 4.3.

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

The above path expression is interpreted to get all full names of the first staff in all laboratories in the department. The $[*]$ is used to specify all elements in a multi-valued attribute. For convenience, this $[*]$ can be omitted.

4.3 Navigational Integration

4.3.1 Definition

As described earlier, the navigational integration is the process to create derived links between two pages of WISs. In general, derived links from one page to another page

represent that elements in these two pages have relationship with each other. Thus, the *navigational integration* is defined below.

Definition 3. Let S and R be WISs (S is called a sender, R is called a receiver). Let x and y be elements of the result pages of S and R, respectively. A **navigational integration** from S to R by creating derived links from x to y is defined as follows:

$$S \Rightarrow R \quad \stackrel{def}{=} \quad f(S' \rightarrow ... \rightarrow \{x\}) = R' \rightarrow ... \rightarrow \{y\},$$

where S' and R' are the interface definitions wrapping the result page of S and that of R, respectively. In addition, f is *relationship* between x and y.

Currently, we restrict navigational integration in such a way that an element x of a WIS is specified by a path expression; furthermore, we assume that an *element* in navigational integration must take an atomic value. By these restrictions, we can avoid ambiguity which would occur when considering integration from one list to another list.

4.3.2 Modes of Navigational Integration

From the above definition, we can conclude that navigational integration can be implemented if the relationship between elements in two WISs can be described in terms of WISs' properties. The relationship between elements described above can be expressed using two ways: (i) server-access method, and (ii) element comparison. The first way is the relationship that an element of one WIS is passed as an argument of a static-method of the interface definition of the other WIS. The second way is the relationship that an element of one WIS is compared with an element of the other WIS by a comparison operator.

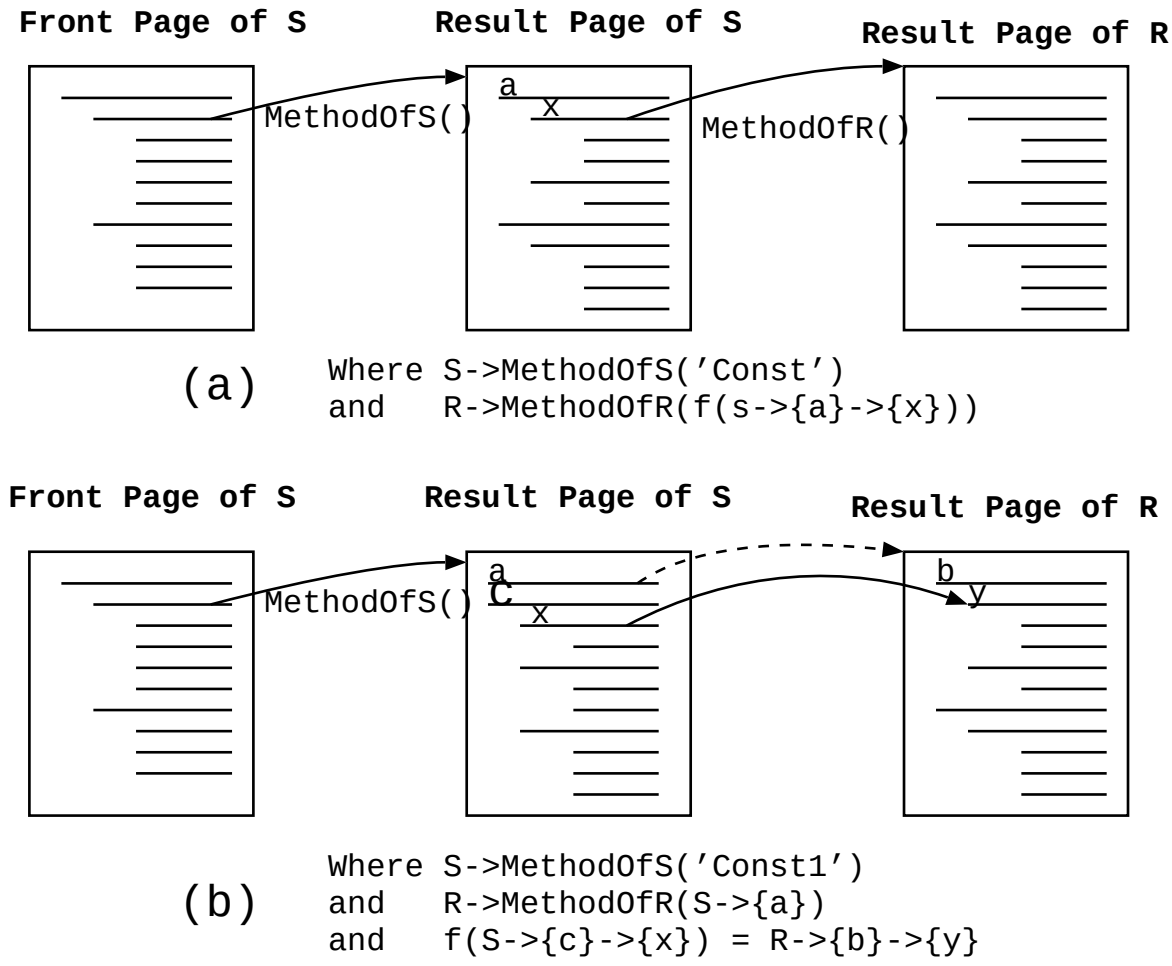


Figure 4.5: Modes of navigational integration

To describe navigational integration in a declarative form, we use a query command whose WHERE clause specifies the predicates that describe 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 described 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 Figure 4.5-a or 4.5-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 4.5 illustrates these two modes with their WHERE clauses. These modes are as follows:

- The first mode (Figure 4.5-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.

- The second mode (Figure 4.5-b) is to embed both 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 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. We will describe a syntax of a query command and how to execute the query in detail later.

4.3.3 Implementation

In general, the relationship between the elements of S and the elements of R can be one of three types: (i) one-to-one, (ii) many-to-one, and (iii) one-to-many. In implementation, if the relationship is one-to-many, an additional feature is required to help users understand the relationship between elements more clearly. Figure 4.6 classifies possible cases of navigational integration between the result pages (from S to R). We can divide two modes of navigational integration into four cases as follows:

Case 1: This case is the first mode (described in section 4.3.2) that is used when R provides an access method requiring an entity y (e.g., a CGI-form text). In this case, the relationship between x and y is either one-to-one or many-to-one. One element of x is mapped to one element of y.

Case 2: This case is the same mode as the first case except that the relationship between x and y is one-to-many. A single element of x is mapped to multiple elements of y.

Case 3: This case is the second mode (described in section 4.3.2). In this case, the relationship between elements is either one-to-one or many-to-one. One element

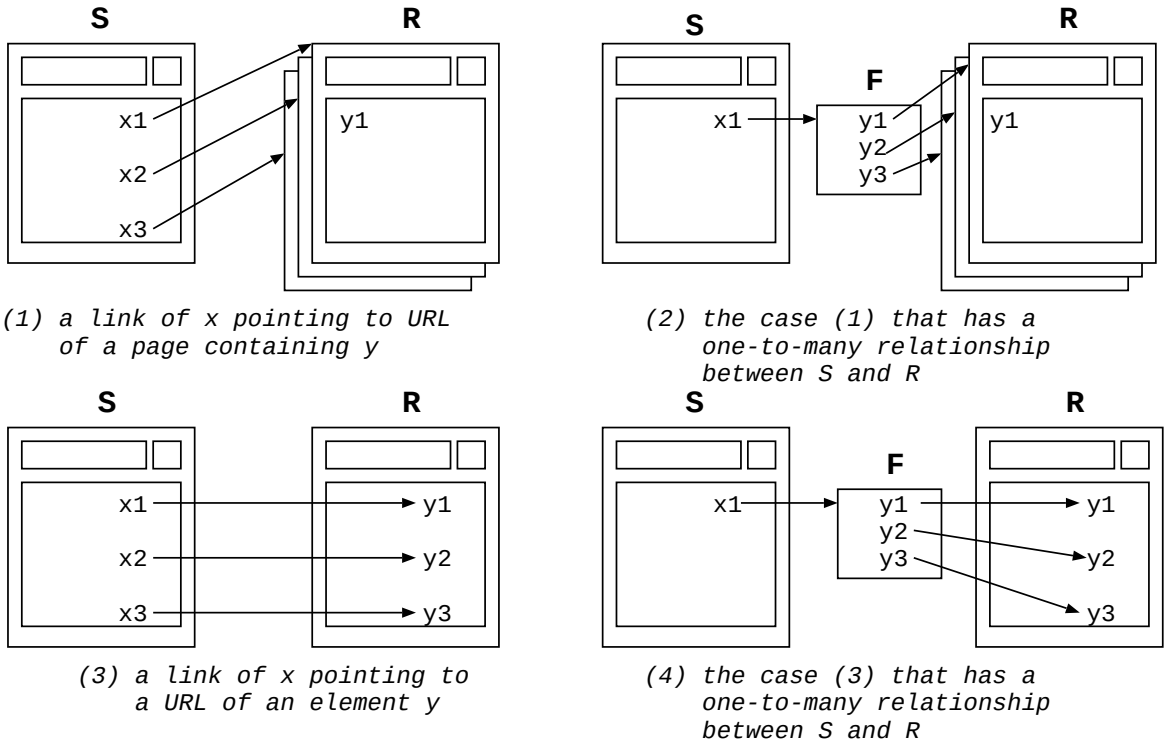


Figure 4.6: Possible cases of navigational integration

of x is mapped to one element of y.

Case 4: This case is the same mode as the third case except that the relationship between x and y is one-to-many. A single element of x is mapped to multiple elements of y.

As shown in Figure 4.6, a mapping table F is shown to users to explain the one-to-many relationship from $S \rightarrow \{x\}$ to $R \rightarrow \{y\}$ due to the relationship between them.

4.4 A Query Language for Navigational Integration

Here, we are ready to demonstrate how navigational integration is described. We present a SQL-like query language designed for creating derived links of navigational integration between WISs based on our common data model. It has a following form:

select *
from *object-variable* in *interface-name*
[,*object-variable* in *interface-name*]
source *interface-name* of *database-name* on *address*
[, *interface-name* of *database-name* on *address*]
[**where** *object-condition* or *method-call*]

From clause contains a list of *object-variables* of interface definitions. *Object-variable* is bound to object collection in *interface-name*. **Source** clause contains the address and database of the interface definition specified in **from** clause. **Where** clause contains query conditions. There are two types of the query condition: an object condition and a method-call condition. The object condition is used for making derived links between elements of different WISs (as shown in case (3) and (4) of Figure 4.6)

while the method-call condition is used for making derived links between elements and pages (as shown in case (1) and (2) of Figure 4.6)

Assume that a mobile user finds the two WISs (WIS1 and WIS2 in Fig.3.1-a and 3.1-b whose interface definitions are shown in Figure 4.3 and 4.4) and wants to do navigational integration of them (WIS3 in Fig.3.1-c). Then, the following command performs navigational integration from a given (single) department page of the first WIS (whose url = `URLx`) 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:LabDomain2Lab(
           D->{laboratories}->{labName})))
```

For example, in the above SQL-like statement, *getByLaboratory* is the method defined in the interface definition of WCA of the second WIS (see Figure 4.4); this method is the major function of this WCA. Its argument requires data whose domain is *JLab*. Because the domain of $D \rightarrow \{laboratories\} \rightarrow \{labName\}$ is *LabDomain*, the mediator finds out this conflict, and then inserts functions *LabDomain2Lab* and *Lab2JLab* to convert data from the domain of $D \rightarrow \{laboratories\} \rightarrow \{labName\}$ to the domain of the argument of the function *getByLaboratory*. In this example, the symbol "&L:" means that the Lab2Jlab is the function provided by the MIR of *Department* and "&D:" means that the LabDomain2JLab is the function provided by the MIR of

LabBuilding. These two functions can be regarded as the relationship between those elements in the result pages of the two WISs. We will describe the domain conversion in the next chapter.

4.5 Mediator

Let us recall that the mediator (located on the user's computer) generates the above command by interacting with the user. To help a user issue this request easily[22], we implemented a user interface of the mediator as shown in Figure 4.7. This mediator explores information downloaded from cells and interacts with the user to generate a query command.

In a process of generating a query command, first, the user chooses the database and the interface name he wants. Then, the mediator will show the attributes and methods of the selected interface definition and generate items for *from* and *source* clauses. Next, the user can choose the attribute and the method. When the user selects the method shown in the list, the mediator will check the arguments of the selected method, provide hints, and request the user to input data for each argument. If the domain of the input data and the domain of the argument are different, the mediator performs semantic conflict resolution and inserts conversion functions to match their domains. The conversion functions that are inserted for resolving semantic conflict are also determined by the mediator using the information of sub-part of the domain hierarchy that are downloaded together with the interface definitions. The detail of algorithm of semantic conflict resolution, domain hierarchy and download mechanism will be described in the next chapter.

After the user completes his requirement, the mediator will translate the data in

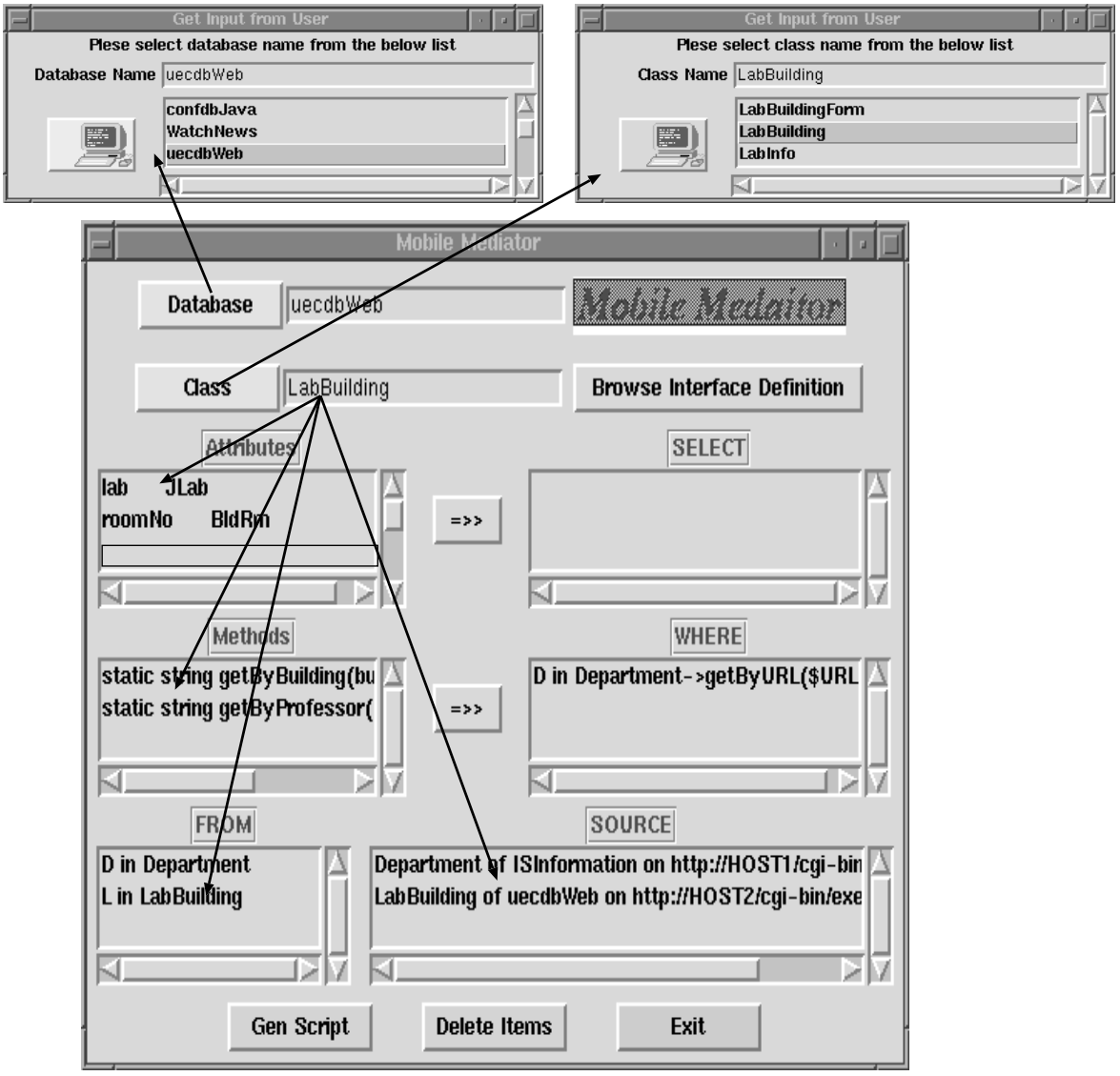


Figure 4.7: An interface of a mediator on a mobile unit

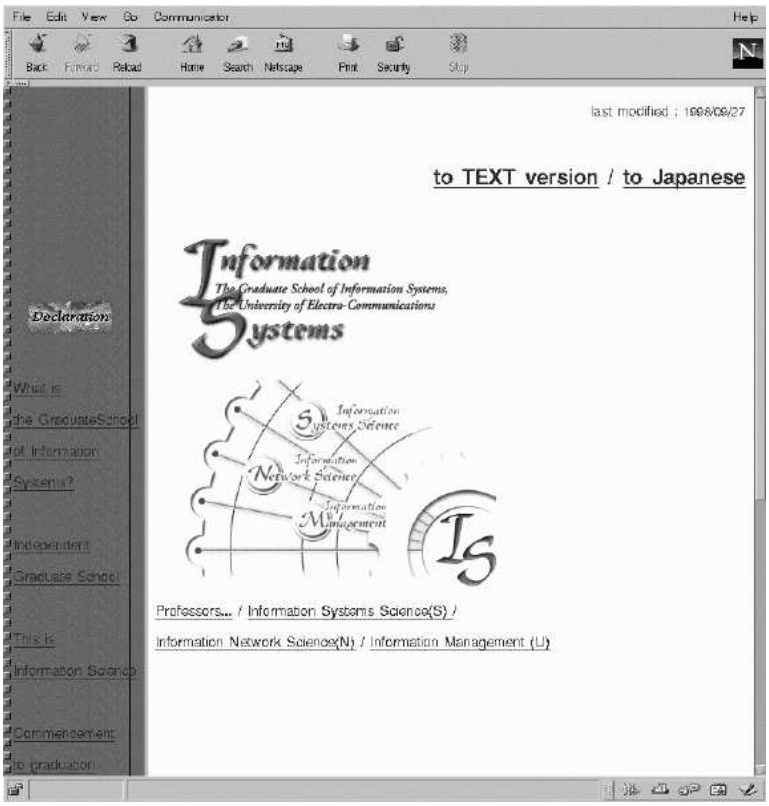
the form of the user interface into a query command and automatically embed this command into the front page of WCA of the first WIS through a subroutine defined in the interface definition of the department page. Figure 4.8 shows the front page and its HTML source with the embedded query command shown in the previous section.

4.6 Proxy Executor and Wrapper

In previous sections, we described how to generate a new customized WCA. In this section, we will describe how to execute it. Let us recall the execution steps in navigational integration using Figure 4.9. First, the user loads the new front page with the embedded query command into his browser. When the user activates an embedded link, the query command is sent to a proxy executor (step 1). The proxy server generates an execution plan by analyzing the query into sub-queries, registers the execution plan as an executable persistent object [12], and sends the first query to the MIR of WIS1 (step 2). The intermediate result is sent back to the user at the step 4 of Figure 4.9-a. The result page is shown in Figure 4.10. As shown here, the next step of the execution is nothing but a derived link, and this is implemented as a CGI link to activate the wrapper of WIS2. In Figure 4.9-b, after the user selects the link, the executable persistent object in the proxy executor is activated to continue the process (step 5). The next execution is executed and the result page of the second WIS is accessed and returned to the user again (step 6-8). The result is shown in Figure 4.11.

Note that, in this implementation, we set the proxy executor of the first WIS to control overall processing. Thus, *cell1* in Figure 4.9 performs the task of the cell0.

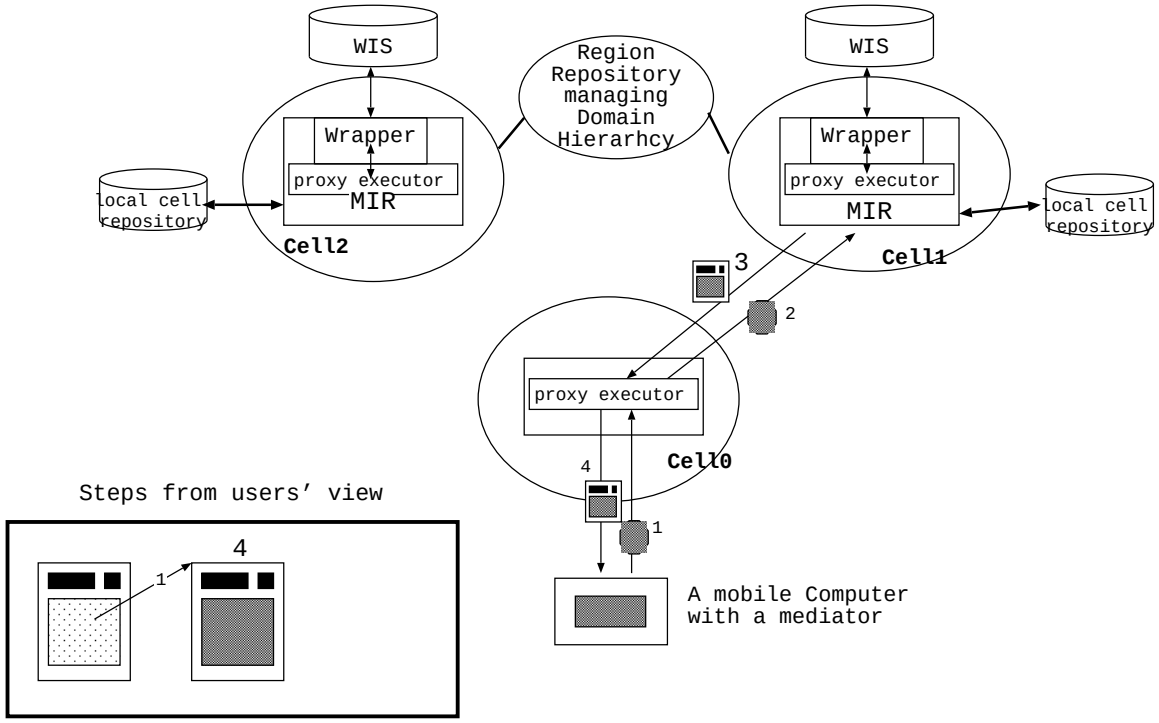
Now, we will describe modules of a wrapper and a proxy executor in a MIR, and describe how they execute a query command. Figure 4.12 shows modules in the wrapper



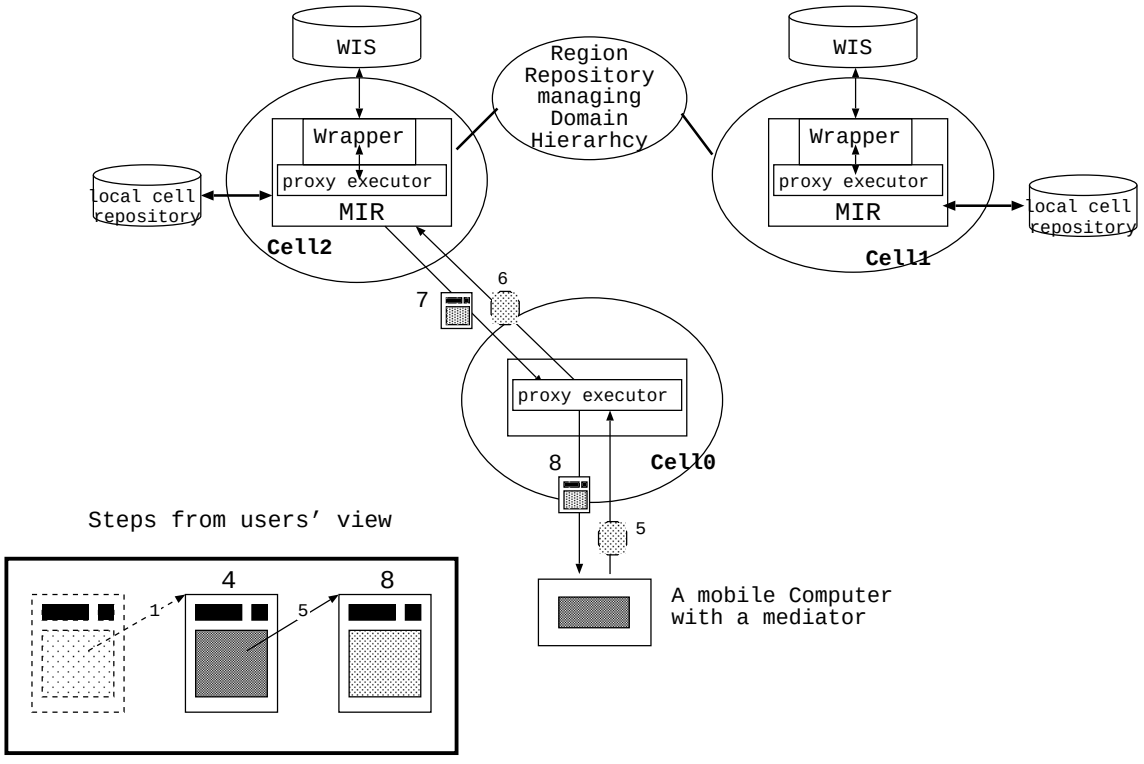

```
<!DOCTYPE HTML PUBLIC "-//IBM//DTD HPB HTML//EN">
<html>
<head>
<title>IS HP Top Page (ENGLISH)</title>
</head>
<body background="../images/bgd/is-bgd1.gif" bgcolor="#FFFFFF">
<div align="right">
...
<table width="640" cellspacing="0" cellpadding="0">
  <tr>
    <td valign="top" width="480">
    ...
    ...
    <a href="http://HOST1/cgi-bin/execute1.pl?
      URLx=http://xxx/S.html&
      PPSQL=
      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($URLx)
      and    LabBuilding->getByLaboratory(
               &L:Lab2JLab(&D:LabD2Lab(
      _ _ _ _ _ _D->{laboratories}->{labName})))"> _ _ _
      Information Systems Science(S) </a>/<br> <a
      href="http://HOST1/cgi-bin/execute1.pl?
      URLx=http://xxx/N.html&
      PPSQL=
      select *
      from   D in Department, L in LabBuilding
      source Department of ISInformation
          on http://HOST1/cgi-bin/execute.pl,
             LabBuilding of uecdbWeb
          on http://HOST2/cgi-bin/execute.pl
      where  DepartMent->getByURL($URLx)
      and    LabBuilding->getByLaboratory(
               &L:Lab2JLab(&D:LabD2Lab(
      _ _ _ _ _ _D->{laboratories}->{labName})))"> _ _ _
      Information Network Science(N)</a> /
    ...
  </tr>
</table>
</body>
</html>
```

a link embedded with a query command

Figure 4.8: A front page and its source with embedded query commands

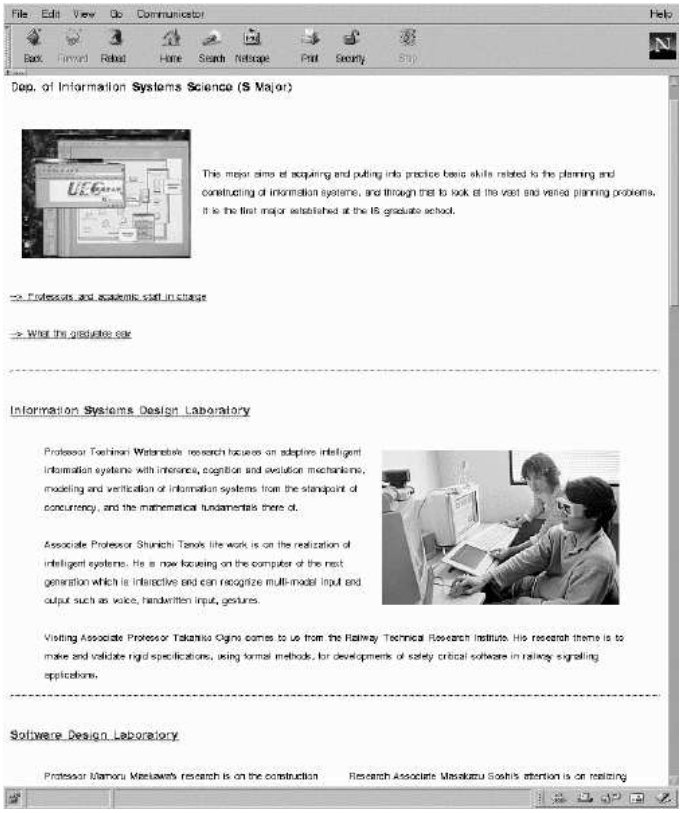


(a) The Execution Step in Navigational Integration (1)



(a) The Execution Step in Navigational Integration (2)

Figure 4.9: Execution Step in Navigational Integration




```
<!DOCTYPE HTML PUBLIC "-//IBM//DTD HPB HTML//EN">
<html>
<head>
<title>Information Systems Science</title>
</head>
<body bgcolor="#FFFFFF">
<!-- begin Department Object-->
<h2><A Name="908454679-0-deptName">Dep. of Information Systems Science
</A>(S Major)</h2> <div align="center"><center>
.....
.....
.....
<!--begin Laboratoy Object-->
<h3><A Href="http://HOST1/cgi-bin/execute.pl?PPCode=PPCode1035&Step=2&
Method=getDocument&Bind_1=Information System Design">
<A Name="908454679-1-labName">Information Systems Design Laboratory</A></A>
</h3> <div align="center"><center>
<table width="90%" cellspacing="0" cellpadding="0">
  <tr>
    <td valign="top">
    <img src="http://www.is.uec.ac.jp/images/photo_zu/s-jikken2.JPG"
    alt="s-jikken2.JPG" align="right" border="0" hspace="8" vspace="8"
    WIDTH="280" HEIGHT="183">
    <!-- begin Staff Object-->
    <A Name="908454679-2-position">Professor</A>
    <A Name="908454679-2-fullName">Toshinori Watanabe</A>'s research
    focuses on adaptive intelligent information systems with
    inference, cognition and evolution mechanisms, modeling and
    verification of information systems from the standpoint of
    concurrency, and the mathematical fundamentals there of.<br>
    <br>
    <!--end Staff Object-->
    <!--begin Staff Object-->
    <A Name="908454679-3-position">Associate Professor</A>
    <A Name="908454679-3-fullName">Shunichi Tano</A>'s life work is
    on the realization of intelligent systems. He is now focusing on
    the computer of the next generation which is interactive and can
    recognize multi-modal input and output such as voice, handwritten
    input, gestures.<br>
    <br>
    <!--end Staff Object-->
....
....
</table>
```

a data tag embedded by the specific wrapper

a execution tag embedded by the general wrapper in the proxy server

a absolute tag converted by the wrapper

a object tag embedded by the specific wrapper

Figure 4.10: A result page of the first WIS and its source with embedded tags

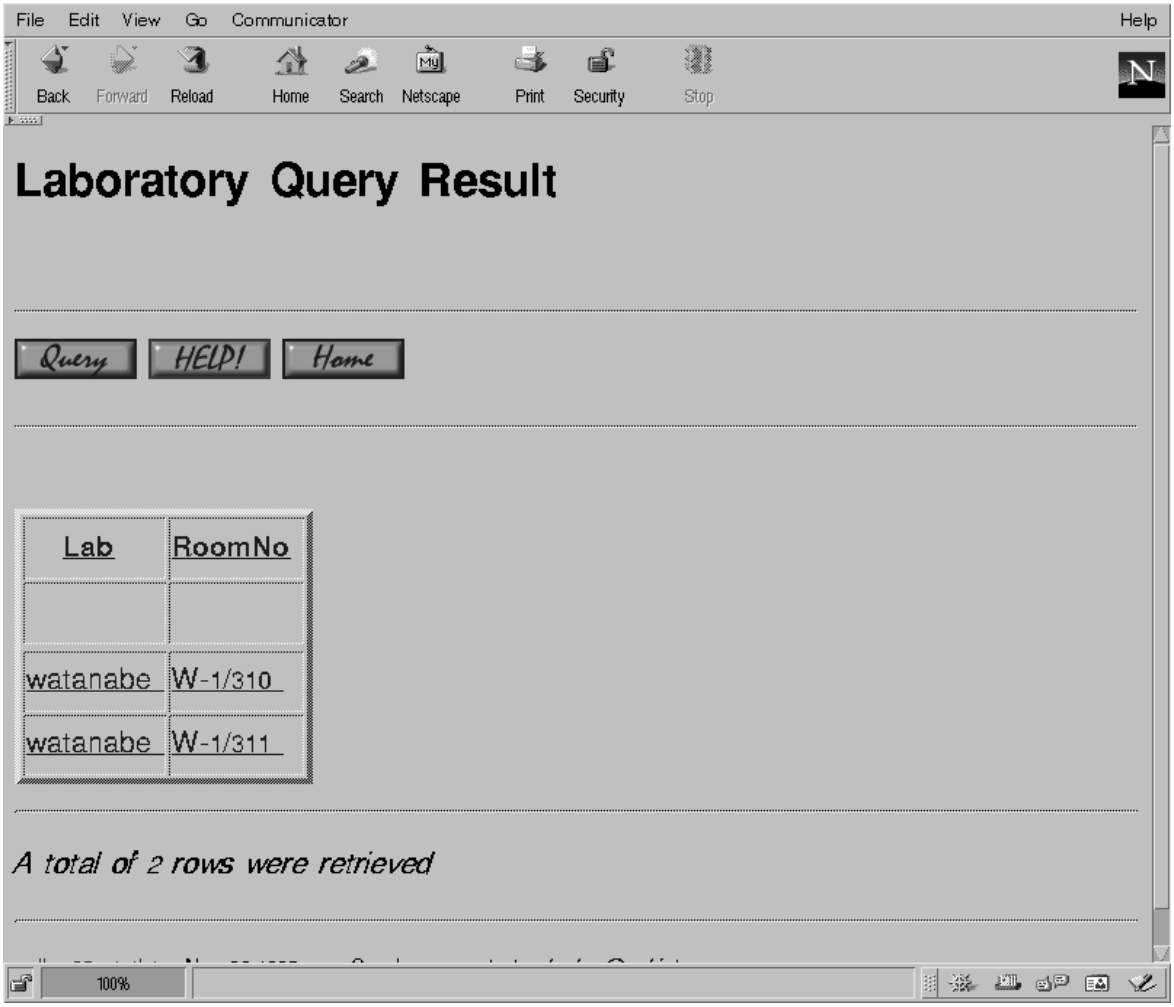


Figure 4.11: A result page of the second WIS

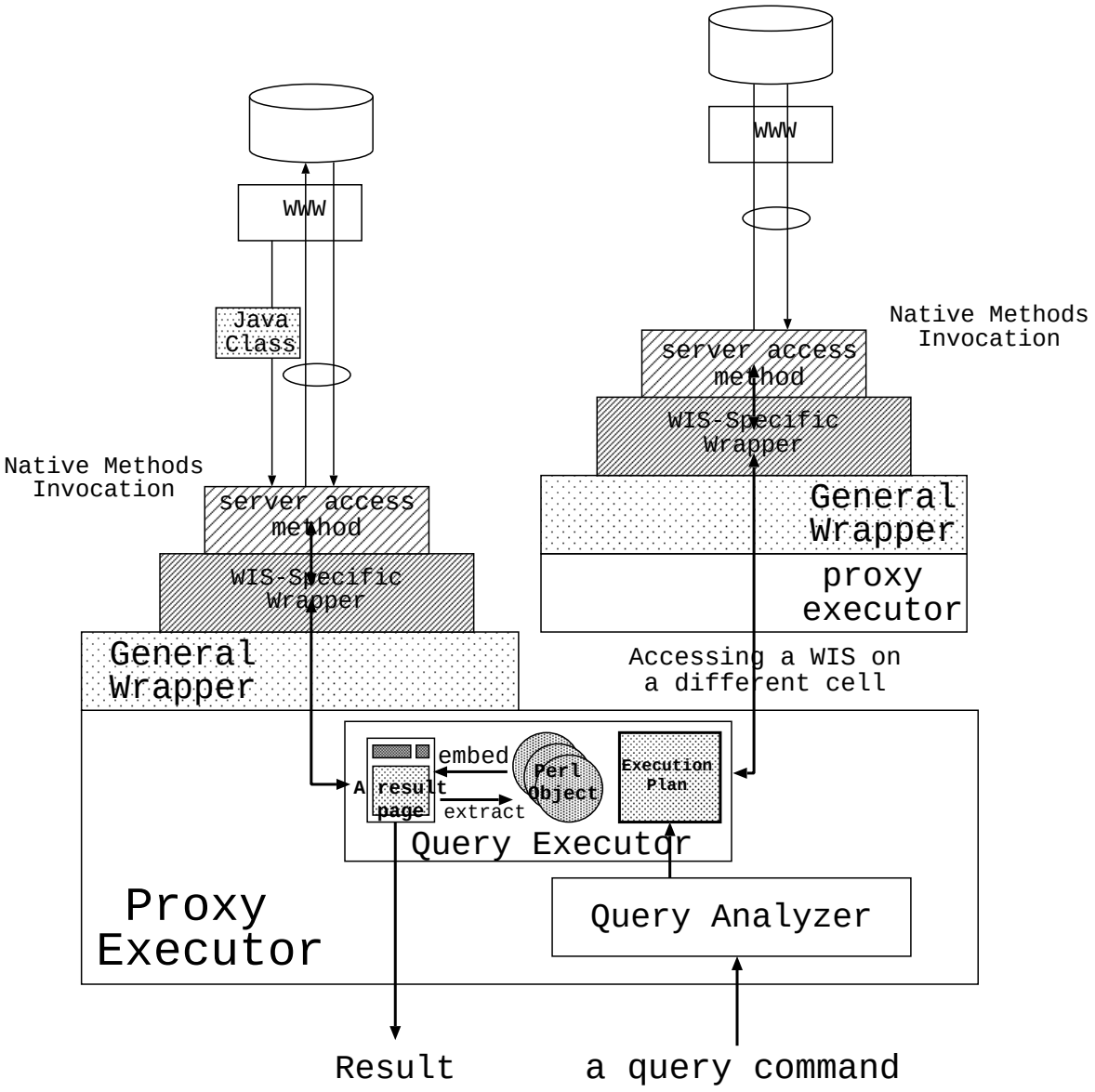


Figure 4.12: Modules in the wrapper and the proxy executor

and the proxy executor in a MIR of a cell. The modules in the proxy executor that cooperate with the wrapper are described as follows:

Query Analyzer: The query analyzer checks an incoming query command (step 1 in Figure 4.9) and builds a query plan in a bottom-up manner. First, an access plan for each WIS is generated. Then, two-way joins are generated repeatedly until a complete plan is built up. Unlike the traditional integration system, the execution plan of a query command of the navigational integration is decided by a sequence of its **where** conditions. Currently, no query optimization to arrange the sequences of information access is performed.

In case that the incoming request is not a query command, the query analyzer will know that it is the request to activate the registered persistent object and pass the request to the query executor.

Query Executor: The query executor is one of the important components of the proxy executor that controls the overall processing of the navigational integration. It performs the following tasks:

- It initializes the environment and registers the query command as an executable persistent object into the system. This persistent object is maintained in the system and will be activated in the next step of navigational integration. (e.g, step 5 in Figure 4.9.)
- It executes the persistent object. It sends each sub-query to the corresponding wrapper and receives the result page.
- Based on the modes of navigational integration, it embeds derived links into the result page and sends the result page back to the user.

Wrapper: The wrapper consists of two sub-modules (i) a **general wrapper** and (ii) several **WIS-specific wrappers**. Each WIS-specific wrapper is implemented corresponding to the standard interfaces defined in the general wrapper.

With the interface definition, the WIS-specific wrapper performs the following tasks:

- It converts a sub-query sent from the query executor of the proxy executor into a native command of the corresponding Web client application.
- It executes the native command to retrieve a result page, extracts the data in the result page, and converts the extracted data into objects represented in the common data model. After that, the WIS-specific wrapper embeds the objects into the result page of the WIS for marking the positions of data. With these data positions, the query executor of the proxy executor can determine where the derived links should be embedded.
- It returns the objects and the result page to the query executor.

Implementation of the wrapper and the proxy executor is not so different from the other integration systems. The additional operation is to embed objects and derived links into result pages. The details of the concept of implementation can be seen in [41, 42]

4.7 Summary

In this chapter, first, we have proposed the common data model for modeling a Web information source. Our common data model is a variant of object-oriented model enhanced with a concept called *domain hierarchy*. The common data model can model

nested structures of data in a Web document and server methods of the WIS. Data extracted from a Web document by a wrapper are mapped into the objects in our common data model. Second, path expressions for the common data model are also proposed for expressing navigation through the object composition. By wrapping a Web document into our common data model, we can use the path expression to specify an element in a nested structure of the Web document. Third, the general definition of navigational integration is also proposed. The navigational integration in the WWW framework is defined as the process of creating derived links between Web pages. We represent how to describe the navigational integration in terms of path expressions. Fourth, to help users specify navigational integration easier, the query language and the mediator on mobile computers are proposed. An example of navigational integration is also demonstrated. Finally, we have described the main modules in the MIR for executing navigational integration.

Chapter 5

Domain Hierarchy for Resolving Semantic Conflict

Various projects on the integration systems of heterogeneous resources and Web site management systems resolve the semantic conflict problem among the well-known information sources by allowing developers to define an integrate view on a centralized server. This approach will work well on a static environment of an organization that is equivalent to a cell in our approach.

How can mobile users integrate information by avoiding the semantic conflict problem even though the range of integration covers different cells? The research [11] has proposed the concept called an *ontology*. *The ontology* is an agreement on the terminology of vocabulary among autonomous organizations to permit information of heterogeneous resources to be integrated. We apply the concept of the ontology and extend it for supporting information integration among *autonomous distributed organizations*. This chapter presents such an ontology that we call a *domain hierarchy*

5.1 A Domain Hierarchy

In a wide-area system, the semantic conflict between data provided by different WISs must be solved. As described earlier, we construct the system architecture of our

approach shown in Figure 5.1. This system contains multiple subsystems called *cells*. A *cell* is an independently-developed system that manages multiple WISs and provides access to these WISs through wrappers in our approach. Furthermore, multiple cells form a *region*, which is designed for providing semantic convertibility of information between its cells so that they can exchange data among themselves. To achieve a semantic convertibility, we propose a multilevel data representation called a *Domain Hierarchy*. A *domain* in our approach refers to a description of a pool of values that have an exact data-presentation and meaning.

Before describing how to resolve semantic conflict between data of different WISs, we describe the detail of the domain hierarchy. Figure 5.1 shows a domain hierarchy in a region. The domain hierarchy contains two components as follows:

1. A common domain hierarchy (CDH): it is a set of domains that are used to define the representative data, in meaning and format, shared by all cells in the same region.

2. A local domain hierarchy (LDH): each cell maintains one LDH. The set of domains in each LDH are used to define data, in meaning and format, appeared in WISs that are maintained in its cell.

In each cell, the schema and access methods of WISs managed in the cell are wrapped by unified forms called *interface definitions*. Using this wrapping, domains of both attributes and arguments (of access methods) are mapped to the domains in the local domain hierarchy of the cell. Moreover, the domains in each LDH are linked to the domains in the CDH for providing semantic convertibility among data in different cells.

In practice, the CDH is maintained in a cell repository of one cell in the region that is selected as a primary cell. In the other cells, each of them maintains its own LDH and the copy of the CDH that is distributed from the primary cell. Certainly, the primary cell also maintains its CDH.

5.2 Construction

Now, we describe how to build a domain hierarchy. Figure 5.2 shows an example of a domain hierarchy in a region consisting of two cells. Each domain has relationships with other domains. The relationships used in the domain hierarchy can be divided into two types:

1. *one-to-one relationship:* For two given domains P and Q, there is a one-to-one relationship between P and Q when there exists a value in one domain for every value in the other domain and vice versa. This type of relationship always represents an equivalence relationship whose data have the same meaning but are represented in the different formats.

2. *many-to-one relationship:* For two given domains P and Q, there is a many-to-one relationship from P to Q when multiple values of P are associated with a value of Q. On the other hand, we can say that there is a one-to-many relationship from Q to P.

   With this type of relationship, the relationship between P and Q may be an $is-a$ relationship, i.e., all the instances of P can also be regarded as belonging to Q. The relationship between Q and P may be a *contain* relationship, i.e., an instance of Q contains multiple instances of P as its subpart objects.

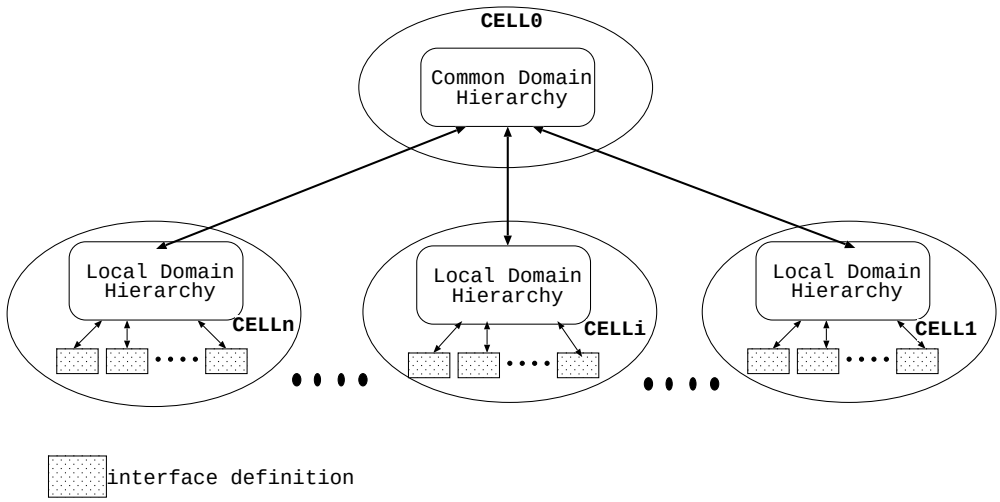


Figure 5.1: A domain hierarchy in a region

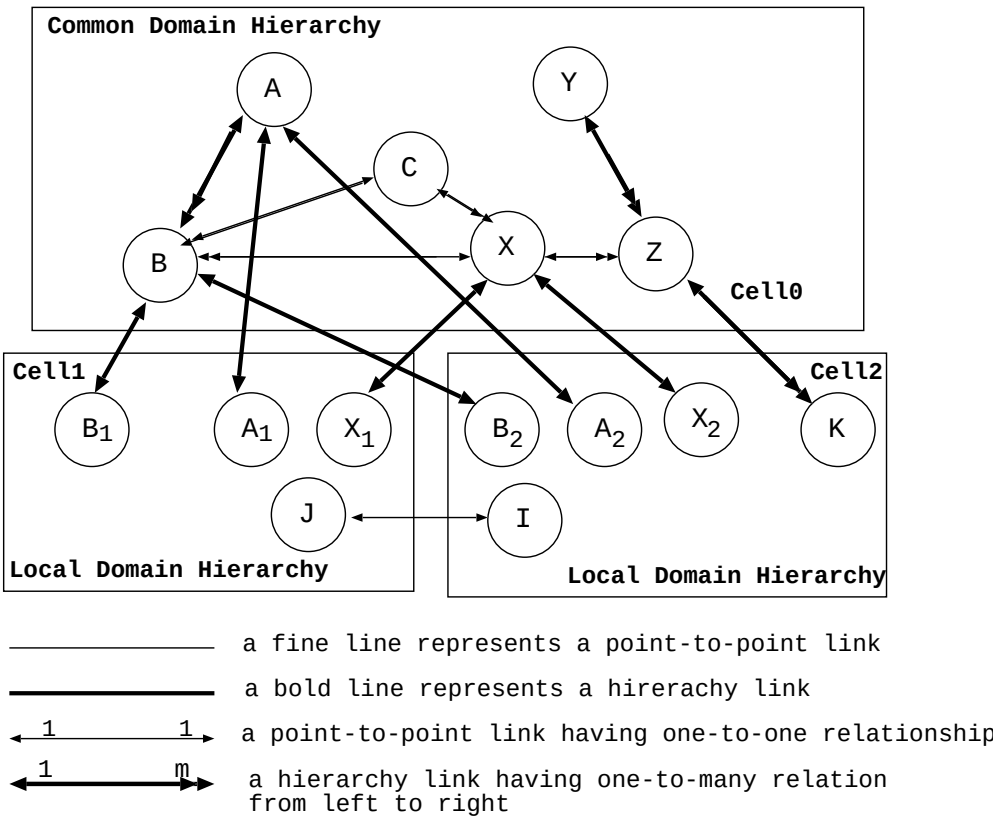


Figure 5.2: A concept of a domain hierarchy

Domains in our approach are organized into a hierarchical structure, where a type of hierarchical relationship may be a *one-to-one* relationship (data-format conversion) or a many-to-one relationship (an *is-a* relationship, or a *contain* relationship). Therefore, for any given domain D, its parent domain is equal to, is more general than, or contains D.

As the step for selecting a parent domain, the type of hierarchical relationship should be selected in the following order: (i) a one-to-one relationship, and (ii) a many-to-one relationship. The *hierarchy links* (depicted by bold links in Figure 5.2) represent such hierarchical relationships.

As a result, for any given domain D, D has a hierarchy link 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 5.2) between D and D_i'. Therefore, D can have one hierarchy link to its parent, and have several point-to-point links to D_i'.

In implementation, each link has conversion functions for converting semantics between the two associated domains. For a given domain D, D has a domain rule. The *domain rule of* D is a set of rules that describe: (i) the facts about D, (ii) the hierarchy link to the parent domain, and (iii) the point-to-point links to the other domains. The syntax and meaning of the domain rule will be described later.

To understand how to create a domain hierarchy more clearly, let us consider an example shown in Figure 5.2. Suppose that the domain A_1 has a *one-to-one* relationship with the domain A. It means that data of A and data of A_1 have the same meaning but are represented in different formats. A is selected as the parent domain of A_1. Suppose that one data-unit of A *contains* many data-units of domain B, we can set the domain

A as the parent domain of B. Suppose that the domain Y has an *is-a* relationship with the domain Z, then Y is set as the parent domain of Z. These relationships are represented by the hierarchy links as shown in Figure 5.2.

Note that at B, it also has relationships with the domains C and X. Because the hierarchy links of B is already set, the relationships to C and X are represented by the point-to-point links.

In the above consideration, there are some points we should note as follows:

- In practice, a hierarchy link from a domain in LDH to a domain in CDH should be chosen primarily from among one-to-one relationships because we expect that the domains in the CDH are set as the standard domains for representing standard conversion formats. In case that there is not any domain in CDH that has the one-to-one relationship with the domain in LDH, a domain in CDH that has an *is-a* or a *contain* relationship will be chosen as a parent of the domain in LDH. In Figure 5.2, the domain K and Z are in this case.

- A hierarchy link and its conversion functions are described in a child domain of the link. This makes distinct separation between the LDH and the CDH.

- A point-to-point link and its conversion functions are described in both domains that are associated with this link if these domains are managed in the CDH or in the LDH of the same cell.

- If required, a point-to-point link can be created between domains of different cells. In Figure 5.2, the domain J and I are in this case.

- A point-to-point link and its conversion functions are allowed to be described only in one domain if the other domain is in a different cell. This is because we

cannot force the cell manager of the other side to describe such a relationship. In this situation, there are some cases where the conversion path cannot be found because reachability between two domains is degraded. We will discuss this in detail later.

Using the information in the domain rule, the system can determine what functions should be provided to convert semantic of data from one domain to the other domain.

So far, there are important points we should note as follows:

1. The level of hierarchy should not be kept so deep (three or four is reasonable) in order to reduce cost of conversion processing.

2. The CDH is the shared part of all cells in the region while the LDH is the private part of each cell. In each cell, its administrator maintains the copy of the CDH of its region and also maintains its LDH in the MIR of his cell.

3. Making the distinction between the CDH and the LDH and putting the conversion task between the two layers to the LDH makes the system easy to be maintained and gradually extended. The number of domains in the CDH and the LDH are less than that in a corresponding global schema that maintains all domains of the region. The number of domains in each cell is dependent on the number of domains of the CDH and its LDH. When the CDH is updated, maintenance of all cells is required by copying the CDH to all cells in the region. A new cell can be added into the region by sharing the same CDH. A cell can add a new domain by linking the new domain with a corresponding domain in the CDH without informing this changing to the other cells. A domain or a cell can be deleted easily, but the point-to-point links that are defined in (domains of) the

other cells will be corrupted. It is the responsibility of the cell administrators to maintain them.

5.3 Package Download

In chapter 3, we have described that the functions of the mediator in our approach consist of: (i) downloading necessary information from cells, (ii) resolving semantic conflicts among data of different WISs, and (iii) generating a new customized WCA to perform navigational integration. In chapter 4, we have described the third function and have given the rough overview about the first function and the second function. In this section, we will give the detail of the first function. After that, the second function will be described in details.

To integrate information from several information sources, necessary information of desired WCAs must be packed into packages and be downloaded from cells. Now, let us give some preparations for defining a package of WCA. We use Figure 5.3 to explain the definitions.

Definition 1. A *WCA-domain set* of a given WCA w is the set of domains $\{d_1, d_2, ..., d_n\}$ that are directly used in the interface definitions wrapping the WCA w. □

For example, in Figure 5.3, the WCA-domain set of the right-hand side WCA consists of domains A_2, X_2, and K. These domains are attached to the attributes or the arguments of the methods in the interface definitions of this WCA.

Definition 2. A *hierarchy-domain set* of a domain d_0 is a set of domains $\{d_0, d_1, d_2, ..., d_i, ..., d_n\}$ where d_i is the parent domain of domain d_{i-1} (i = 1,..., n), and d_n is the highest level of parent domain of d_0. □

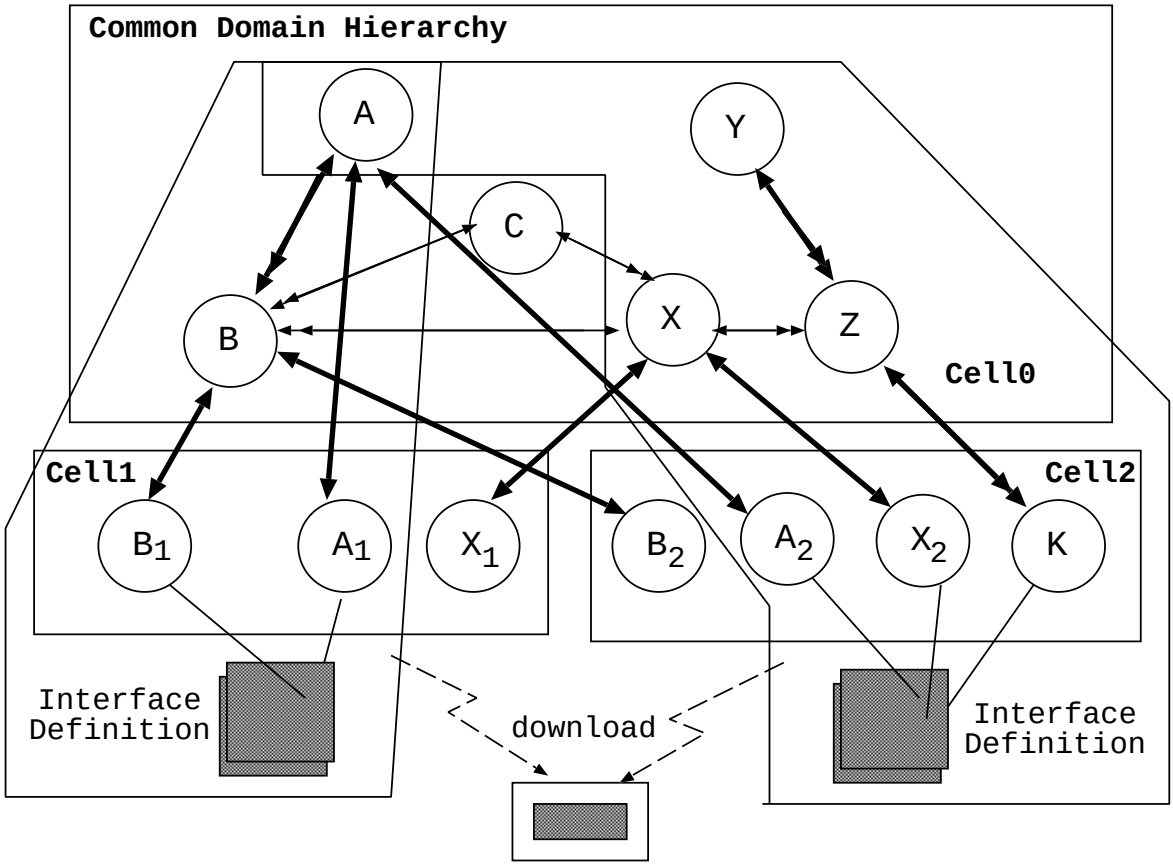


Figure 5.3: Downloading package

For example, in Figure 5.3, $\{A_2, A\}$, $\{X_2, X\}$, and $\{K, Z, Y\}$ are hierarchy-domain sets of the domain A_2, X_2, and K, respectively.

Definition 3. A *WCA package* of a given WCA w consists of: (i) a front page of the WCA, (ii) a set of interface-definitions wrapping the WCA w and (iii) the domain rules for the hierarchy-domain sets of the WCA-domain set of the WCA w. □

For example, in Figure 5.3, the WCA package of the right-hand side WCA consists of a front page, interface definitions and the domain rules for $\{ A_2, A, X_2, X, K, Z, Y\}$.

For any given WIS, its WCA-package contains all the information that the mediator must download from a cell. After downloading the WCA-packages from cells, the user interacts with the mediator to create a query command. However, the semantic conflict occurs when he tries to join two data that have different domains. Because of this problem, there arises a further requirement that the mediator must resolve the semantic conflict among the domains by using the downloaded subparts of the domain hierarchy in the WCA packages.

In the next section, we will describe how the mediator works to resolve semantic conflict by finding conversion paths between two given domains. First, we will describe how conversion paths can be found out based on reachability of two given domains. Next, to get reasonable conversion paths, the legality of conversion paths will be described. Based on the reachability and the legality of conversion paths, finally, we will describe the algorithm of semantic conflict resolution for finding the conversion paths.

5.4 Reachability of Conversion Paths

Now, we are ready to give a definition of a conversion path as follows:

Definition 4. Let hd_{X_1} and hd_{Y_1} be the hierarchy-domain sets of a domain X_1 and a domain Y_1, respectively. Then, a *conversion path* between two given domains, X_1 and Y_1, is a path of sequence of domains,

$$X_1, ..., X_i, SD, Y_j, ..., Y_1 \ (1 \leq i \leq n \text{ and } 1 \leq j \leq m),$$

where

1. $\{X_1, ..., X_i\} \subset hd_{X_1}$ and each X_{k+1} is a parent domain of X_k $(1 \leq k \leq i-1)$ via a hierarchy link, and

2. $\{Y_j, ..., Y_1\} \subset hd_{Y_1}$ and each Y_{m+1} is a parent domain of X_m $(1 \leq m \leq j-1)$ via a hierarchy link, and

3. SD (called a *shared domain*) is a domain that has links with X_i and Y_j. □

As shown in Figure 5.4, for two given domains, X_1 and Y_1, there are three cases of the shared domain:

1. A domain that exists in hierarchy-domain sets of both sides. In Figure 5.4, the domain G is in this case.

2. A domain that exists in the hierarchy-domain set of one side and has a point-to-point link with a domain in the hierarchy-domain set of the other side. In Figure 5.4, the domains X_i and Y_i are in this case.

3. A domain that has point-to-point links with domains in the hierarchy-domain sets of both side. In Figure 5.4, the domain D is in this case.

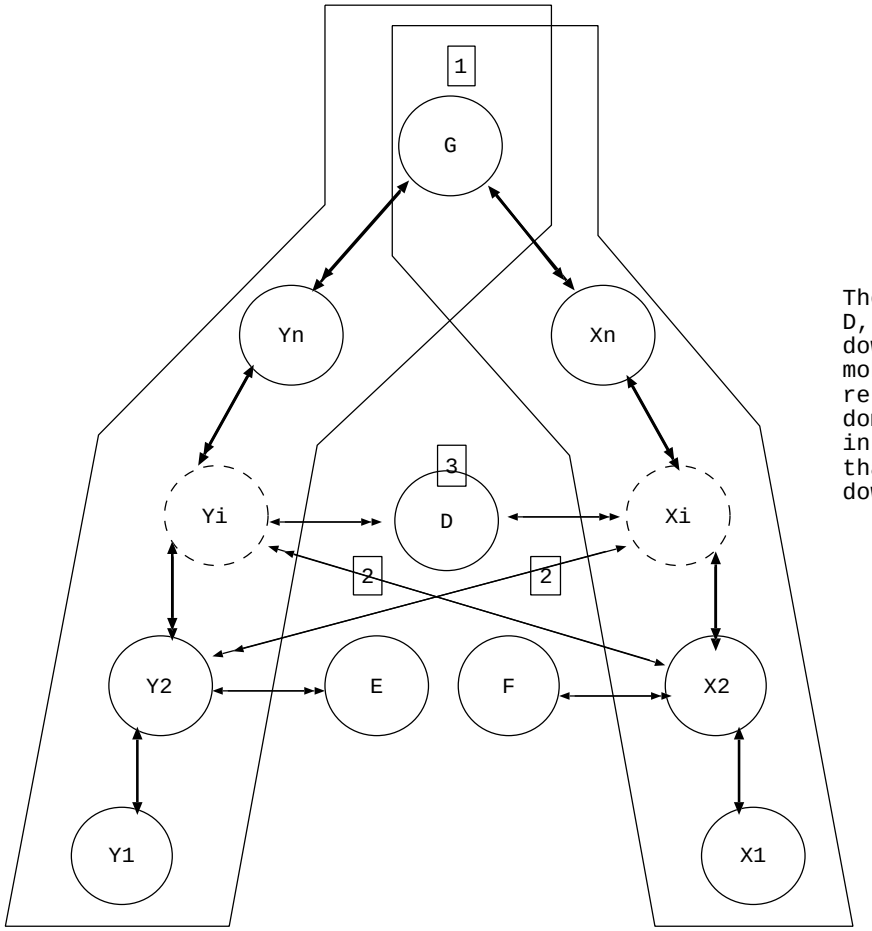


Figure 5.4: A general concept of matching process

For example, returning to Figure 5.3, the semantic conflict between the domain A_2 and the domain A_1 can be solved by converting data of A_2 to data of A_1 through the following conversion path:

$$A_2 \rightarrow A \rightarrow A_1$$

where A is the domain that exists in hierarchy-domain sets of both sides.

Similarly, the conversion from data of the domain X_2 to data of the domain B_1 can be performed through the following path:

$$X_2 \rightarrow X \rightarrow B \rightarrow B_1.$$

This corresponds to the second case.

5.5 Legality of Conversion

As described earlier, the semantic conversion can be performed through one of the conversion paths between two domains. In general, it would be simply to explore all possible paths and return the one with the shortest length. This approach would work well if we did not take account of the correctness of conversion.

However, unexpected data emerge when the conversion along the conversion path causes an indirect many-to-many relationship between data of a starting domain and data of an ending domain. Therefore, an *illegal path* can be defined as follows:

Definition 5. Suppose p is a conversion path between a domain A and a domain Y. If the indirect relationship between A and Y is a many-to-many relationship, then the conversion path p is illegal. Otherwise, such a relationship is legal. □

Table 5.1 shows the types of relationships that are created in domain conversions. The column shows the type of the relationship between a domain A and a domain B (A

$\rightarrow$ B) and the row shows the type of relationship between a domain X and a domain Y (X $\rightarrow$ Y) . Each entry in the table shows the relationship between the domain A and the domain Y according to the conversion A $\rightarrow$ B $\rightarrow$ X $\rightarrow$ Y.

Table 5.1: A combination between two relationships

	one-to-one	one-to-many	many-to-one
one-to-one	one-to-one	one-to-many	many-to-one
one-to-many	one-to-many	one-to-many	many-to-many
many-to-one	many-to-one	many-to-many	many-to-one

From this table, we can say that the indirect many-to-many relationship occurs when the conversion path contains either a relationship one-to-many that follows a relationship many-to-one or vice versa.

5.6 Semantic Conflict Resolution

Until now, the reachability and the legality of conversion paths have been described. Based on these two properties of conversion paths, this section describes the algorithm to find the most reasonable domain-conversion path between domains for resolving semantic conflict. Given two atomic domains A and B, a set of conversion paths (CP) between them is computed by a binary function called *path(A,B)*. In addition, the best conversion path in CP is computed by a unary function called *bestPath(CP)*. Thus, the best paths (BP) between domains A and B can be determined as follows:

$$BP = bestPath(path(A, B)).$$

The $path()$ function on two domains A and B returns a set of *conversion paths* between them and the $bestPath()$ function returns the *best*(and legal) paths between A and B.

In general, all conversion paths are not necessarily legal. We have found that the unexpected data emerge when an indirect relationship between two domains via a conversion path is a many-to-many relationship. The execution of the $bestPath()$ function is based on this policy; it filters out illegal paths and then finds the best path.

Algorithm: Based on the above computation formulation, we have developed and implemented an algorithm to find the best conversion path. First, let us explain the algorithm of *path()*. The algorithm is specified in Figure 5.5.

In Figure 5.5, when a sender domain and a receiver domain are given, lines (1) and (2) generate two lists that contain domains in their ancestor domains named *senderDH* and *receiverDH*, respectively. The name of the parent domain can be found in the *Parent* fact defined in the domain rule of each domain. In line (3)-(5) each domain in *senderDH* calls its method *genSupportDomain()* to find other domains that have point-to-point relationships with it and keeps the list of those domains. The names of domains of point-to-point relationships can be found in the *Support* fact defined in the domain rule of each domain. Likewise, line (6)-(9) do the same work for each domain in *receiverDH*. For a given domain x, let $x.supportDomain$ be a set of domains each of which has a point-to-point link with x. For two given domains x and y, line (10)-(22) find a shared domain (sharedDomain) such that $\{sd|sd \in \{x \cup x.supportDomain\}$ and $sd \in \{y \cup y.supportDomain\}\}$, and then find a path $senderDomain \rightarrow ... \rightarrow x \rightarrow sharedDomain \rightarrow y \rightarrow ... \rightarrow receiverDomain$, and register the path as one of the answers. Line (23)-(36) do the same work as line (10)-(22), but it searches a path starting from the receiver domain to the sender domain. We must search in a bi-directional manner because a point-to-point link between two domains in different cells located in different regions is defined at only one side. Searching can be done

```
     sub path(senderDom,receiverDom){
(1)    senderDH[]   = genAncestorList(senderDom);
(2)    receiverDH[] = genAncestorList(receiverDom);
(3)    foreach d in senderDH {
(4)       d.genSupportDomain();
(5)    }
(6)    foreach d in receiverDH[] {
(7)       d.genSupportDomain();
(8)    }
(9)    foundPath = false;
(10)   foreach x in senderDH[]{
(11)     foreach y in receiverDH[]{
(12)      sharedDomain = findSharedDom(x,y);
(13)      if (sharedDomain){
(14)          path = new Path(senderDomain,x,
(15)                          sharedDomain,
(16)                          y,receierDomain,
(17)                          'senderSide');
(18)          foundPath = true;
(19)          allPath.register(path);
(20)      }
(21)     }
(22)   }
(23)   if (foundPath == false){
(24)    foreach x in receiverDH[]{
(25)      foreach y in senderDH[]{
(26)        sharedDomain = findSharedDom(x,y);
(27)        if (sharedDomain){
(28)           path = new Path(senderDomain,x,
(29)                           sharedDomain,
(30)                           y,receierDomain,
(31)                           'receiverSide');
(32)           allPath.register(path);
(33)        }
(34)      }
(35)    }
(36)   }
(37)   return allPath;
(38) }
```

Figure 5.5: Algorithm for searching a conversion path

in either direction, but conversion must start from the sender domain to the receiver domain through the shared domain.

Until now, we have explained the algorithm of *path()* to find the conversion paths between two given domains. Now, we will explain the algorithm of $bestPath()$ to find out the best path(s) in the set of conversion paths. There are two parameters in comparison process: (i) a type of the indirect relationship, and (ii) length of the conversion path. Based on the types of relationship, we choose the order of preference as follows:

$$one - to - one > many - to - one > one - to - many,$$

where $A > B$ means that A is preferable to B. If the conversion paths have the same type of relationships, the $bestPath()$ function will choose the one that has the shortest length.

Note that, the $bestPath()$ filters out the conversion paths whose type of indirect relationships between two end domains are many-to-many relationship. Therefore, the illegal paths are filtered out in this step.

After the mediator finds the best conversion path, it starts to execute each domain rule along the conversion path. In each domain rule, the mediator inserts the conversion function for converting domain. As the result, conversion functions along the conversion path are inserted to convert data of the domain of the sender into the data of the domain of the receiver.

5.7 Syntax of a Domain Rule and an Example

5.7.1 Syntax of a Domain Rule

Now, we will describe syntax of the domain rule. It has syntax as follows:

Rule: Domain-Name (Parent-Domain)
  Rule-Predicate
External:
  Function-Description
Default:
  Default-Value
Comment:
  Comment-String

A domain rule consists of four parts: rule, external, default, and comment. The **Rule** part is an executable script described as series of *Horn clauses*; its flow control is the same of Prolog. The **Default** part describes the default value of this domain. The **Comment** part gives the explanation about this domain. The **External** part describes the list of conversion functions used in this domain.

Next, we give a systematic treatment of the syntax and basic concepts of semantic of domain rules in details.

Strings of characters: In domain rules, data-unit is represented as strings of characters. Strings of characters are enclosed in single quotes.

Variables: Variables are strings of letters, digits or underscore character (_) started with a character $.

Predicate: A predicate can be regarded as an object that has several components. In another view, it can be seen as a function that has several arguments. There are two types of predicates: a user-defined predicate and a system predicate. For a given domain rule D, each system predicate gives the meaning as follows:

Region('X') This predicate says that D is the domain defined in a region named X.

Cell('X') This predicate says that D is the domain defined in a cell named X.

CDH('yes') This predicate says that D is the domain defined in a common domain hierarchy. If the argument is 'no', it is the domain in a local domain hierarchy.

Parent('RegionX','CellY','domainZ','RelType') This predicate says that the parent domain of D is a domain named *domainZ* that is defined in a region *RegionX* and a cell *CellY*. The type of relationship between D and its parent is *RelType*.

SupportDomain('RegionX','CellY','domainZ','RelType','dirc', 'convFunction') This predicate says that D has a point-to-point link with a domain named *domainZ* that is defined in a region *RegionX* and a cell *CellY*. The type of relationship between D and *domainZ* is *RelType* in the direction *dirc* specified in the fifth argument. The conversion function between D and *domainZ* in the direction *dirc* is *convFunction*. The fifth and sixth arguments can be omitted if bidirection is supported and there is only one conversion function for each direction. In this case, the fourth argument shows the relationship of the forward direction. The mediator determines the opposite relationship for the backward direction. (For example, if the relationship in the forward direction is one-to-many, then the relationship in the backward direction is many-to-one.)

The predicates *Region*, *Cell*, and *CDH* must be defined in all domain rules while the predicates *Parent* and *SupportDomain* are not always necessary.

Other system predicates that are supported in a domain rule are shown as follows:

Confirm('message') This predicate shows users the *message* and requests them an answer 'y/n'. If the users answer 'y', the predicate succeeds. If not, it fails.

$Y = ConvFunction(direction,'convf',$X) This predicate checks the direction(forward or backward) of the conversion. If the direction is equal to the constant in the first argument, $Y = convf($X).

Domain($X,'RegionX','CellY','DomainZ') This predicate checks the domain of input $X. If its region, cell and domain are *RegionX*, *CellY*, and *DomainZ*, respectively, this predicate succeeds. If not, it fails.

Output($X) This command sends the value $X as a result of the domain rule. It is equal to the *return* command in a C program.

$Y = Requires('DomainX') This predicate requests users to input an additional data-value whose domain is *DomainX*. If the domain of the users' data can be converted to *DomainX* by using the semantic conflict resolution, then the command returns the result to the variable $Y. If not, it fails. This command is useful for the domain that is composed of several subdomains.

A user-defined predicate is in form of *PredicateName (listOfVariables)*. It is used to join the system predicates together to create a domain rule.

Execution: The syntax and execution of clauses in a domain rule is similar to those in prolog programs. A clause consists of one head and several goals. Its syntax is as follows:

$$H :- G_1, ..., G_n,$$

where H and $G_i (1 \leq i \leq n)$ have the syntax of predicate shown above. The meaning of the above clause are:

H is true if all $G_i, 1 \leq i \leq n$, are true.

```
( 1) Rule:    Building (Region)
( 2)   Building($X)      :- chkBldCode($X,$Y), Output($Y).
( 3)   chkBldCode($X,$Y) :- Domain($X,'hol.is.uec.ac.jp','carina','Region'),
( 4)                          convertToRegion($X,$Y).
( 5)   chkBldCode($X,$Y) :- Domain($X,'hol.is.uec.ac.jp','carina','Dept'),
( 6)                          convertToDept($X,$Y).
( 7)   convertToRegion($X,$Y) :- $Y = ConvFunction('Forward','bld2Region',$X).
( 8)   convertToRegion($X,$Y) :- $Y = ConvFunction('Backward','region2Bld',$X).
( 9)   convertToDept($X,$Y)   :- $Y = ConvFunction('Forward','bld2Dept',$X).
(10)   convertToDept($X,$Y)   :- $Y = ConvFunction('Backward','dept2Bld',$X).
(11)   Region('hol.is.uec.ac.jp').
(12)   Cell('carina').
(13)   CDH('yes').
(14)   Parent('hol.is.uec.ac.jp','carina','Region','many-to-one').
(15)   SupportDomain('hol.is.uec.ac.jp','carina','Dept','many-to-one').
(16) External:
(17)   bld2Region(input),bld2Region.pl,'W5','West'
(18)   region2Bld(input),region2Bld.pl,'West','{W1,W2,W3,W4}'
(19)   bld2Dept(input),bld2Dept.pl,'W5','S'
(20)   dept2Bld(input),region2Bld.pl,'S','{W5,W9}'
(21) Default:  'W5'
(22) Comment:
(23)    'A data format of this domain is
(24)     String followed with Number(W5).'
```

Figure 5.6: An example of a domain rule

5.7.2 An Example

Let us describe meaning of a domain rule and its operations by an example shown in Figure 5.6. The domain name can be defined as shown in line 1. The **Default** ([line 21]) part describes the default value of this domain. The **Comment** ([lines 22-24]) part gives the explanation about this domain. The **External** ([lines 16-20]) part describes the list of conversion functions used in this domain. It consists of four elements: (i) a program name, (ii) a program file, (iii) a sample of input data, and (iv) a sample of output data. For example, line 19 specifies that the function "bld2Dept()" requires one argument. It is written in a file named "bld2Dept.pl". It changes the format of domain from "Building" to "Department". The last two elements show their data instances.

The most important part of the interface rule is **Rule** ([lines 1-15]). This part is an executable script described as series of *Horn clauses*. As shown in the figure, the predicates, except the first one ("Building($X)") , that start with an uppercase letter are system predicates.

Let the current domain be D. There are two modes of execution for D: a forward mode and a backward mode. A conversion function of the forward mode of the execution is a function that converts the format of a value of D to another, while the backward mode is in the reversed direction. We divide the execution into two modes because there are some cases that this two-way conversion cannot be performed.

Consider the domain rule of the *Building* domain again. When the mediator executes the interface rule, it inserts the conversion functions in the forward mode ([lines 7 or 9]). On the other hand, it can insert the conversion functions in the backward mode([line 8 or 10]). For example, in line [9], the mediator inserts the function "bld2Dept()" for convert the domain of input data ($X) from *Building* to *Dept*.

The lines 11-12 show the region name and cell name, respectively. The line 13 specifies that *Building* is the common domain. The lines 14-15 show that this domain has two facts about the *hierarchy* links and the *point-to-point* links. These facts are read by the mediator before starting computation of the conversion paths. These facts are used for checking the reachability of conversion paths.

5.8 Concatenation Relationship

So far, we have described the semantic conflict resolution between two elements whose data-types are regarded as atomic data. In general, data can be represented at different levels of detail, especially when they are represented as character strings. Thus, conflicts of type of concatenation relationship (one-to-many relationships) arise if one element can be written in term of concatenation of other elements.

Now, consider a navigational integration from an element of WIS S to another element of WIS R where the conversion path between these two domains consists of a domain x and domains y_1 ... y_n that have a concatenation relationship, i.e., $x = f(y_1, ..., y_n)$. Our problem is to determine the semantic conflict resolution between domains of both elements. Below, we present the framework that is applicable to our problem.

Suppose that domain of x is X. The value x consists of a set of values y_i(i = 1,...,n) that are concatenated together. Let domain of y_i be Y_i. Semantic conflict resolution from Y_i to X can be computed as follows:

$$X = f(Y_1, ..., Y_n)$$

where f is a conversion function that concatenates data of domain $Y_i(i = 1, ..., n)$ as shown in Figure 5.7. The relationship between X and $f(Y_1, ..., Y_n)$ is one-to-one

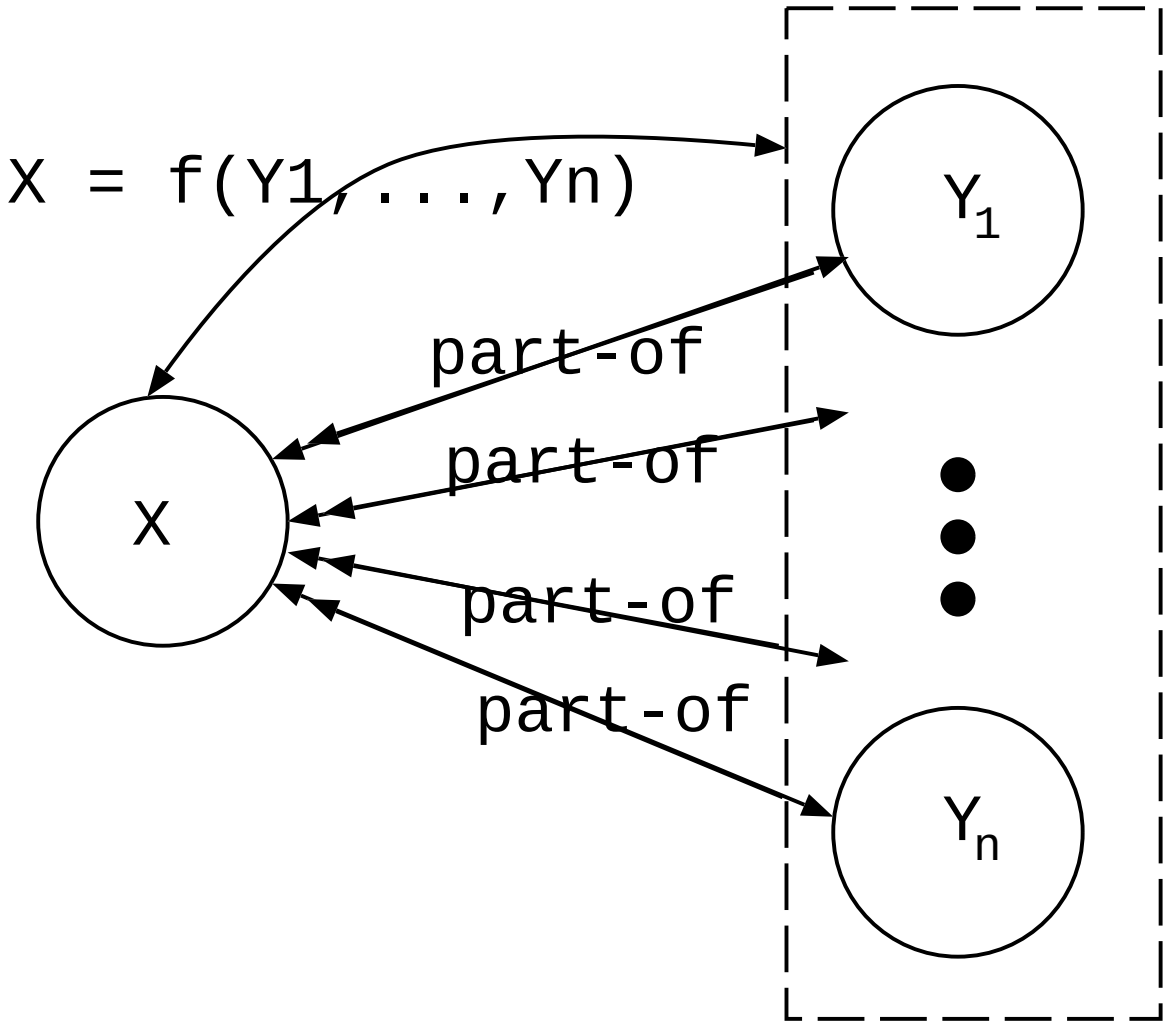


Figure 5.7: An example of domain hierarchy of the concatenation relationship

relationship. Thus, to resolve such a semantic conflict, the conversion $f()$ must be defined in a domain rule of each domain.

Consider the domain hierarchy and the domain rule of domain $LastName$ shown in Figure 5.8 and 5.9, respectively. The domain rule $LastName$ can request input-data from users by the *Confirm* predicate. If the user agrees, the rule will request the user to give it an input-data of domain $FirstName$ for full string matching. It is done by using the *Requires* predicate ([lines 5-8]). The mediator checks whether the user's input can be converted into the $FirstName$ domain. If a conversion path exists, the mediator performs conversion. If any conversion path does not exist, the execution of conversion fails.

If the execution of conversion fails or the user does not agree with the Confirm predicate, the substring matching mode will be performed ([lines 9-10]). However, the substring matching is one-to-many relationship. A table lookup function $lookupNameByLastName()$ is called. The processing cost is more expensive.

From this example, we can see that the rule developer can control the integration characteristic using combination of predicates.

For example, when a user tries to match the data of $JLastName$ domain to the data of the $JFullName$ domain, the semantic conflict can be solved through the domain hierarchy including $FullName$, $FirstName$ and $LastName$ domain. The mediator finds out that there are two conversion paths between $JLastName$ and $JFullName$ ([A] and [B]). According to the $LastName$ domain rule, the path using concatenation function (one-to-one [A]) has higher priority than the path using substring matching (one-to-many [B]). Thus, the mediator requests the user to provide an additional data-unit of domain $FirstName$. If the user input data of the domain $JFisrtName$, the

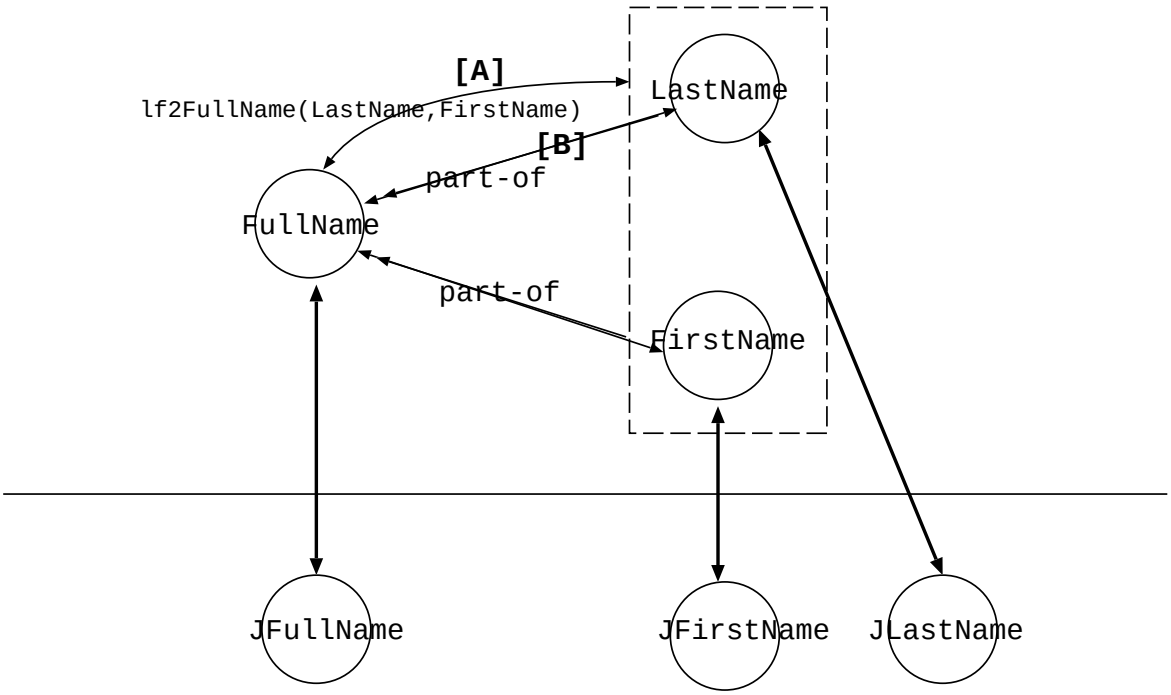


Figure 5.8: An example of domain hierarchy of the lastName domain

```
[ 1] Rule LastName:
[ 2]    LastName($X)     :- chkInput($X,$Y),Output($Y).
[ 3]    chkInput($X,$Y)  :- Domain($X,'is.uec.ac.jp','hol',FullName),
[ 4]                          matchName($X,$Y).
[ 5]    matchName($X,$Y) :- Domain(is.uec.ac.jp','hol','FullName','Forward').
[ 6]                        Confirm('Do you want to use full name matching?[y/n]'),
[ 7]                        $F = Requires('FirstName'),
[ 8]                        $Y = ConvFunction('Forward','lf2FullName',$X,$F).
[ 9]    matchName($X,$Y) :- Confirm('Do you want to use substring matching?[y/n]'),
[10]                        $Y = ConvFunction('Forward','lookupNameByLastName',$X).
[11]    matchName($X,$Y) :- $Y = ConvFunction('Backward','lastName',$X).
[12]    Region('is.uec.ac.jp').
[13]    Cell('hol').
[14]    CDH('yes').
[15]    SupportDomain('is.uec.ac.jp','hol','FullName','one-to-many','Forward','lookupNameByLastName').
[16]    SupportDomain('is.uec.ac.jp','hol','FullName','one-to-one','Forward','lf2FullName').
[17]    SupportDomain('is.uec.ac.jp','hol','FullName','one-to-many','Backward','lastName').
[18] Exteranl:
[19]    lastName(input),lastName.pl,'Hoshi Mamoru','Hoshi'.
[20]    lf2FullName(input,input),lf2FullName.pl,'Hoshi','Mamoru','Hoshi Mamoru'.
[21]    lookupNameByLastName(input),looupNameByLastName.pl,'Hoshi',{'Hoshi Mamoru','Hoshi Noriko').
[22] Default:
[23] Comment:'Hoshi'

         'This is the last name data. It starts with uppercase
          and follows with lowercase.'
```

Figure 5.9: An example of interface rules of the lastName domain

conversion path using the concatenating function $lf2FullName$ is used. The mediator will return the result as follows:

$$name2Jname(lf2FullName(jlast2Last(S \rightarrow \{lastname\}), jfirst2First(S \rightarrow \{firstname\}))$$

where $jlast2Last()$, $jfirst2First()$, and $name2Jname()$ perform data conversion between domains $JLastName \rightarrow LastName$, $JFirstName \rightarrow FirstName$, and $FullName \rightarrow JFullName$, respectively.

5.9 A Domain of Complex String

So far, we have described the semantic conflict resolution between data of two atomic domains. In general, Web documents may contain long strings, and so we can view them as complex string. The semantic conflict resolution for complex string is similar to that for concatenated domains. We must define the relationships (links) between the concatenated domain and its relevant domains. However, complex string may contain a lot of data, and so extracting data from complex string requires expensive processing cost.

For example, assume that a Web document contains paragraphs describing research topics of laboratories in a university. Consider a situation that a user tries to relate given keywords of research topics in another Web document to these paragraphs. Processing of extracting keywords of research topics is to extract each keyword in the paragraphs. Wrapping and describing the complex string is more difficult than the concatenated string.

For the reason described above, we propose a technique to deal with how to define a domain rule for complex string. To formalize our idea, for a given domain of a complex

string D, designing a domain rule has some principles as follows:

- Set a domain name for D as an atomic domain in a local domain hierarchy.
- As a domain rule of D, define relationships between D and other domains. For each domain L, the conversion functions between D and L are defined as follows:
  - For the forward direction (from D to L), the function returns elements (in L) that exist in a given complex string of D. The processing cost of such a function is high because it must loop to check existence of each element of L in the complex string of D. For example, the function that returns research-topic keywords that exist in a paragraph is the function in this case.
  - For the conversion function in the backward direction (from L to D), the string matching is performed. In this case, checking existence of an incoming string/word from the sender in the complex string can be performed in cheaper processing cost because no loop processing is performed.

We can see that generating derived links from atomic data (i.e., keywords) to complex string (i.e., paragrapgh) can be performed easily.

5.10 Execution Plan due to Semantic Conflict Resolution

In this section, we describe the execution plan due to semantic conflict resolution. Let us consider the semantic conflict resolution shown in Figure 5.10. In general, derived links between the result pages of two WISs can be created in two modes:

- In the first mode, the data elements of the first WIS are passed as arguments of the static method of the second WIS to access the result page (as shown in Figure 3.4-a).

- In the second mode, the elements of the first WIS are compared with the elements of the second WIS (as shown in Figure 3.4-b). To execute it, first, the result page of the second WIS must be generated by invoking the static method of the second WIS. The static method may be: (i) explicitly specified in the query command with the data elements of the first WIS or constant values as its arguments, or (ii) implicitly invoked by the wrapper of the second WIS. Thereafter, comparison between the data elements of the first WIS and those of the second WIS is processed.

With the characteristics of the access modes, the mediator must determine the execution plan due to the semantic conflict resolution as follows:

- For the first mode, the native method must be used to access the result page from the second WIS. Thus, data of domain of the sender X must be changed into data of domain of the receiver Y through the conversion path, and this value is passed as an argument to retrieve data from the second WIS (WIS2).

- For the second mode, the indirect relationship between X and Y might be one-to-many (see Figure 5.10). In this case, the mediator finds a shared domain C in the conversion path where the relationship from X to C and the relationship from Y to C are one-to-one and many-to-one, respectively. This is the condition that the temporary data generated by the conversion process are minimum. Then, data of domain of the sender X must be changed into data of C and data of domain of

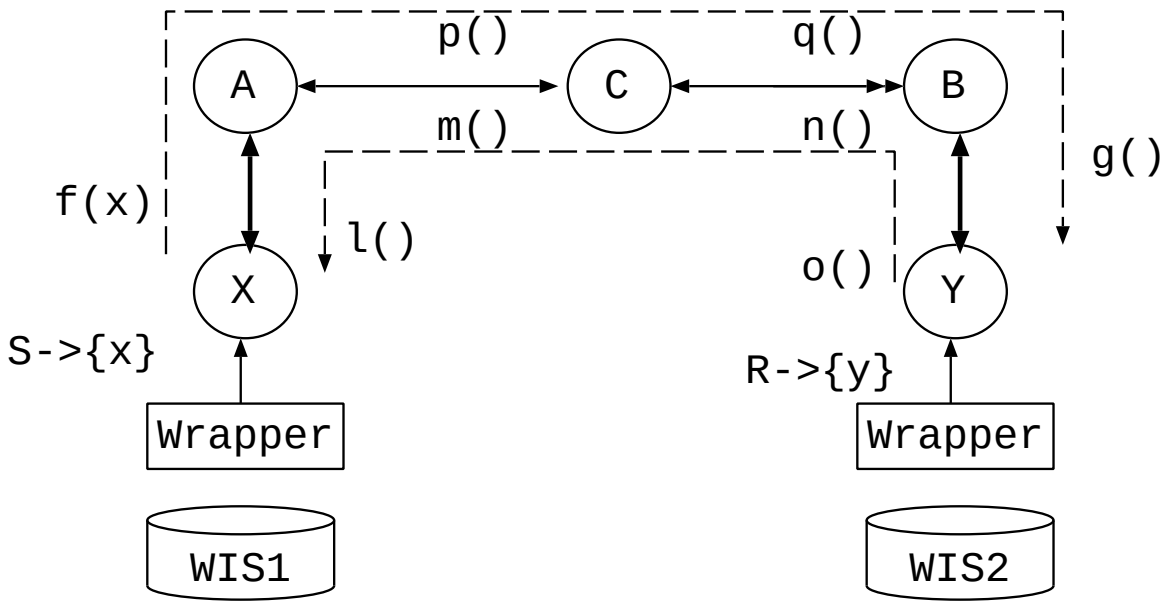


Figure 5.10: An execution plan due to semantic conflict resolution

the receiver Y must also be changed into data of C. Thereafter, the comparison between data converted from two domains must be performed.

In this way, the mediator must specify appropriate execution plans according to an indirect relationship between two end domains of navigatinal integratin. In particular, if the receiver WIS is an ordinary Web server providing a simple document and navigational integration is in the second mode, this function of execution plan generation is effective.

Example: To understand how to generate the execution plan more clearly, let us consider two WISs shown in Figure 5.10 (WIS1 acts as a sender and WIS2 acts as a receiver). Let $f(), p(), q()$ and $g()$ be a sequence of functions for converting data from the domain X to the domain Y and $o(), n(), m()$, and $l()$ be as a sequence of functions for converting data from Y to X.

- If the navigational integration is created by the first mode, the execution plan described in the where condition of a query command is generated as follows:

$$R \rightarrow \{y\} = g(q(p(f(S \rightarrow \{x\})))).$$

That is, data of the sender must be converted into data of the receiver. Each item of data is sent to the receiver WIS when the user make a request.

- If the navigational integration is created by the second mode and if the indirect relationship between X and Y is one-to-many, the generated condition is:

$$n(o(R \rightarrow \{y\})) = p(f(S \rightarrow \{x\})).$$

In this case, the data from both sides are converted to the domain C and compared together at this point. From the point of domain C, the indirect relationship from X to C and Y to C are one-to-one and many-to-one, respectively.

5.11 Extensibility for Handling Multi-regions

We have described the benefit of separating between the CDH and the LDH in the domain hierarchy in the the view point of system evolution and scalability within the same region. In this section, we will discuss how to handle a semantic conflict problem that occurs between different WISs in different regions.

To handle the semantic conversion among multiple regions, we do not decide to create another CDH between the CDHs of two regions. We think that it is an endless way to extend the scale of system. Instead, we have three solutions as follows:

1. Set global domains that are standard for all regions. For example, the ISBN number is one of the domain of this case.

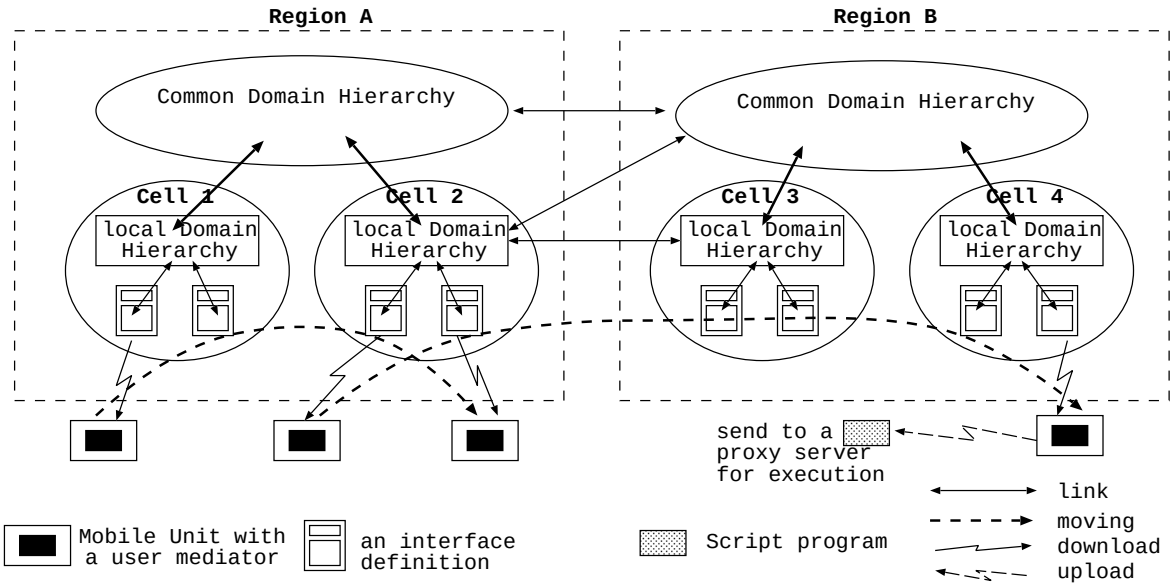


Figure 5.11: A general environment in our approach

2. Add point-to-point links between domains of CDHs of different regions by a region manager.

3. Add point-to-point links between domains of different regions by a cell manager.

The second way provides the semantic conversion of data between two regions while the third way provides that function in the smaller scale. The scale of the third way covers one cell and the other region/cell. The general environment of our approach is shown in Figure 5.11.

5.12 Summary

In this chapter, we have presented a new framework for mobile users to automatically resolve semantic conflcit among multiple independently-developed organizations. The common domain hierarchy (CDH) is proposed to express the standard exchanging format among autonomous cells and the local domain hierarchy (LDH) to express the

different kinds of data formats in each cell. By defining the hierarchy relationships and the conversion functions between the CDH and the LDH, the information conversion environment between autonomous cells that share the same CDH can be established. As shown in previous chapters, the interface definition is used to wrap and describe properties of a WCA uniformly. By attaching the domains used in WCAs to corresponding domains in the domain hierarchy, the information between WCAs can be exchanged together. The domain hierarchy information and interface definitions are packed into a package, downloaded into a user's mobile unit while he is moving in the networks. With a mediator on his mobile unit, the user can explore information in these packages and generate a customized WCA to integrate information from multiple WISs by themselves. The semantic conflict between different two data of different WISs can be solved by finding a conversion path between them.

To find out the ability of resolution, the reachability between two domains in the domain hierarchy is investigated. Moreover, to prevent ambiguous meaning of the conversion, the legality of the conversion path is proposed. An example of a domain rule is also described.

In general, information can be represented not only as atomic data but also in different levels of details, especially when it is represented as charecter strings (i.e., concatenated data-elements or paragraph). In this chapter, we have also discussed the semantic conflict resolution of such information. The execution plans due to semantic conflict resolution have been also described.

Chapter 6

System Prototype

In this chapter, we will describe the system prototype of our approach. First, we start with the prototype configuration that we implemented. Next, we show the region design that we described in chapter 5. Thereafter, an applicability of system is described. It includes the range of WISs to which our approach can be applied and the efficiency and effectiveness of the proxy server. Finally, efficiency and effectiveness of the mediator is discussed.

6.1 Prototype Configuration

The prototype system that we implemented consists of three cells to form a region. Each cell is assumed as a laboratory. Each cell has a workstation running as a MIR and manages several existing Web information sources. An overview of the system architecture of our prototype is shown in Figure 6.1. Each MIR is implemented on a SUN workstation Ultra1. One MIR maintains a region repository (CDH). A mobile computer is a notebook computer with a Pentium MMX 233 MHz and running Linux operating system.

Table 6.1 shows the number of lines of program code devoted to implementing each module in the prototype system. Most of them are written in the Perl language and

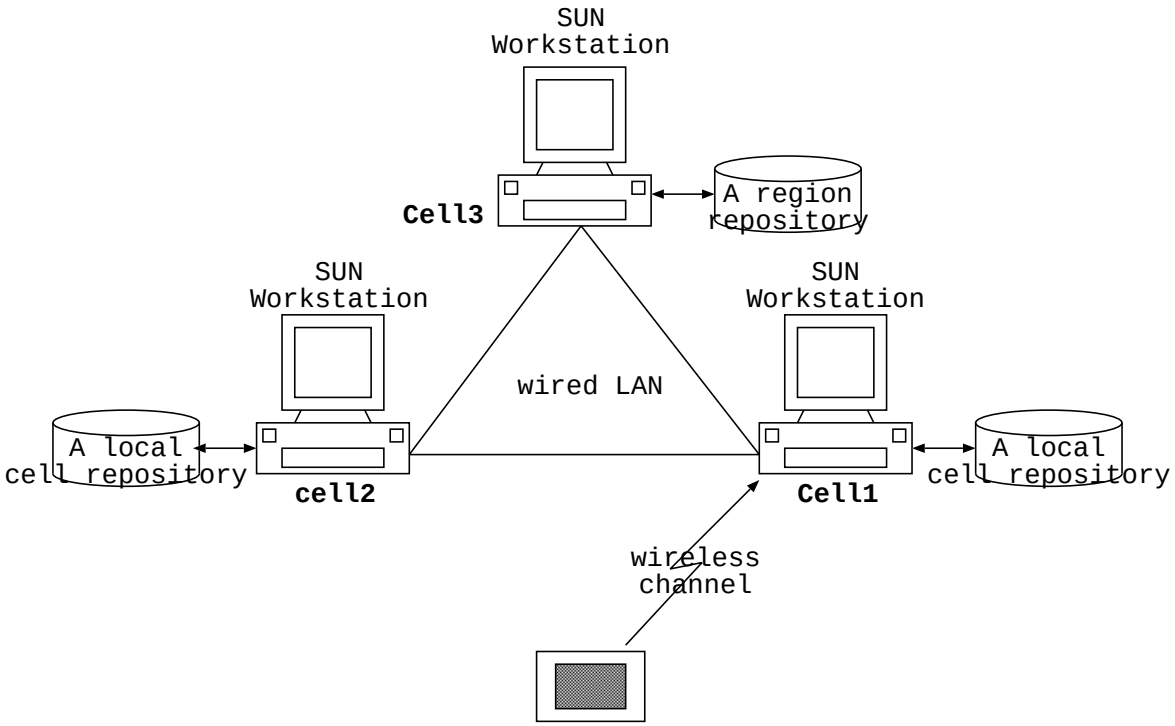


Figure 6.1: A prototype configuration

Table 6.1: Lines of program code for each module in the prototype system

Modules	Sub-module	Lines of Code
(1) Common Library	(i) Perl-objects in the Common Data Model	1716
	(ii) WIS-specific development libraries	1580
(2) Mediator	(i) GUI	983
	(ii) Rule Interpreter	2960
(3) General Wrapper , Proxy Executor and repository manager		2968
Total		10207

some parts concerned with Java-based WISs are written in the Java language.

The common library is divided into two sub-modules. The first one is the library for handling objects of the common data model that are implemented by data structures provided in the Perl language. The second part is a set of development libraries for developing WIS-specific wrappers.

The Mediator module is divided into two modules: a GUI part and a domain rule interpreter. The GUI module is written by a perl/Tk[21] graphic library in Perl. The domain rule interpreter includes the function of conversion-path finding.

The general wrapper defines the standard interface to communicate with WIS-specific wrappers and works with the proxy executor to execute users' request. Thus, we count them as one module.

We implemented the wrappers for the following WISs shown in table 6.2. To test the heterogeneous WCAs, we used several types of the clients shown previously in Fig. 3.6. The first source is a collection of HTML files which is accessible through a http protocol. The second is the clickable map developed by the Java language. The others

Table 6.2: Information sources and their Web-embedded Client Applications

Information Sources	Client	Backend	MIR
Department Homepage	-	ordinary WWW	MIR1
Conference DB	Applet	Postgres	MIR1
Laboratory DB	CGI	Postgres RDB	MIR2
Search Engine	CGI	Altavista search Engine	MIR2

Table 6.3: Lines of program code of interface definitions and WIS-specific wrapper for each WIS

Information Sources	Interface Definition	WIS-specific Wrapper
Department Homepage	112	167
Conference DB	74	182
Laboratory DB	154	185
Search Engine	61	138

are Web-embedded clients using the CGI interface. (The Altavista server[2] is running at http://www.altavista.com/.)

Table 6.3 shows the lines of code of the interface definitions and the WIS-specific wrappers for the WISs specified in Table 6.2. In practice, the lines of code depend on the complexity of WISs. Using the common library shown above, the lines of code of simple WIS-specific wrappers in our prototype do not exceed 200 lines in Perl.

6.2 A Region Design for a university

Figure 6.2 shows a domain hierarchy for a university of our system prototype. It contains two cells. Each cell maintains the domains used by its maintained WISs.

As shown in Figure 6.2, the common domain hierarchy maintains the domains that are always used in the university. In each cell, the local domain hierarchy maintains

only domains in its cell. Hierarchy links between domains in the CDH and domains in the LDH are created. If there is not a domain in the CDH that can be set as a parent domain for domains in the LDH, the cell manager can set its own hierarchical structure in the LDH and waits for these domains (or some domains in different formats) to be promoted as domains in the CDH. The below part of Figure 6.2 shows such domains.

In the traditional integration systems that employ a centralized system architecture, all domains must be maintained in one host server. Moreover, the static semantic resolution is more complicated and hard to be maintained because the number of conversions for each type of domain is proportional to $O(n^2)$ ($= 2n(n-1)$ where n is the number of formats of a domain). For example, the date domain in Figure 6.2 has four formats: Date(American format), EDate1, EDate2 (European format) and JDate (Japanese format). To create conversions between them in case of centralized resolution, they require 2*4*(4-1) = 24 functions for describing format conversion.

In contrast, our approach requires less maintenance cost. The number of domains that are maintained in each cell is the summation of number of domains in the CDH and the number of domains in its LDH. Moreover, the number of conversion functions for each type of domain maintained in each cell is proportional to the number of types in the cell (= 2n where n is the number of types). For example, the number of conversion functions between the American date and two types of the European date in the first cell equals 2*2 = 4.

In other words, our approach allows each organization to maintain only necessary information for providing the semantic conflict resolution among several organizations in the same region. In the point of view of extensibility such as adding a new domain and a new cell, it has been already described in the chapter 5.

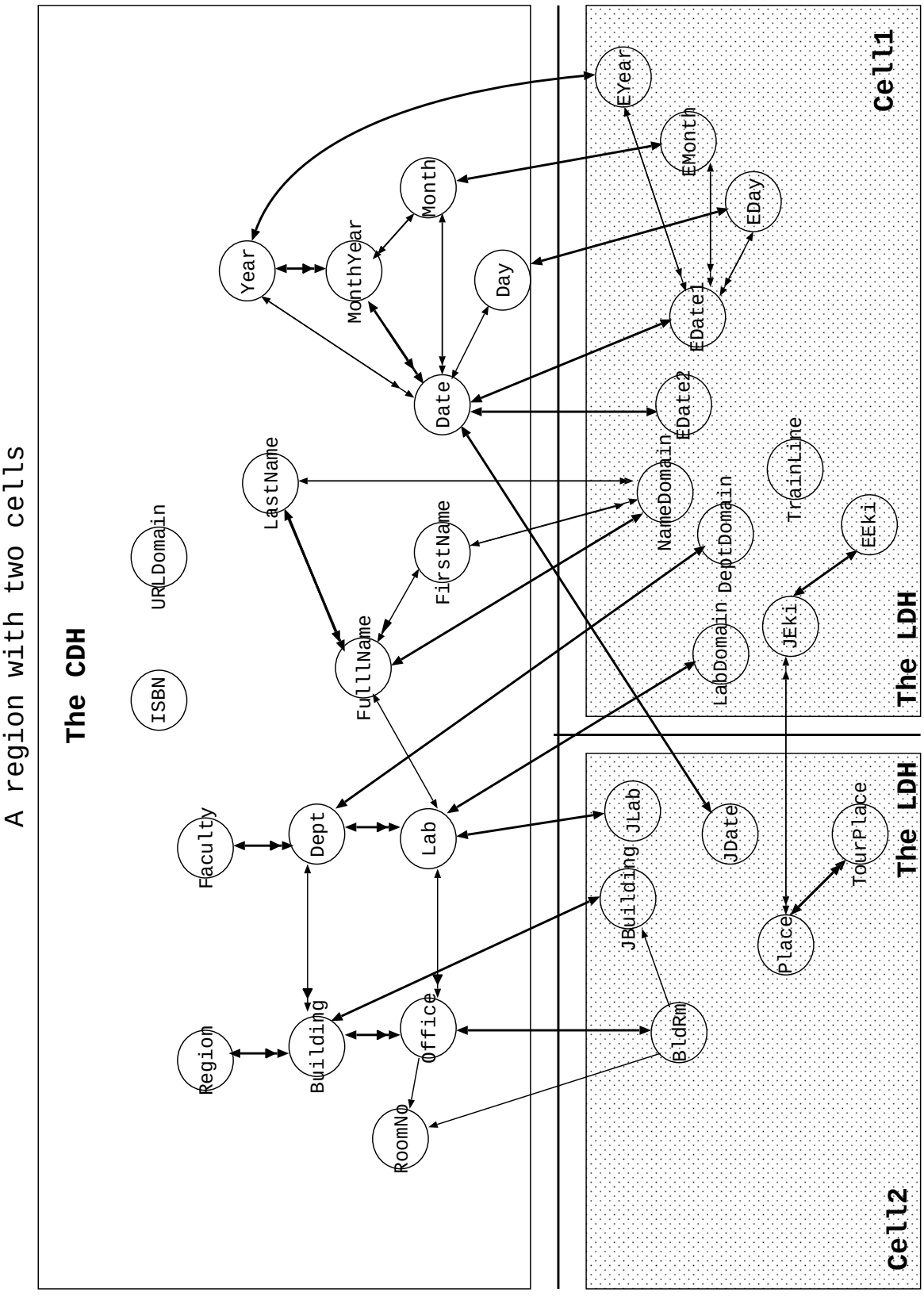


Figure 6.2: A domain hierarchy for a region in a university

6.3 Applicability of System

This section illustrates the applicability of our system. It consists of the following topics: (i) effectiveness of wrapper, (ii) efficiency and effectiveness of proxy executor, and (iii) efficiency and effectiveness of mediator.

6.3.1 Effectiveness of Wrapper

Using the proposed common data model, we can define an interface definition and develop a WIS-specific wrapper for wrapping a WIS. The server-access methods of ordinary HTML and CGI-form based WIS can be wrapped through methods of interface definitions easily. However, data structures of documents that we used in this prototype are regular structures. In the viewpoint of Document Tag Definitions (DTD), the regular structure can be defined by using DTD without OR structure. The extension for supporting *OR* structure will be described in chapter 7.

As described above, wrapping ordinary HTML or CGI-form based WIS is easy. The difficult wrapping task is to wrap a Java-based WIS. In this type of WISs, developers must investigate whether a given Java-based WIS satisfies the constraint described in section 3.3 of chapter 4.

The next difficult task is to provide a method for embedding a query command into the front page of an original WIS. In case of the ordinary HTML and CGI-based WIS, embedding a query command is only an operation of string replacement. In case of a Java-based WIS, first, all class files of the WIS must be downloaded and installed into the MIR. Next, a template of a Java program for embedding a query command must be generated. The program generated by this template must also inherit all classes of the original WISs.

6.3.2 Efficiency and Effectiveness of Proxy Executor

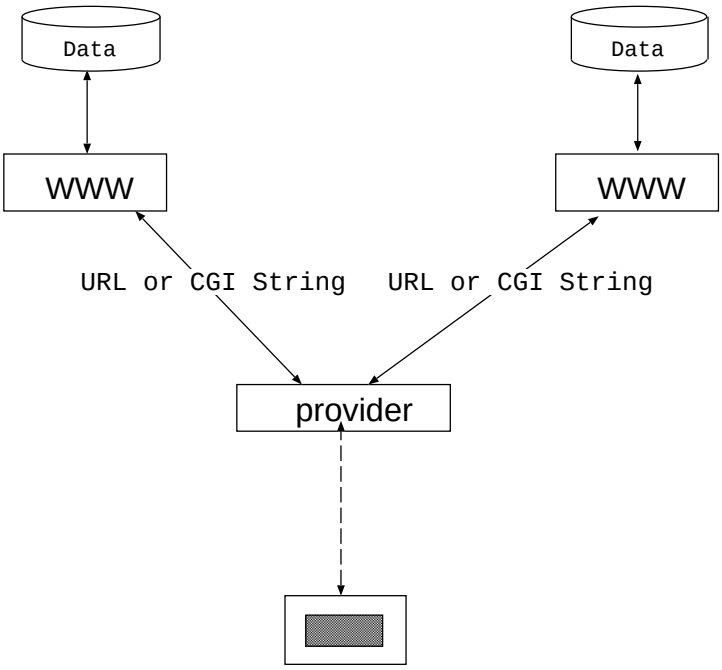
Comparing our navigational integration with the current Web approach shown in Figure 6.3, we can see that our approach has an additional processing cost, which is the process between the proxy executor and the wrapper. In the experiment, we look into the execution time of several steps as follows:

Initialization time: It includes time for: setting up environment, a query analysis, generating execution plan, and invoking WIS-specific wrappers.

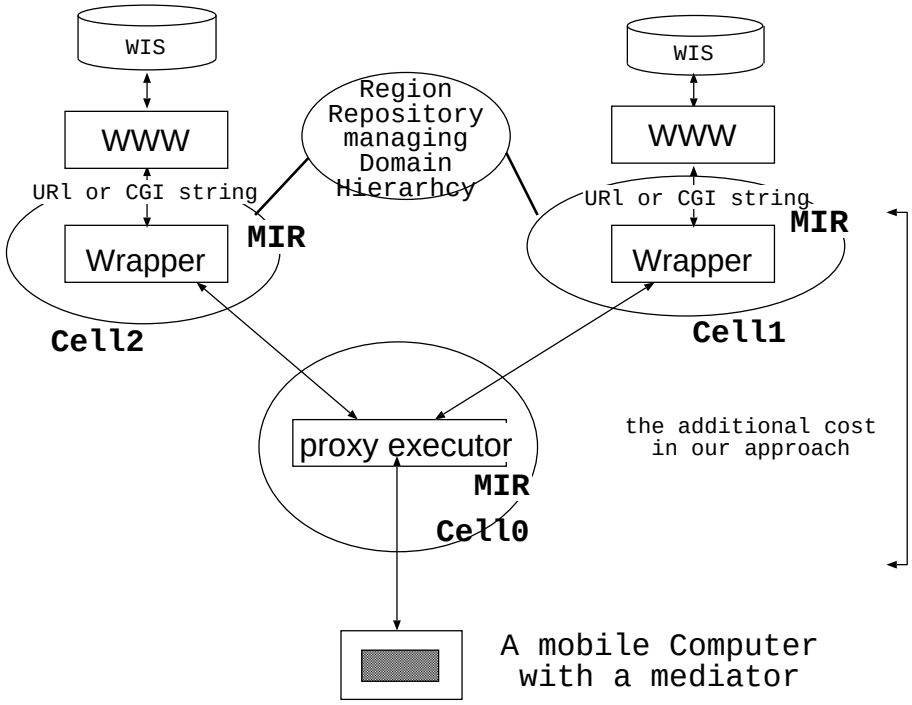
Execution time of WIS: This time is the difference between the time when the WIS-specific wrapper sends a request to a WWW server and the time when the WIS-specific wrapper receives a result page from the wrapper.

Post-processing time: It is the time for running four steps: (i) extract objects from the result page and embed positions of the objects into the result page, (ii) send the result page to the proxy executor, (iii) resolve semantic conflict, and (iv) embed navigational execution tags into the result page.

Below is the execution time of each step of the query shown in section 4.4 of chapter 4. We can see that the initialization time for starting execution is 0.5 second. Some of the time is also taken for loading Perl libraries. The time to access the department Web page from the WWW server by specifying URL is 0.48 second. In this experiment, the last is the post-processing time. It takes 0.14 second. We do not include the semantic conflict resolving. The execution time for resolving semantic conflict resolution is hard to be predicted. It may take very short in case of format conversion and take longer in case of invoking database lookup.



(a) A System Architecture of the WWW



(b) A System Architecture of Our Approach

Figure 6.3: Comparison between the navigational integration and the WWW

Initialization time	0.46 usr 0.04 sys = 0.50 secs)
Time of WIS (Native Command)	0.42 usr 0.06 sys = 0.48 secs)
Time of Post-processing	0.12 usr 0.02 sys = 0.14 secs)

Clearly, there is a large variance among the response times of processing in the step of post-processing depending on the complexity of semantic conflict resolution. However, the overall time is acceptable. Thus, it can be said that users receive benefits of automatic semantic conflict resolution and derived links embedding without becoming impatient on the self-conversion and keyboard typing.

6.3.3 Efficiency and Effectiveness of Mediator

In the experiment of this prototype, we find that the mediator can search the conversion paths between two domains in an acceptable response time. A user does not wait too long for finding appropriate conversion paths. From the result of experiment, we got the result that most of the execution time is consumed by interacting with a user. For example, we tested the path computation module for finding conversion paths between two domains. The solution of conversion path in this test has two possible paths with minimum of length 3. As a result, we found that overall execution takes 7 seconds while the computation taken by the CPU is 0.85 seconds.

In practice, we find that the problem of the mediator is not the efficiency of the conversion path computation. The problem is how mobile users can recognize what WISs can be integrated together. One solution is the function that finds out pairs of convertible domains in two WISs. Based on our reachability computation described in the previous chapter, it can be straightforwardly implemented.

Chapter 7

Enhancements

7.1 Enhancements for Supporting Semi-structured Data

Recently, handling semi-structured [25, 24] data becomes an important topic in the WWW framework. In general, the characteristics of semi-structured data are different from those strictly typed in conventional database (e.g., the relational database or the object-oriented database). To support the semi-structured data, enhancements must be included to support the following particular properties:

A heterogeneous domain: Some elements in a document may use several different forms to present information. Their meaning and format may be irregular. For example, a "place" may be represented in some parts as a building name and in others as a pair of a building name and a room number.

A heterogeneous structure: The same information may be structured in different forms. For example, "staff" information may be structured somewhere as Teacher objects and elsewhere as Visitor objects.

A concatenation domain: Some elements may be represented in terms of concatenation of several sub-elements. For example, "name" information is represented

by the concatenation of first name and last name.

In this section, we describe enhancements for supporting these semi-structured data in terms of a common data model, path expressions, and semantic conflict resolution algorithms.

7.1.1 The Common Data Model

In this subsection, first, we will describe how to enhance the common data model described in chapter 4 for supporting semi-structured data. Next, we describe how to map schema between DTD[29] and our common data model. Finally, we show an example.

(1) Enhanced Domain for Supporting Semi-Structured Data

To describe how to model semi-structured 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 chapter 4. In addition, Let the '|' symbol indicate an OR condition 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. **refto** IntfName

5. (**refto** IntfName |**refto** IntfName | ...)

6. **setof** IntfName

7. **setof** (IntfName | IntfName | ...)

8. (AtomicDomain | AtomicDomain | ... | **refto** (IntfName | IntfName | ...))

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

The syntactic grammar of *DomainName* is shown in Table 7.1.

Table 7.1: A syntactic grammar of *DomainName*

$< DomainName >$	::=	$< AtomicDomain > \mid < AtomicAndDomain > \mid$ $< AtomicOrDomain > \mid < setOfIntfs > \mid$ $< reftoIntfs > \mid < DomainStruc >$
$< AtomicDomain >$	::=	$< letter > \{[< letter > \mid < digit >]*\}$
$< AtomicAndDomain >$	::=	$'('< AtomDomAndList >')'$
$< AtomDomAndList >$	::=	$< AtomicDomain > \{',' < AtomicDomain >\}+$
$< AtomicOrDomain >$	::=	$'('< AtomDomOrList >')'$
$< AtomDomOrList >$	::=	$< AtomicDomain > \{'\mid' < AtomicDomain >\}+$
$< IntfName >$	::=	$< letter > \{[< letter > \mid < digit >]*\}$
$< IntfNames >$	::=	$< IntfName > \{'\mid' < IntfName >\}+$
$< setOfIntfs >$	::=	**setof** $'('IntfNames')'$
$< setOfIntfs >$	::=	**setof** $IntfName$
$< refToIntfs >$	::=	**refto** $IntfName$ [$'\mid'$ **refto** $IntfName$]*
$< DomainStruc >$	::=	$'('< AtomicDomain >' \mid' < setOfIntfs >')'$
$< DomainStruc >$	::=	$'('< AtomDomOrList >' \mid' < setOfIntfs >')'$
$< digit >$	::=	0,1,2,3,4,5,6,7,8,9
$< letter >$	::=	a,b,...,z,A,B,...Z

The variations 1, 4 and 6 are the simple domains for supporting structured documents described in chapter 4. The variation 2 supports the heterogeneous domain property. The variations 5 and 7 support the heterogeneous structure properties. The variation 8 and 9 support combination of the heterogeneous domain and the heterogeneous structure properties. The variation 3 supports the concatenation domain properties.

Another thing that must be considered is domains of arguments of methods defined in interface definitions. Unlike domains of elements defined in a DTD that describes a structure of a document, the domains of the arguments of the methods are not so complex. In practice, the variations of the domains of the arguments of the methods can be expressed as follows:

1. AtomicDomain
2. (AtomicDomain | AtomicDomain |...)
3. **setof** AtomicDomain

The first variation is a common way for defining a domain of an argument. The second variation can be found in a CGI-form that can receive either full-string or sub-string of data as an input data. The sub-string may be data of another domain. Thus, the input data may be a value of the heterogeneous domain. The third variation can be seen in search engines that can receive a list of data as an input data.

(2) Schema Matching between DTD and The Common Data Model

Schema matching[32] is the process of matching elements of a given DTD with elements of our common data model. After the schema matching is set, the instances of the DTD can be translated into the instances of our common data model.

It is clear this data model can support these combinations by separating them into several attributes based on the variations described above. As an example, consider the following DTD.

```
<!ELEMENT   C       (D,(E*|F))>
<!ELEMENT   E       (G,H)>
<!ELEMENT   D       (#PCDATA)>
<!ELEMENT   F       (#PCDATA)>
<!ELEMENT   G       (#PCDATA)>
<!ELEMENT   H       (#PCDATA)>
```

From the above DTD, it cannot define a domain to support $D,(E*|F)$, but it supports $(E*|F)$. Therefore, this DTD must be changed into another equivalent DTD as follows:

```
<!ELEMENT   C       (D,X)>
<!ELEMENT   X       (E*|F)>
<!ELEMENT   E       (G,H)>
....
```

In the equivalent DTD, a new derived element X is defined to handle the $(E*|F)$. The domain of X can be defined as $(setof\ E|AtomicDomainF)$ where $AtomicDomainF$ is the atomic domain of the element F. Also, E is defined as another interface definition. In this way, even though some additional elements must be added, we can transform a DTD of any document into our common data model without losing any data.

(3) Example

As an explicit example, consider the interface definitions in Figure 7.1, which describes a document about a university. Suppose that the staffs in this university are divided into two types: teachers in this university and visitors invited from the outside. They are represented in different structures. Each staff has a *contact* attribute whose

```
database ISInformation
address  http://HOST1/cgi-bin/execute.pl
interface  Laboratory
body
   labName    LabDomain,
   staffs     setof (Teacher|Visitor),
method
public:
  ...

endInterface
```

(a) New "Laboratory" Interface Definition

```
database ISInformation
address  http://HOST1/cgi-bin/execute.pl
interface  Teacher
body
   position    PostDomain,
   fullName    (FirstName,LastName)
   contact     (EmailDomain|RoomNoDomain)
method
  ...
  ...
endInterface
```

(b) The "Teacher" Interface Definition

```
database ISInformation
address  http://HOST1/cgi-bin/execute.pl
interface  Visitor
body
   position    PostDomain,
   fullName   (FirstName,LastName)
   contact    (EMailDomain|RoomNoDomain)
   company     CompDomain optional,
method
...
endInterface
```

(c) The "Visitor" Interface Definition

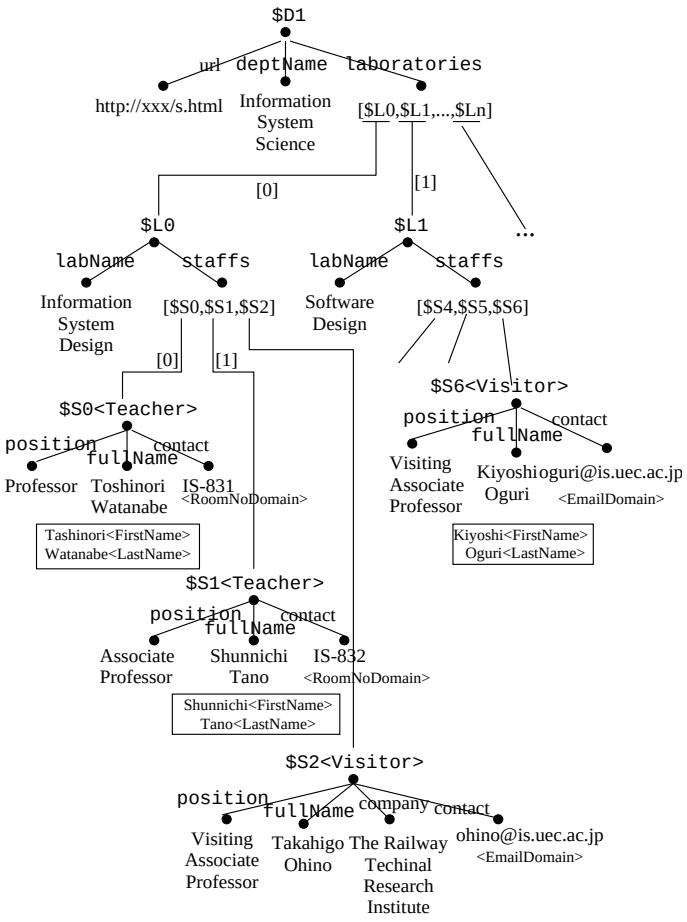


Figure 7.1: New interface definition and an instance example

value may be an e-mail address or a room number. In addition, the domain of the staff name can be divided into two domains: the *FirstName* and the *LastName* domain.

As shown above, our common data model provides an *optional* attribute, which allows to describe information that can be omitted (e.g., attribute *company* in the interface *Visitor*). The union type allows to describe the *heterogeneous structure* that can represent the same information in different structures (e.g., attribute *staffs* in the interface *Laboratory*). The union domain allows to describe the *heterogeneous domain* that can represent the same information in different domains (e.g., attribute *contact* in the interface *visitor* and the interface *teacher*). The concatenation of domains allows to describe the *concatenation domain* that represents an element consisting a sequence of other elements (e.g., attribute *fullName*).

7.1.2 Extended Path Expression

As described earlier, the use of path expressions starts with a name of an interface definition and is followed by concatenation of attribute names (e.g., $Department \rightarrow \{laboratories\}[*] \rightarrow \{staffs\}[1] \rightarrow \{fullName\}$). The notation [] is used for specifying the element number in a multi-valued attribute.

Hereafter, we will describe the general form of path expression that is enhanced from the definition described in chapter 4.

Definition 1. Let $t_0, ..., t_n$ be (not necessarily distinct) interfaces. Suppose that the interface t_{n-1} is described by

interface t_{n-1} **is**

$$[..., A_n \ \ (dom_1|dom_2|...|dom_k), ...],$$

or

interface t_{n-1} **is**

$$[..., A_n \ \ (dom_1, dom_2, ..., dom_k), ...].$$

That is, t_{n-1} is the interface with an attribute A_n and the domain of A_n is a heterogeneous type or a concatenation type. Let $domainVar \in \{dom_1, dom_2, ..., dom_k\}$. Then, a **path expression** on t_0 is an expression $t_0 \rightarrow A_1 \rightarrow ... \rightarrow A_i \rightarrow ... \rightarrow A_n < domainVar >$ where $1 \leq i \leq n-1$, and $< domainVar >$ is an optional term. The result of the path expression is the set of objects (or values) of the specified domain of the attribute A_n that can be reached via the specified attribute chain. □

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

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

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:

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

The path expression in our approach provides a uniform mechanism for not only expressing navigation through the nested structure but also specifying domain of information. This ability of domain expression in the path expression is necessary because our framework concentrates on resolving semantic conflict between different information. This ability is not supported by any previous systems or query languages because most of them concentrate on querying and restructuring semi-structured data. That is, our system can find out how to convert the semantics of each data-value to the semantics of target data even though they can be expressed in several representations (domains).

Operation on Extended Path Expressions

We provide some operators. Let pe_i (i = 1, 2, ...) be an extended path expression described above. The operators provided for operating on the extended path expressions can be listed as follows:

1. $pe_i.pe_j$ – **concatenation** operator. It returns a value that is a concatenation of two data-values specified by two path expressions. For example, $Department \rightarrow \{laboratoies\}[*] \rightarrow \{staffs\}[1] \rightarrow \{fullName\} < FirstName > .Department \rightarrow \{laboratoies\}[*] \rightarrow \{staffs\}[1] \rightarrow \{fullName\} < LastName >$ returns a value that is a concatenation of a firstname value and a lastname value.

2. $[pe_i]$ – **set** operator. It returns a set of values specified by one higher level in a path expression. For example, $[Department \rightarrow \{laboratories\}[*] \rightarrow \{staffs\}[*] \rightarrow \{fullName\}]$ returns a set of staff names for each laboratory.

3. $[pe_i, pe_j, ...]$ – **list** operator. It returns a list of values specified by a path expression. For example, $[Department \rightarrow \{laboratories\}[*] \rightarrow \{staffs\}[1] \rightarrow \{fullName\}, Department \rightarrow \{laboratoies\}[*] \rightarrow \{staffs\}[1] \rightarrow \{position\}]$ returns a list of name of the first staff followed by his/her position of all laboratories.

The second operator and the third operator return the same format of a sequence of values as a result. However, their inputs are different. The second operator requires a path expression, where the domain of the attribute before the last attribute is in the form of **setof**. In contrast, the third operator does not require that.

Using these operators, users can create a new data-value in terms of several path expressions.

From now on, we will use the term *path expressions* as the meaning of *extended path expression.*

7.1.3 A Navigational Integration in Terms of Path Expressions

The previous section has described how to use path expressions to specify data-values in semi-structured documents, which contain heterogeneity characteristics. The next step is to specify navigational integration in terms of path expressions in query conditions.

(1) DNF Expression of Navigational Integration in a WHERE Clause

To describe navigational integration, we let the mediator use a *disjunctive normal form* (DNF) in the where clause of a query command. That is, a condition clause consists of a set of sub-conditions that are ORed together, where each *sub-condition* in turn consists of a set of atomic expressions or a method invocation connected only by ANDs. For the moment (until section 7.1.5), we use a DNF expression under some restrictions.

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

$$((S \rightarrow methodOfS(pe_{R1}) \textbf{ and } (pe_{S11} = pe_{R11}) \textbf{ and } ...)$$

$$\textbf{or}$$

$$(S \rightarrow methodOfS(pe_{R2}) \textbf{ and } (pe_{S21} = pe_{R21}) \textbf{ and } ...)),$$

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.

Consider the above condition in which sub-conditions are glued together by OR for expressing navigational integration from R to S. Each sub-condition represents one

derived link in the navigational integration. Note that, a DNF condition described above has the following constraints:

1. A derived link is generated when all statements in a sub-condition are valid. That is, all simple comparisons and a method invocation in a sub-condition of the DNF condition must be satisfied.

2. In a DNF condition, there is only one sub-condition that is valid.

In this example, there are two sub-conditions. both sub-conditions are the patterns of the second mode of navigational integration that use the method to retrieve a result Web page and the remaining comparison operation is to create derived links between elements in two result pages of the WISs R and S. Either of these subconditions is valid but both of them cannot be valid at the same time.

In the following, we call such a DNF condition consisting of the above restrictions *a restricted DNF condition.*

(2) Position of Derived Links

One important thing in the navigational integration is to fix a starting position of derived links. In case of navigational integration between two atomic domains, the starting position of derived links are the positions of elements corresponding to sender's domain. However, the operators provided earlier (concatenation, set, list) work on multiple data-values, and so positioning derived links is not trivial.

From now on, let us use the term *position* as the meaning of *starting position.* We have the following rules for deciding the position of derived links:

A concatenation operator: In this case, the position of derived links is the position of the first element. For example, for given attributes A, B, C, the position of derived links of $(A.B.C)$ is the position of attribute A.

A set operator: In this case, the position of derived links is the position of one upper level attribute. For example, the position of derived links of $[Int1 \rightarrow ...\{A\}[*] \rightarrow \{B\}]$ is the position of attribute A.

A list operator: In this case, the position of derived links is the position of the first element. For example, for given attributes A, B, C, the position of derived links of $[A, B, C]$ is the position of attribute A.

Figure 7.2 shows the positions of links for each operator.

In case where we want to generate derived links at the positions specified by an attribute which are different from the default positions, we can place the required attribute at the front of a sub-condition clause. For example, using this modification, the condition clause will be changed into the following form:

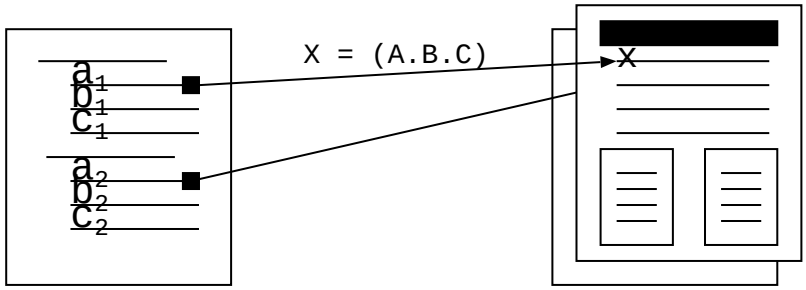
$$(pe_{R3} \textbf{ and } S- > methodOfS(pe_{R1}) \textbf{ and } (pe_{S1} = [pe_{R2}, pe_{R3}]))$$

Without the first term (pe_{R3}), derived links are generated at the positions of pe_{R2}. However, by adding pe_{R3}, users can force the system to generate the derived links at the positions of pe_{R3}

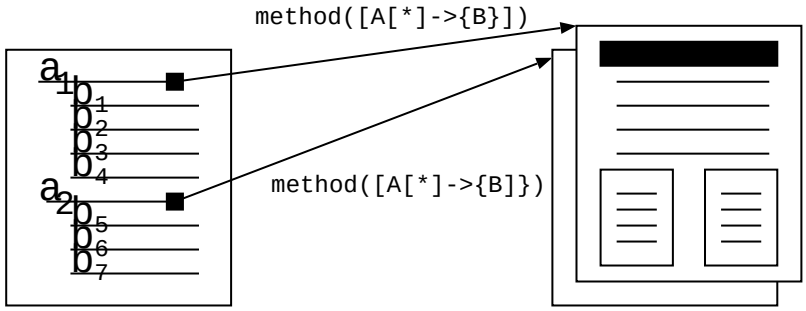
7.1.4 Extended Semantic Conflict Resolution

In the previous section, we have described that queries for navigational integration on semistructured data are given in restricted DNF expressions. It means that a mobile user must specify such a query condition according to interactions with the mediator on his mobile computer.

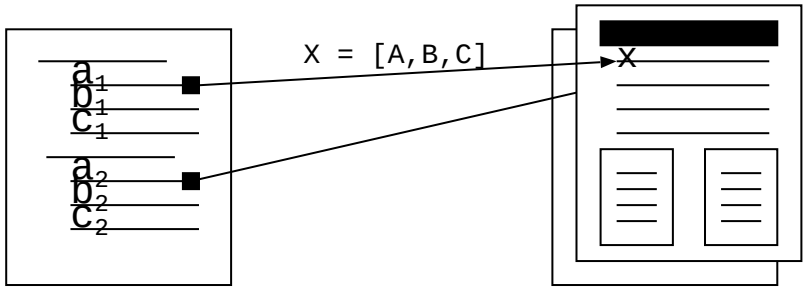
Remember here that the algorithm of semantic conflict resolution in Chapter 5 deals with only atomic domains. Therefore, as far as a query condition of navigational



(a) positions of links in case of using a concatenation operator



(b) positions of links in case of using a set operator



(b) positions of links in case of using a list operator

Figure 7.2: Positions of derived links for each operator

integration on semi-structured data is made of only *the expressions on atomic domains*, the mediator can resolve semantic heterogeneity automatically for building such a query (of a restricted DNF expression).

However, such a query condition (which consists of expressions on atomic domains) will be sometimes too complex in case of navigational integration on semi-structured data. In spite of such complexity, a user must explicitly specify such a query for any given semi-structured data, and it might be a difficult work. Thus, we require an ability that the mediator can generate a query condition of navigational integration on semi-structured data by way of a short-cut form. To achieve this requirement, the mediator should use (in default) an enhanced algorithm of semantic conflict resolution among data of domains of semi-structured documents. In this section, we describe this algorithm for finding domain-conversion paths between elements whose domains are in several types as follows: (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 R to another element y of WIS S 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:

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

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: For instance, let us consider conversion paths between two domains A and B shown in Figure 7.3, where

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

Using *path()* and *bestPath()*, the best domain conversion paths from all subdomains of A to all subdomains of B can be computed as shown in Table 7.2. □

Table 7.2: Relationship and length of all conversion paths JBuilding and BldRm

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

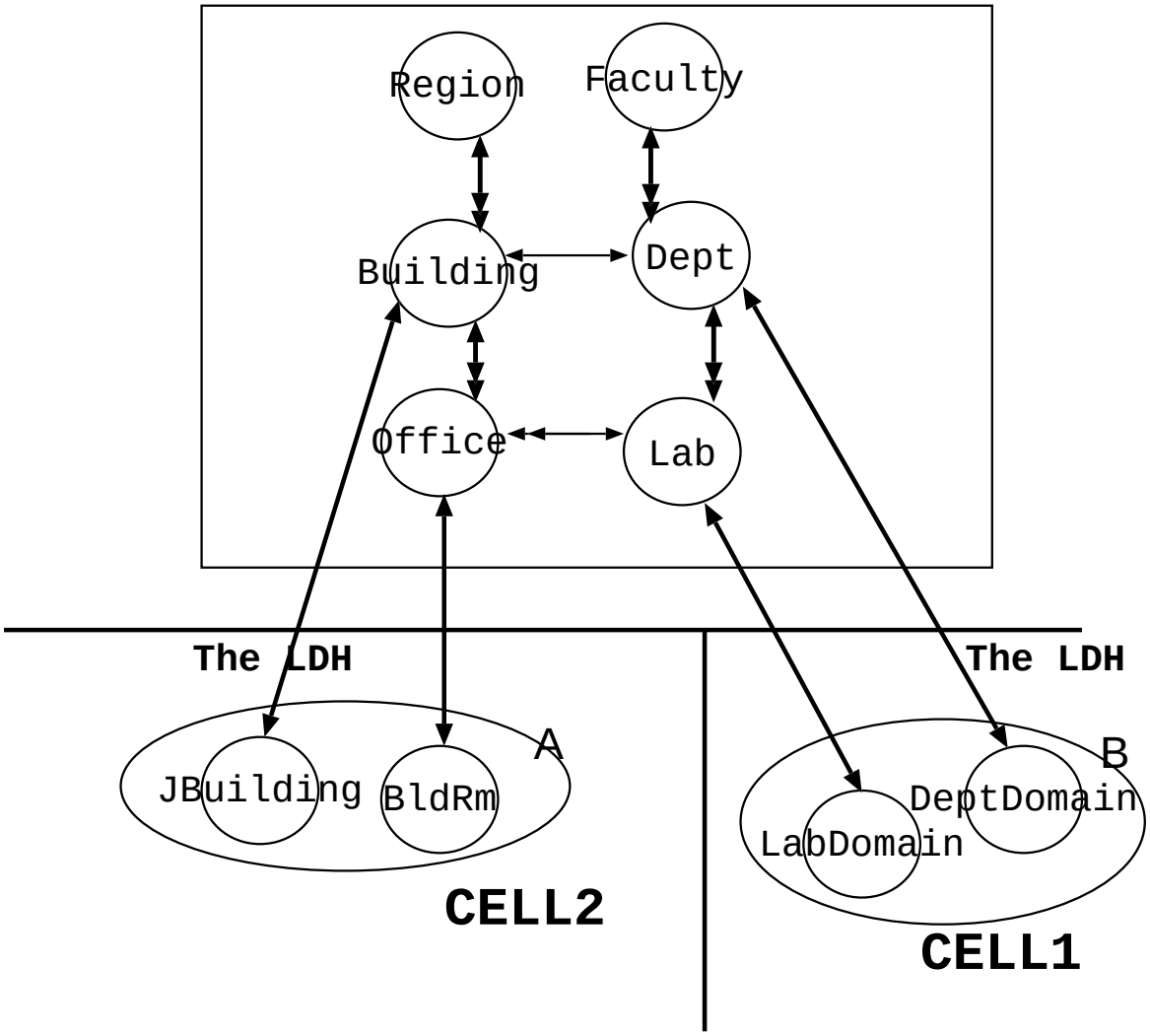


Figure 7.3: Sample of domain hierarchy

the length of the paths. This answer is computed by a function, $filter()$. It receives a set of paths as its argument as follows:

$$FILTER(A,B) \quad = \quad filter(ALLPATH(A,B)).$$

Roughly seeing Table 7.2, all paths in this table are legal but the second path and the third path seem better than the others. By this $filter()$ function, 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. However, in this case, the condition clause for expressing navigational integration cannot be represented in the restricted DNF condition described in section 7.1.3. We will discuss the general forms of navigational integration in terms of path expressions and condition clauses later.

(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 two attributes x and y whose domains are A and B, respectively, and

$$\begin{aligned} A &= FirstName, LastName \\ B &= JFirstName, JLastName. \end{aligned}$$

The domain conversion paths from all sub domains of A to all sub domains of B can be computed as follows:

$$FirstName \rightarrow ... \rightarrow JFirstName$$

$$LastName \rightarrow ... \rightarrow JLastName.$$

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

$$S \rightarrow \{y\} < JFirstName > = f(R \rightarrow \{x\} < FirstName >)$$

and

$$S \rightarrow \{y\} < JLastName > = g(R \rightarrow \{x\} < LastName >),$$

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* by using the algorithm described in chapter 5.

   (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:

$$Int1 \rightarrow ... \rightarrow \{A\} < J > \quad = \quad \mathrm{f}(Int2 \rightarrow ... \rightarrow \{B\} < M >)$$

where $Int1$ and $Int2$ are the root interface name that lead to the attributes A and B, respectively. The mediator does not resolve semantic conflict resolution 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

$$\begin{array}{rcl} Int1 \rightarrow ... \rightarrow \{A\} < J > & = & \mathrm{f}(Int2 \rightarrow ... \rightarrow \{B\} < M >) \\ & \textbf{or} & \\ Int1 \rightarrow ... \rightarrow \{A\}[*] \rightarrow \{P\} & = & \mathrm{g}(Int2 \rightarrow ... \rightarrow \{B\}[*] \rightarrow \{Q\}). \end{array}$$

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. The second condition clause is also 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.

7.1.5 General Form of Navigational Integration Expressed in a Where Clause

Until now, we have proposed the semantic conflict resolution algorithm for supporting the heterogeneous properties that always exist in the semi-structured documents. In addition, we have also proposed how to express the navigational integration in condition clause of our proposed query command. We simplify semantic conflict resolution between two heterogeneous domains by filtering some conversion paths that have less priorities. We express navigational integration in the WHERE clause in a restricted DNF expression. Added with the positions of derived links, the WHERE clause can be written as follows:

$$
\begin{array}{l}
(\\
\quad (R \rightarrow \{r\} < DOM_1 > \textbf{and } subCond_1) \\
\qquad \textbf{or} \\
\quad (R \rightarrow \{r\} < DOM_2 > \textbf{and } subCond_2) \\
\qquad \textbf{or} \\
\quad ... \\
\quad (R \rightarrow \{r\} < DOM_n > \textbf{and } subCond_n) \\
)
\end{array}
$$

where $subCond_n$ is a sub-condition that is in a form $((S \rightarrow methodOfS(pe_{R_n})$ **and** $(pe_{S_{n1}} = pe_{R_{n1}})$ **and** $(pe_{S_{n2}} = pe_{R_{n2}})$ **and** ...). See the detail of sub-condition in section 7.1.3.

The WHERE clause shown above is the restricted DNF condition for expressing navigational integration from a WIS R to a WIS S. The derived links are created from the element r of WIS R where the domain of the element r is a heterogeneous type $(DOM_1|DOM_2|...|DOM_n)$.

As a result, the mediator can automatically generate a query condition of navigational integration in a restricted DNF expression when we simplify the algorithm of semantic conflict resolution. However, in case that users require the exact solution, they must write a query condition in *a general DNF expression*. After all, this needs users' participations.

Here, let us consider the general DNF expression of WHERE clause. This is necessary when the mediator does not perform the filtering process ($filter()$) in the semantic conflict resolution of the heterogeneous domains. In this case, the WHERE clause can be written in the general DNF expression as follows:

$$
\begin{array}{l}
(\\
\quad (R \rightarrow \{r\} < DOM_1 > \textbf{and } subCond_1) \\
\qquad \textbf{or} \\
\quad (R \rightarrow \{r\} < DOM_1 > \textbf{and } subCond11) \\
\qquad \textbf{or} \\
\quad \ldots \\
\quad) \\
\quad (R \rightarrow \{r\} < DOM_2 > \textbf{and } subCond_2) \\
\qquad \textbf{or} \\
\quad (R \rightarrow \{r\} < DOM_2 > \textbf{and } subCond_{21}) \\
\qquad \textbf{or} \\
\quad \ldots \\
\quad) \\
\quad (R \rightarrow \{r\} < DOM_n > \textbf{and } subCond_n) \\
\qquad \textbf{or} \\
\quad (R \rightarrow \{r\} < DOM_n > \textbf{and } subCond_{n1}) \\
\qquad \textbf{or} \\
\quad \ldots \\
\quad) \\
)
\end{array}
$$

Note that, $subCond_{11}$, ..., $subCond_{21}$, ..., $subCond_{n1}$, ... are the semantic conflict resolutions that are filtered out when the simplified answer is preferred.

To write the above WHERE clause for each derived links, we can write in another way as follows:

$$
\begin{array}{l}
(\\
\quad (R \rightarrow \{r\} < DOM_1 > \textbf{and } ((subCond_1) \textbf{ or } (subCond_{11}) \textbf{ or } ....)) \\
\qquad \textbf{or} \\
\quad (R \rightarrow \{r\} < DOM_2 > \textbf{and } ((subCond_2) \textbf{ or } (subCond_{21}) \textbf{ or } ....)) \\
\qquad \textbf{or} \\
\quad \ldots \\
\quad (R \rightarrow \{r\} < DOM_n > \textbf{and } ((subCond_n) \textbf{ or } (subCond_{n1}) \textbf{ or } ....)) \\
)
\end{array}
$$

In conclusion, we have described the enhancements of our system in terms of the common data model, the path expression, and the algorithm of semantic conflict resolution. These enhancements can support the automated semantic conflict resolution of navigational integration among data of semi-structured documents. The mediator

can automatically generate a query condition of navigational integration in a restricted DNF expression by using filter functions to simplify conditions. However, the exact query can be written in a general DNF expression. This requires users' participation.

7.2 At-Once Mode Integration

So far, we have described the integration style called *navigational integration* for the WWW framework. In some cases, users may want to filter and see information without navigating though the original pages. To provide this function, we have implemented another mode called *an at-once mode*. Generally, this mode is equal to the join process that can be found in the traditional database such as relational databases. In this mode, data model, path expressions, and a syntax of query command are the same as those of the navigational integration mode. Users can switch to use this mode easily by filling the target data that they want to select in the select clause of a query command. The result of the execution is represented in the same way as a table in the relational model.

Let us show an example of the at-once mode. Suppose that there are two information sources whose interface definitions are shown in Figure 7.4. A user wants to retrieve the laboratory information and URLs of home pages of each laboratory. To do so, he uses the following query command:

```
select L->{building},L->{lab},S->{urlList}
from L in LabBuilding, S in SearchEngine
source LabBuilding of uecdbWeb on
       http://YYY/cgi-bin/Executor.pl,
       SearchEngine of altavistaWeb on
       http://ZZZ/cgi-bin/Executor.pl
where LabBuilding->getByParam($building,$floor,$lab)
```

```
database uecdbWeb
address http://YYY/cgi-bin/Executor.pl
interface LabBuilding
body:
   building          Building,
   floor             integer,
   lab               JLab,
method:
public:
 static string getByParam(Building $building,
                          integer  $floor,
                          JLab     $lab);
private:
 OID  new(Building $building,integer $floor,JLab $lab)
 void printDatail();
implement:
   ....
endinterface
```

(a) An Interface Definition of the Laboratory WCA

```
database altaVistaWeb
address http://ZZZ/cgi-bin/Executor.pl
interface SearchEngine
body:
   keyword string,
   urlList string,
method:
public:
  static bool searchByKey(string $keyword);
private:
  OID  new(string $keyword,string $URLList);
implement:
    ...
    ...
endinterface
```

(a) An Interface definition of the Search Engine WCA

Figure 7.4: Interface definitions of the Laboratory WCA and the search engine WCA

```
and   SearchEngine->searchByKey(L->{lab}).
```

From this command, the data from two sources are joined by using the L→{lab} attribute. Because the domain required by the first argument of the searchByKey() method is string, no interface rule is executed. The URLs described in *source* clause are the addresses to which the script can be sent to be executed via http protocol. The argument of a method can be bound to input of a front page of a customized WCA. In this example, arguments of the getByBuilding method are replaced by user's input. After completing the command generating, the mediator will convert this command into a Perl script and save it into a file. Let this file name be 'LabHomepage.pl'.

To embed the query script into the front page of the first WIS, the following command is issued.

```
generate LabHomepage.html
from uecdbWeb::LabBuildingForm
where setSubmit('http://YYY/cgi-bin/Executor.pl')
and   setScriptFile('LabHomepage.pl')"
```

The *LabBuildingForm* is the inteface definition wrapping the front page of the first WIS. In general, there are at least two methods that must be defined in the interface definition of front page: (i) the *setSubmit()* method that overwrites the original entry point of the front page (get/post CGI address) to the new CGI address and (ii) the *setScriptFile()* that reads the script file program, embeds it as a hidden field in the HTML form and saves the result as a HTML file. The above command can be automatically performed by the mediator. The value of the argument of the method *setSubmit* is set to the address of the cell specified in the interface definition of the first WIS in default. The value of the argument of the method *setScriptFile* is given by users during the building process of the mediator.

This command generates a new HTML named "LabHomepage.html" shown in the upper part of Figure 7.5. The "LabHomepage.pl" will be embedded into this page and the CGI address of the search button is changed into "http://YYY/cgi-bin/Executor.pl". The variable of building entry on the form is bound to the arguments of getByParam() method. The result of execution is shown in the below part of Figure 7.5.

7.3 Reuse via View Definition

As shown in the above example, we can see that the at-once mode integration can be regarded as the logical WCA of a new WIS. Therefore, reusing it as a new WIS in our framework is one important task. In fact, in relational database concept, a "view" is a virtual (or logical) table defined by other tables. Therefore, we apply the view concept to the at-once mode integration.

We provide the command for defining a view. The general systax of the view definition is:

create view *view-name* **as**
select *targeted-attributes*
from *object-variable* in *interface-name*
[,*object-variable* in *interface-name*]
source *interface-name* of *database-name* on *address*
[, *interface-name* of *database-name* on *address*]
[**where** *object-condition* or *method-call*],

where *view-name* is the view name and the select command has the same syntax of one defined in the chapter 4. When this view is executed, the query following the **as** is not executed; instead, a new interface definition named *view-name* is created.

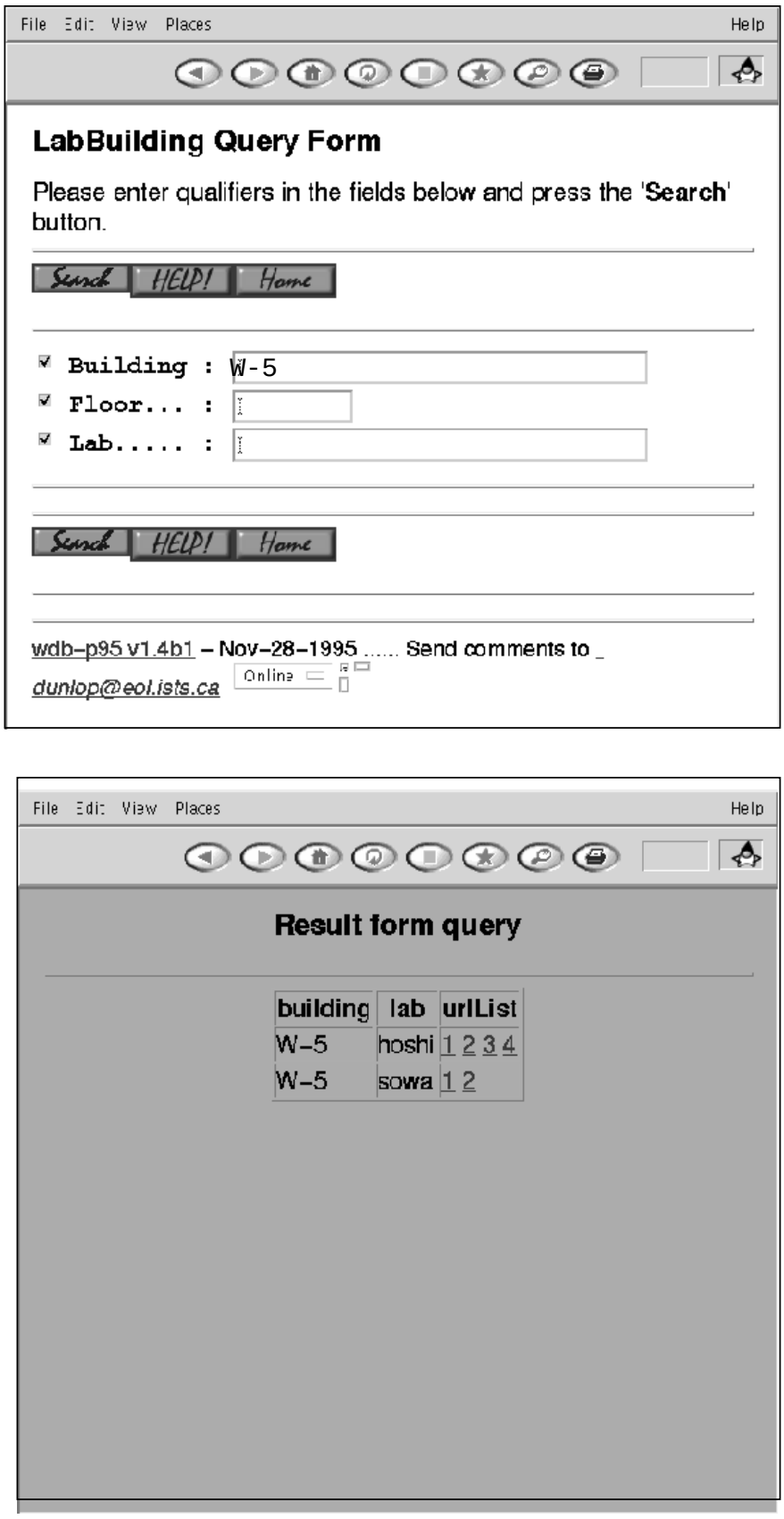


Figure 7.5: A customized WECA

Attributes in the body part of the new interface defintion are the *targeted-attributes* in the **select** clause. The content of the static method is replace by the query command. To the user, however, it is now as if there were an interface definition called *view-name*.

Let us illustrate the semantics of these statements. Suppose we want to create a new interface definition named *LabHomepage* where the query command is the at-once integration shown above. We can generate it by the following statement:

```
create view LabHomepage as
select L->{building},L->{lab}, S->{urlList}
from L in LabBuilding, S in SearchEngine
source LabBuilding of uecdbWeb on
       http://YYY/cgi-bin/Executor.pl,
       SearchEngine of altavistaWeb on
       http://ZZZ/cgi-bin/Executor.pl
where LabBuilding->getByParam($building,$floor,$lab)
and   SearchEngine->searchByKey(L->{lab}).
```

The effect of this command is to create a new interface definition called *LabHomepage* (LabHomepage.def as its file name) shown in Figure 7.6. It has three attributes called *building*, lab, and urlList, corresponding respectively to the three targeted attributes in the select clause. In addition, its static method is replaced by the script program generated from the query command.

As the result, we can use the LabHomepage as the new interface definition desribing a new WIS in our framework. The below is a query command of navigational integration mode that is similar to the command in the chapter 4 but uses the new interface definition as the second source.

```
select *
from   D in Department, L in LabHomepage,
```

```
database  uecdbWeb
address  http://YYY/cgi-bin/Executor.pl
interface   LabHomepage
body:
     building          Building,
     lab               JLab,
     urlList           string
method:
public:
   static string getByParam(Building $building,
                            integer  $floor,
                            JLab     $lab);
 private:
   OID  new(Building $building,integer $floor,JLab $lab);
   void printDatail();
 implement:
   sub getByParam{
       my $package = shift;
       my $building = shift;
       my $lab     = shift;
       my @result;
       ....
       ....

       @result = query(create view LabHomepage as
                       select L->{building},L->{floor},L->{lab},
                              S->{urlList}
                       from L in LabBuilding, S in SearchEngine
                       source LabBuilding of uecdbWeb on
                              http://YYY/cgi-bin/Executor.pl,
                              SearchEngine of altavistaWeb on
                              http://ZZZ/cgi-bin/Executor.pl
                       where LabBuilding->getByParam($building,
                                                     $floor,$lab)
                       and   SearchEngine->searchByKey(L->{lab})
                  )
       return @result;
   }


 endinterface
```

Figure 7.6: A new interface definition created by View

```
source Department of ISInformation on
       http://HOST1/cgi-bin/execute.pl
       LabBuilding of uecdbWeb on
       http://YYY/cgi-bin/Executor.pl,
where  Department->getByURL($URLx)
and    LabHomepage->getByParam('','',
         &L:Lab2JLab(&D:LabDomain2Lab(
           D->{laboratories}->{labName}))).
```

7.4 Topics of Further Research

So far, we have modeled the Web information sources by a set of interface definitions to wrap the front page and the result page. Nested structure in a Web pages can be modeled using relationships between two interface definitions. In general, information provided in a Web site is not represented in a simple form of one front page linked to its result pages. The result pages can contain links to other pages so that users can repeatly navigate through a sequence of result pages. To achieve this, we here describe the extension in terms of data model and path expression.

Data Model: To support multi-level result pages, we add a new type, *link*, into our data model. Unlike the the *setof* or the *refto* type that are used to specify objects in the same page, the *link* type is used to specify objects in the different pages.

Let us consider an example for modeling links from result pages in Figure 4.1. Assume that there is a link at each laboratory name to the Web page of each laboratory. Then, the body part of the interface definition of laboratory element shown in Figure 4.3-b must be changed as follows:

```
database    ISInformation
address     http://HOST1/cgi-bin/execute.pl
interface   Laboratory
body
  labName   LabDomain,
  toLabPage link Labpage,
  staffs    setof Staff,
method
....
```

Figure 7.7 shows instance of the department in our model. The dash line represents a link that its successor object is not retrieved until they are referred to. It can be regarded as the lazy pointer swizzling in the object database concept [12].

Path Expression: To express the link that navigate between different pages, the operator ($\Rightarrow$) is used to follow links. Suppose we are interested in the projects in each laboratory. To reach data of interest, we can express in term of path expression as follows:

$$Department \rightarrow \{laboratories\}[*] \rightarrow \{toLabPage\} \Rightarrow \{project\}[*] \rightarrow \{projectName\}$$

In implementation, the $\Rightarrow$ is a binary operator that takes as input two arguments, URL and $interfacename$ that of the page the link points to. The original links are replaced with CGI string with these two parameters. The execution of operation is to download the page according to the URL and to extract data from the page by the wrapper of the $interfacename$. After embedding the objects into the result page, the wrapper sends the result page back to the proxy executor as described in chapter 4.

With this enhancement, our integration can be applied to the general Web information that consists of one front page and multi-levels of result pages.

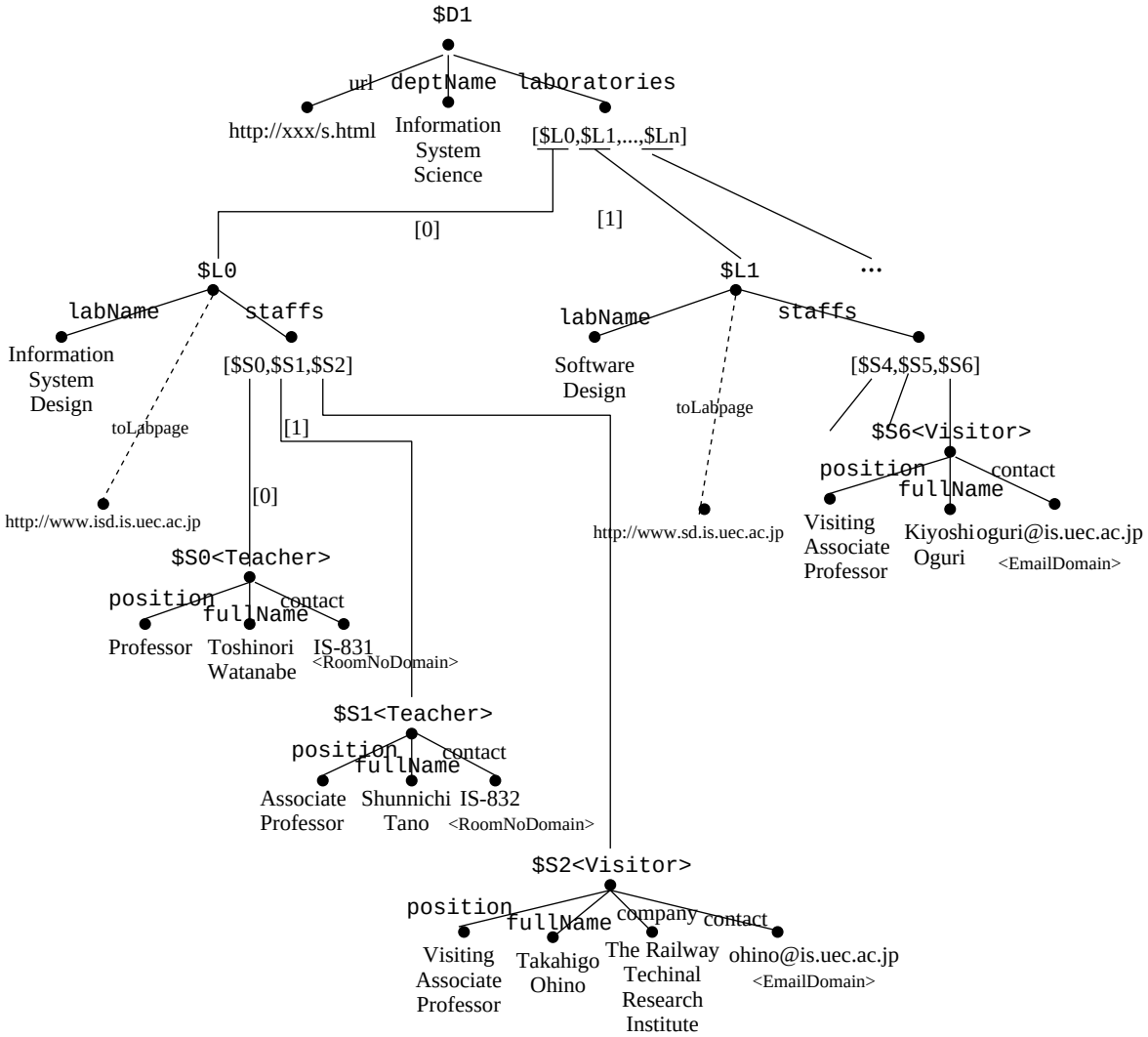


Figure 7.7: New interface definition and an instance example for external link

7.5 Summary

This chapter proposes enhancements for supporting semi-structured data in term of a common data model, path expressions, and semantic conflict resolution algorithm. We describe the properties that do not exist in structured documents. These properties are: (i) a heterogeneous domain, (ii) a heterogeneous structure, and (iii) a concatenation domain. These properties can be described by the DTD (Document type definition). To map DTD into our common data model, we have enhanced our common data model to support the 'OR' or 'OPTION' in DTD. Thereafter, we have enhanced our path expressions for specifying domains and proposed the operators on path expressions. Based on the common data model and extended path expressions, we have proposed the restricted DNF expression for specifying derived links in navigational integration. The semantic conflict resolution is also extended for handling properties of semi-structured documents. In addition, the general DNF expression of navigational integration in a WHERE clause is also described.

To provide the same mode of integration as that of the traditional database approach, the at-once integration is proposed. We have also described the way of reusing it using a view declaration. Moreover, the topic of further research concerning enhancement for supporting multi-level result pages are discussed. These enhancements make our approach so rich that it can be used to apply to model services provided in today's WWW.

Chapter 8

Conclusion

This thesis has proposed a system for integrating autonomous WISs by mobile users. Most of previous works were proposed for a static and centralized integration system that cannot serve our requirements. To achieve our goal, we have proposed several ideas to improve the current technologies. The major results of our work can be summarized as follows:

- To provide a new integration style for the WWW framework, the *navigational integration* has been proposed. This is a process to embed invocations of server-access functions of one WIS as derived links into a result page of another WIS.

- To provide mobile end-users with an ability to define navigational integration on their mobile computers in a disconnected mode, this thesis has proposed a new system architecture. This system architecture is based on the wrapper/proxy executor/mediator architecture. Likewise traditional architectures, the wrapper module is placed in a remote server for wrapping a WIS. The different point is that the mediator module is installed in a mobile computer. The mediator in our approach performs functions of defining integration, resolving semantic conflict and generating a new client application. The function that executes requests sent

from users is placed in a stationary remote server. In this thesis, it is called *a proxy executor*.

- To facilitate ability of automated data conversion among independently-developed autonomous organizations, this thesis has proposed the *domain hierarchy* as a common ontology among these organization.

- A common data model and path expression extended with the concept of domain hierarchy have been proposed. Our data model supports not only the data conversion but also structured/semi-structured data appearing in the WWW.

- The download algorithm for acquiring necessary information from WISs has been proposed. This thesis has shown that the minimum amount of necessary information should be downloaded from WISs to the mediators placed in mobile computers. Consequently, mobile users can resolve semantic conflict and define the navigational integration among WISs on their mobile computers in a disconnected mode.

- The semantic conflict resolution algorithm has been proposed. This thesis gives details of how to use the downloaded information to find out the conversion path for resolving semantic conflict between two data-unit. Reachability of a conversion path between two domains of data has been described. The legality of the conversion path has been proposed for preventing ambiguous meaning of data conversion.

- To support navigational integration for semi-structured Web documents, this thesis has also proposed enhancements in term of a common data model, path expressions, and algorithms for resolving semantic conflict.

It should be noted that our major originality is the navigational integration of WISs defined by mobile users solely on mobile computers. Representation style of the navigational integration is based on navigating via links as integration. It is also well suited to the WWW framework. Moreover, semantic conflicts between data of different WISs are automatically resolved on users' computers without network-connections. We have not found any work that concentrates on the same requirements as ours. In our approach, organizations (cells) offer packages wrapping WISs and export them to inform mobile users of their services. The mobile users then have the freedom to use the packages when they see that these packages are suitable to their need. The mobile users can create *customized* WCAs that integrate the required WISs. As a result, the mobile users can present their requests and get the answers in a manner suitable to themselves. Thus, in our approach, mobile users can combine the services of several organizations rather than simply using the services provided by an individual organization.

Today's networks already contain a lot of WISs capable of providing specific services. The number of such WISs will continue to grow and they are autonomous. At the same time, recent hardware technologies make the size of computers smaller so that users can take them while moving. Currently, using such devices to access services from public networks is a common way that can be seen everywhere. Like the traditional databases, we expect that there is need for mobile users to integrate these WISs by themselves. We believe that our framework would be the key scheme in making such use truly practical.

Bibliography

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

[2] Altavista Technology Home Page. http://www.altavista.com.

[3] A. Silberschatz, M. Stonebraker, J.D. Ullman ed.: "Database Systems: Achievements and Opportunities." In *ACM SIGMOD Record*, Vol.19 (No.4), pp. 6–22, 1990.

[4] C. J. Date. "An introduction to database management systems (sixth edition)." *Addison-Wesley Publishing Company*, 1994.

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

[6] 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.

[7] D. Quass, et al.: "Querying Semistructured Heterogeneous Information." Available by ftp at *db.stanford.edu as the file /pub/quass/1994/querying-submit.ps.*

[8] 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.

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

[10] 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.

[11] G. Wiederhold: "Interoperation, Mediation, and Ontologies." In *Proceedings International Symposium on Fifth Generation Computer Systems (FGCS94), Workshop on Heterogeneous Cooperative Knowledge-Bases*, Vol.W3, pp. 33–48, ICOT, Tokyo, Japan, Dec. 1994.

[12] J. Eliot B. Moss. "Working with Persistent Objects: To swizzle or not to Swizzle." In *IEEE Trans. Software Engineering* Vol.18, pp. 657–673, Aug 1992

[13] J. Gosling, H. McGilton. The Java Language Environment *A white paper*. Sun MicroSystems Co.

[14] L. Hass, et al. "Optimizing queries across diverse data sources." In *Proceeding of the 23th VLDB Conference*, Athens, pp. 276–285, Greece, 1997.

[15] L. Wall, T. Christiansen and R. L. Schwartz. "Programming Perl." *O'Reilly & Associates, Inc*, 1996.

[16] M. C. Shan, et al. "Pegasus: A Heterogenous Information Management System." In *Modern Database Systems* Addison Wesley pp. 665–682, 1995

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

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

[19] 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, Greece, pp. 266–275, 1997.

[20] 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.

[21] N. Walsh. "Learning Perl/Tk." *O'Reilly & Associates, Inc*, 1999.

[22] O. Etzioni and D. Weld. "A Softbot-based Interface to the Internet." In *Communications of the ACM*, Vol 37, No 7, pp. 72–76, July 1994.

[23] 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.

[24] R. Goldman, J. Widom: "DataGuides: Enabling Query Formulation and Optimization in Semistructure Databases." In *Proceeding of the 23th VLDB Conference*, Athens, pp. 436–445, Greece, 1997.

[25] S. Abiteboul.: "Querying semi-structured data." In *Proceedings of the Int. Conf. on Database Theory (ICDT),* Delphi, Greece, pp. 1–18, 1997.

[26] S. Bressan, C.H. Goh, et al.: "The COntext INterchange Mediator Prototype." In *Proc. of the ACM SIGMOD Conf. on Management of Data*, Washington, USA, pp. 525–527, 1997.

[27] S. Chawathe, et al.: "The TSIMMIS project: Integration of heterogenous information sources." In *Proceedings of the ACM the 100th IPSJ,* Tokyo, Japan, pp. 7–18, Oct 1994.

[28] 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.

[29] S. ST. Laurent., "XML A primer". *MIS:Press, An imprint of IDG Books Worldwide*, 1998.

[30] T. Berners-Lee, et. al., "The World Wide Web Initiative." In *Proc. INET '93*, 1993.

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

[32] T. Milo and S. Zohar. "Using schema matching to simplify heterogeneous data translation." In *Proceeding of the 24th VLDB Conference*, New York City, USA, pp. 122–133, 1997.

[33] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. "From structured documents to novel query facility." In *Proc. of the ACM SIGMOD Conf. on Management of Data*, Minnesota, USA, pp. 313–324, 1994.

[34] 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

[35] V. Vassalos and Y. Papakonstantinou "Describing and Using Query Capability of Heterogeneous Sources." In *Proceeding of the 23th VLDB Conference*, Athens, Greece, pp. 256–265, 1997.

[36] 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.

[37] W. Sae-Tung, T. Ohmori, and M. Hoshi. "An information integration architecture for mobile users in WWW environment." In *Trans. Information Processing Society of Japan*, 39(4): pp. 888–900, April 1998.

[38] W. Sae-Tung, J. Ge, T. OHMORI, M. HOSHI, "Integrating Web Information Sources for Mobile Users(1) - System Architecture." 情報処理学会第 58 回全国大会 3T-8, 1999 年 3 月.

[39] W. Sae-Tung, T. Ohmori, M. Hoshi "Integration of Web Information Sources by Mobile Users." 第 10 回データ工学ワークショップ'99 論文集、4C-4, 電子情報通信学会、1999 年 3 月.

[40] 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.

[41] Y. Papakonstantinou, H. Garcia-Molina, J.D. Ullman: "Medmaker: A mediation system based on declarative Specifications." Available by ftp at *db.stanford.edu as the file /pub/papakonstantinou/1995/medmaker.ps.*

[42] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J.D. Ullman: "A query translation scheme for rapid implementation of wrappers." In *Proc. DOOD Conf.*, 1995. Available by ftp at *db.stanford.edu as the file /pub/papakonstantinou/1995/transgen.ps.*

[43] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina: "Object fusion in mediator system." In *Proc. VLDB Conf.*, 1996. Available by ftp at *db.stanford.edu as the file /pub/papakonstantinou/1996/fusion.ps.*

[44] Y. Papakonstantinou, et al.: "Object exchange across heterogenous information sources." In *IEEE Int. Conf. Data Engineering '95.*, pp. 251–260, 1995.

[45] Y.E. Ioannidis and Y. Lashkari. "Incomplete path expressions and their disambiguation." In *Proc. of the ACM SIGMOD Conf. on Management of Data*, Minnesota, USA, pp. 138–149, 1994.

Acknowledgments

This work is inspired, taught or assisted by numerous people. I would like to thank each of them for invaluable advice, intellectual exchanges, concrete suggestions and technical assistance.

First of all, I would like to express my gratitude to Associate Professor Tadashi Ohmori, my supervisor, who has given me advice and encouragements throughout the whole work. I would also like to express my gratitude to Professor Mamoru Hoshi for his significant discussions and comments on many topics, which inspire me to greater efforts.

I am grateful of Professor Mamoru Maekawa, Professor Toshinori Watanabe, Professor Toshio Okamoto and Professor Toshitsugu Yuba for their valuable suggestions and comments to make the dissertation more readable.

During the course of this study, I have received technical assistance from staffs in my laboratory. In particular, Research Assistant Kensuke Ohnishi and Ph.D student Michihiro Kobayakawa provide me suitable network environment and help me solve some problems in the experiment.

I would like to thank my family, my mother and my younger brother. I took a long time away from them for seven years in Japan. I appreciate my mother's understanding and patience during this long time. I would like to thank my brother's who has taken care of our mother very well during the long time.

Finally, I would like to thank the Japan Ministry of Education for continued financial assistance.

Author Biography

Wisut Sae-Tung was born in Bangkok, Thailand, on January 22, 1966. He received the B.E. degree from Chulalongkorn University, Bangkok, Thailand, in March 1988, and the M.E. degree from the University of Electro-Communications, Tokyo, Japan in March 1996. He has been with the Department of Information System Science, Graduate School of Information Systems, The University of Electro-Communications, Tokyo, working towards the Ph.D. degree. From 1999 he joined the Graduate School of Information Systems as an assistant staff. His research interests include heterogeneous information system, mobile computing, programming language and database technology.

Mr. Wisut is a member of Information Processing Society of Japan(IPSJ).

List of Publications Related to the Dissertation

Journal Paper

- Wisut Sae-Tung,Tadashi Ohmori, Mamoru Hoshi "An Information Integration Architecture for Mobile Users in WWW Environment", Transaction of Information Processing Society of Japan (IPSJ) Vol.39, No.4, pp.888-900, April 1998.

International Conference Paper

- Wisut Sae-Tung,Tadashi Ohmori, Mamoru Hoshi "Navigational Integration of Autonomous Web Information Sources by Mobile Users", 1999 IEEE Pacific Rim Conference on Communications, Computers and Signal Proocessing, Victoria, B.C, Canada (PACRIM 1999), pp. 270–275, August 1999.

Domestic Workshop

- Wisut Sae-Tung, Tadashi Ohmori, Mamoru Hoshi "An Information Retrieval Architecture for Mobile Users: An Experimental Report", in the proceeding of IEICE Data Engineering Workshop (DEWS'97), pp.79-84, March 1997.
- Wisut Sae-Tung, Tadashi Ohmori, Mamoru Hoshi " Integration of Web Information Sources by Mobile Users", in the proceeding of IEICE Data Engineering Workshop (DEWS'99), 4C-4, March 1999.

Convention Papers

- Wisut Sae-Tung、 大森 匡、星 守、”スクリプト言語を用いた移動計算機向けの永続オブジェクトシステム”, 情報処理学会第 51 回全国大会 6D-5, 1995 年 9 月.
- 大森 匡、Wisut Sae-Tung, 星 守、”スクリプト言語による移動計算機向け永続オブジェクトシステム, 電子情報通信学会データ工学研究会資料、DE95-54, 1995 年 10 月.
- 大森 匡、Wisut Sae-Tung, 星 守、”移動計算機における情報ベース検索スクリプトの合成方式”, 情報処理学会第 52 回全国大会 7P-4, 1996 年 3 月.
- Wisut Sae-Tung, Tadashi Ohmori, Mamoru Hoshi, ”An Information Retrieval Architecture for Mobile Computers based on a Persistent Script Language”, 情報処理学会データベースシステム研究会資料、 96-DBS-109-45, 1996 年 7 月.
- Wisut Sae-Tung, Junkai Ge, Tadashi OHMORI, Mamoru HOSHI, ”Intergrating Web Information Sources for Mobile Users(1) - System Architecture”, 情報処理学会第 58 回全国大会 3T-8, 1999 年 3 月.
- Junkai Ge, Wisut Sae-Tung, Tadashi OHMORI, Mamoru HOSHI, ”Intergrating Web Information Sources for Mobile Users(2) - Document Applications”, 情報処理学会第 58 回全国大会 3T-9, 1999 年 3 月.

Contents

1 Introduction **1**

1.1 Background and Motivation 1

1.2 An Example of Integration of Our Approach 3

1.3 Technical Problems and Principle of Our Solutions 7

1.4 Points of Originality 12

1.5 Organization of Thesis 13

2 Related Works **15**

2.1 Query Languages for the Web 15

2.2 Heterogeneous Information Integration 18

2.3 Web Site Management Systems 23

3 Basic Idea and System Concept **27**

3.1 Problem Statements and Strategies 27

3.2 Basic Idea 30

3.2.1 Example of Navigational Integration 30

3.2.2 Basic Idea to Define Navigational Integration 32

3.3 A Component Model of WIS 38

3.4 An Integration Model 40
3.5 Execution Model 42
3.6 Components in System Architecture 46
3.7 Region Design for Semantic Conflict Resolution 49
3.8 Summary 50

4 System Design and Implementation **51**

4.1 Common Data Model 51
4.2 Path Expressions 60
4.3 Navigational Integration 61
4.3.1 Definition 61
4.3.2 Modes of Navigational Integration 62
4.3.3 Implementation 65
4.4 A Query Language for Navigational Integration 67
4.5 Mediator 69
4.6 Proxy Executor and Wrapper 71
4.7 Summary 78

5 Domain Hierarchy for Resolving Semantic Conflict **81**

5.1 A Domain Hierarchy 81
5.2 Construction 83
5.3 Package Download 88
5.4 Reachability of Conversion Paths 90

5.5 Legality of Conversion 93
5.6 Semantic Conflict Resolution 94
5.7 Syntax of a Domain Rule and an Example 97
5.7.1 Syntax of a Domain Rule 97
5.7.2 An Example 102
5.8 Concatenation Relationship 103
5.9 A Domain of Complex String 107
5.10 Execution Plan due to Semantic Conflict Resolution 108
5.11 Extensibility for Handling Multi-regions 111
5.12 Summary 112

6 System Prototype **115**

6.1 Prototype Configuration 115
6.2 A Region Design for a university 118
6.3 Applicability of System 121
6.3.1 Effectiveness of Wrapper 121
6.3.2 Efficiency and Effectiveness of Proxy Executor 122
6.3.3 Efficiency and Effectiveness of Mediator 124

7 Enhancements **125**

7.1 Enhancements for Supporting Semi-structured Data 125
7.1.1 The Common Data Model 126
7.1.2 Extended Path Expression 131

7.1.3 A Navigational Integration in Terms of Path Expressions 134
7.1.4 Extended Semantic Conflict Resolution 136
7.1.5 General Form of Navigational Integration Expressed in a Where Clause 144
7.2 At-Once Mode Integration 147
7.3 Reuse via View Definition 150
7.4 Topics of Further Research 154
7.5 Summary 157

8 Conclusion **159**