# Navigational Integration of Autonomous Web Information Sources by Mobile Users

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

**Abstract**

*This paper presents a new style of integration called* navigational integration *for mobile users to integrate several information sources in the WWW framework. We illustrate basic ideas of the navigational integration and describe a design of a system architecture that executes the navigational integration.*

## 1 Introduction

In today's computer internetworks, many information sources provide their contents via WWWs by embedding their client applications into Web pages. For example, ordinary HTML document servers, CGI-form texts with backend database servers, and java-based clickable maps with backend data services can be found everywhere in the Internet. Let us call such an information source *a Web Information Source* (WIS). In general, these WISs are maintained by multiple autonomous organizations. Let us call such an organization a *cell*. In this environment, mobile users move across cells and download client applications from WISs to query information. This environment is shown in Figure 1.

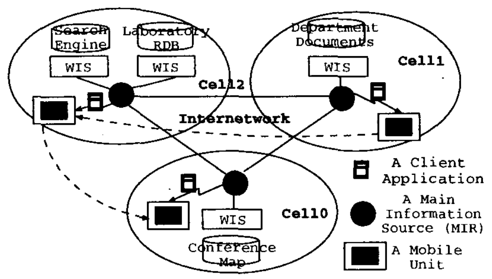


Figure 1: Web Information Sources in a mobile environment

This paper proposes an appropriate style of integration for mobile users who move from one place to another, find new WISs and generate integration among those WISs by themselves on their mobile computers in a disconnected mode. Our integration style, called *navigational integration*, is based on the hyperlink concept of the WWW framework and promotes utilization of filtering methods provided by each WIS. Previous works[2, 1, 4] discuss integration of heterogeneous information sources (including WISs) on a centralized site of developers but do not consider an appropriate style of integration for mobile users.

The paper is organized as follows. Section 2 presents an example of the navigational integration and describes goal of our work. Section 3 describes a system overview. Section 4 describes a system design. Section 5 describes an automatic semantic-conflict resolution that is a necessary mechanism for mobile users to define their integration, followed by concluding remarks.

## 2 Navigational Integration

### 2.1 Example

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

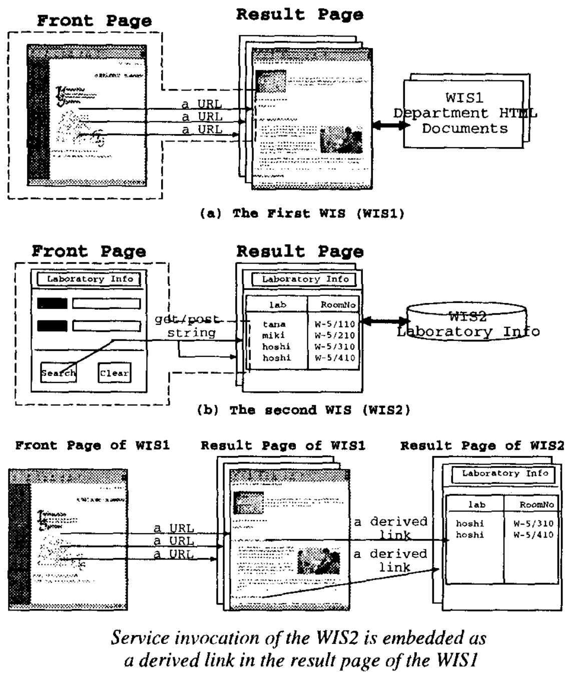


Figure 2: Navigational integration

he can navigate between contents of the web pages of two WISs.

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

## 2.2 The Goal

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

Let us return to Figure 1 and show the overall processing steps taken by mobile users. In this environment, mobile users move across cells and collect client-parts of WISs. Next, they can disconnect their

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

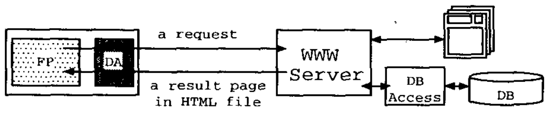


Figure 3: Components of a WIS

computers from networks and define navigational integration by using original client-parts on their mobile computers. Finally, they reconnect their mobile computers to the networks and execute new client-parts to perform navigational integration. In Section 3, we describe an architecture for realizing this execution environment.

## 3 System Overview

### 3.1 Model of a WIS

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

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

### 3.2 Our Approach to Define Navigational Integration

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

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

To define navigational integration in a declarative form, we use a query command whose WHERE clause specifies predicates that define how to express derived

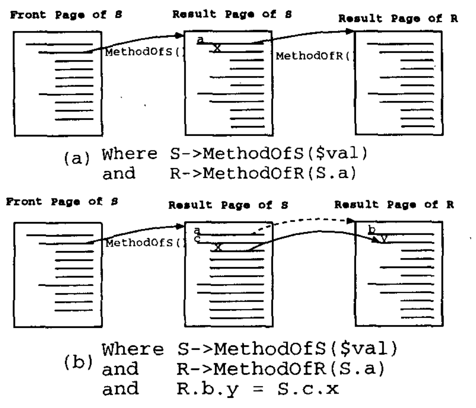


Figure 4: Two modes of navigational integration (written by WHERE clauses)

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

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

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

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

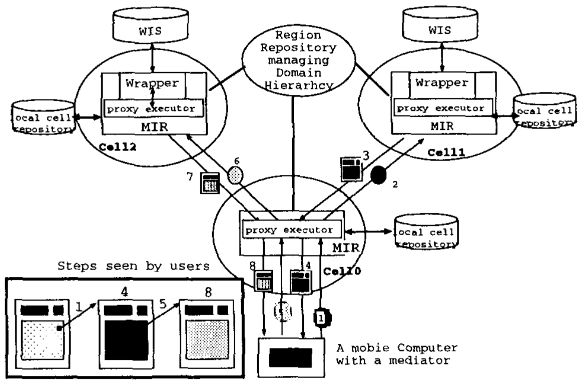


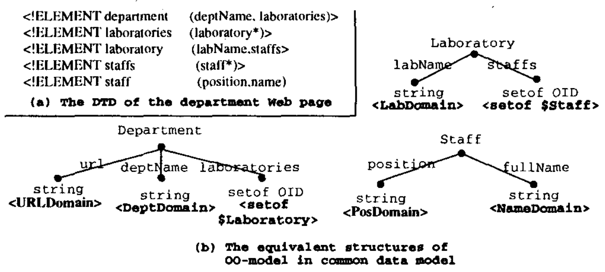
Figure 5: Processing of navigational integration

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 $S$. In the next subsection, we describe a system architecture for executing the query and its processing steps.

## 3.3 System Architecture and Integration Process

To explain a system architecture and the cooperation between its modules for executing navigational integration, let us use Figure 5. In this figure, there are three cells. All cells share a common knowledge called *domain hierarchy* for automatically resolving semantic conflict of data between cells (see detail in section 5). Each cell has a *main information resource* (MIR) as a yellow-page server. In addition, a *wrapper* module that wraps a WIS is placed on the MIR. The client-part of each WIS is wrapped into interface definitions described in subsection 3.2. In the existing integration systems[4, 2], they place a mediator module on a server for: (i) resolving semantic conflict among data of WISs, and (ii) executing a user's query. In contrast, we handle these two works by using two different modules. For the first task, we place a *mediator* module on the mobile computer side. Our *mediator* uses the downloaded client-parts of WISs, helps a mobile user to solve semantic conflict among data, and define navigational integration in a query command. Further, it automatically embeds the query command into the front page of WIS1. For the second task, we place the module called *a proxy executor* for performing this task on the MIR.

Then, a user can execute navigational integration by loading the new client into his browser. The steps



```
<!ELEMENT department      (deptName, laboratories)>
<!ELEMENT laboratories    (laboratory*)>
<!ELEMENT laboratory      (labName,staffs>
<!ELEMENT staffs           (staff*)>
<!ELEMENT staff            (position,name)
```

(a) The DTD of the department Web page

(b) The equivalent structures of OO-model in common data model

Figure 6: A DTD and an equivalent schema

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

## 4 System Design

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

### 4.1 Common Data Model

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

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

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

In order to explain interface definitions, let us use the interface definition of *Department* shown in Figure 7. This interface definition has three attributes:

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

Figure 7: An interface definition for Department

*url*, *deptName* and *laboratories*. The second attribute is a single-value of string and its domain is *DeptDomain*. The third attribute is a multi-value keeping OIDs of instances of a Laboratory interface. Using such a multi-valued attribute, we can represent nested structure of documents. The first attribute *url* is an additional element keeping the URL of the original document page. It does not appear in the DTD shown in Figure 6-a.

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

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

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

### 4.2 Path Expressions and Query Language

A path expression has the form

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

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

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

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

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

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

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

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

As explained in Section 3.3, the mediator automatically embeds the query command into the front page of the first WIS. This is easily done in case of ordinary Web pages or CGI-form pages by replacing the execution tag of HTML source with the query command. In case of a Java applet, it requires the template to generate a Java program that inherits the original applet and overrides the methods that act as the DA part of the first WIS.

## 5 Semantic Conflict Resolution and Script Generation

To provide mobile users with ability for defining navigational integration, the semantic conflict among data of different WISs must be resolved automatically. To achieve this ability, we propose a *domain hierarchy* (DH), which is maintained among all cells in a given network. Here, a *domain* in our approach refers to a pool of values that have an exact data-format and meaning. Figure 8 shows an example of the domain hierarchy, which is maintained by two cells. The domain hierarchy consists of two components: (i) multiple local domain hierarchies (LDHs); each LDH is maintained individually in each cell (shown as the local cell repository in Figure 5), and (ii) a common domain hierarchy (CDH); this is shared by all cells (shown as the region repository in Figure 5). When multiple cells share the same CDH, these cells are called a *region.* The domains in each LDH are defined for describing format/meaning of data used in that cell while the domains in the CDH are defined as a set of standard data formats/meaning for data exchanging among data of those cells.

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

**Construction:** Now, we describe how to build a domain hierarchy. Domains are organized into a hierarchical structure, where a type of hierarchical relationship is a one-to-one relationship (data-format conversion), or an is-a relationship, or a part-of relationship. Therefore, for any given domain $D$, its parent domain is equal to, is more general than, or contains $D$. The *hierarchy links* (depicted by bold links in Figure 8) represent such hierarchical relationships.

As a result, for any given domain $D$, $D$ has a hierarchical relationship with its parent. However, $D$ may have other relationships with some domains $D_i'$

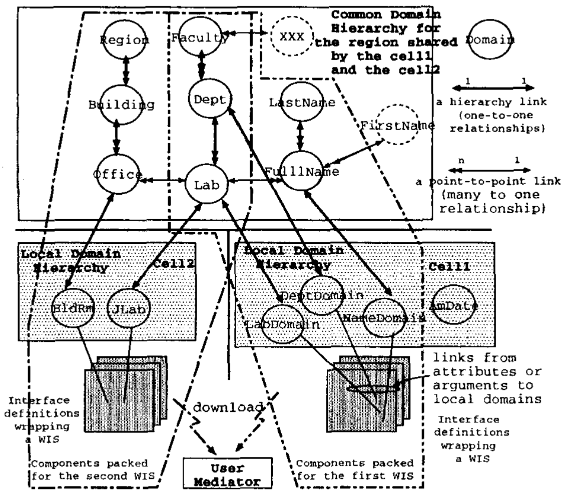

Common Domain Hierarchy for the region shared by the cell1 and the cell2
Region
Building
Office
Faculty
Dept
Lab
XXX
LastName
FirstName
FullName
Domain
1 1 a hierarchy link (one-to-one relationships)
n 1 a point-to-point link (many to one relationship)
Local Domain Hierarchy
Cell2
BldRm
JLab
Local Domain Hierarchy
Cell1
DeptDomain
LabDomain
NameDomain
AmDate
links from attributes or arguments to local domains
Interface definitions wrapping a WIS
Interface definitions wrapping a WIS
download
Components packed for the second WIS
Components packed for the first WIS
User Mediator

Figure 8: Domain hierarchy and downloading mechanism

($i = 1, ..., n$). These relationships are described as *point-to-point links* (depicted by fine links in Figure 8) between $D$ and $D_i'$. Therefore, $D$ can have one hierarchy link to its parent and several point-to-point links to $D_i'$. (In implementation, each link is described by conversion functions for converting semantics between the two associated domains. ) We refer to these links (with their conversion functions) as the *domain rule* of $D$.

**Mediator and Script Generation:** As described in Section 2.2, a mobile user must download software components describing services of WISs from their cells into his mobile computer. This *software component* of a WIS consists of (i) a front page of the WIS, (ii) interface definitions of the WIS, (iii) domain rules of the domains used in these interface definitions, (let us refer to these domains as $X$), and (iv) the domain rules of the ancestor domains of $X$. Therefore, as a result of downloading, a sub-part of the DH is maintained in the user's mobile computer. The download processing is shown in Figure 8. In this figure, the domains and interface definitions surrounded by the dotted line are packed and downloaded into the mobile computer.

The mediator on the user's computer uses the downloaded sub-part of the DH, and it resolves the semantic conflict between two data of different WISs. To do so, the mediator must find conversion paths between domains of the conflicting data, and choose the shortest one as a result. In our scheme, a conversion path between two domains is assured to exist if there is: (i) a domain that is an ancestor of both domains; or (ii) only one point-to-point link between ancestors of both domains.[2] For example, in Figure 8, the conversion path from the domain *LabDomain* to the domain *JLab* is LabDomain $\rightarrow$ Lab $\rightarrow$ JLab. The conversion functions along the conversion path are inserted to convert the semantics between them.

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

According to the above algorithms, the mediator in our system automatically resolves semantic conflict and generates an appropriate query even in a communication-disconnected mode.

## 6 Concluding Remarks

In this paper, we have proposed navigational integration as a new style of integration for mobile users in the WWW framework. We have also proposed the system architecture that is modified from the wrapper/mediator architecture for supporting the navigational integration. For resolving semantic conflict among autonomous WISs, a distributed repository mechanism called *domain hierarchy* is proposed. Currently, a prototype of our system works for various WISs, including WWW document servers, CGI with database servers, and clickable maps.

This paper discussed semantic conflict resolution among atomic data-values in structured documents, but our approach can be extended for semantic conflict resolution in heterogeneous structures and heterogeneous domains. This will be presented in [5].

## References

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

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

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

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

[5] W. Sae-Tung. "Integration of Autonomous Web Information Sources by Mobile Users". *Ph.D. Thesis*, UEC (in preparation), 1999.

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