データベースシステム 93－4
システムソフトウェアと
オペレーティング・システム 59－4
（1993．5．27）

データ知識協調モデルのオブジェクト指向方式による実現法

彭智勇　上林弥彦

京都大学工学部

データ・知識協調モデルが分散したデータベースと知識ベースの統合のために導入された。本稿では、このモデルをSmalltalkシステムを中心に実現する方法について述べる。Smalltalkを用いると動的な変更を扱える自由度があり、また利用者インタフェースの開発も容易となる特色がある。Smalltalkにより複数のデータベースを統合する階層、複数の知識ベースを統合する階層およびデータと知識の協調階層を作る。この協調階層はデータベース階層や知識ベース階層と動的なリンクで結合されており、データや知識が必要に応じて取り込まれる。種々の問題はこの3種の階層の協調により扱うことができる。最後に簡単な実現例を示す。

Realization of Data-Knowledge Coordination Model by an Object-Oriented Based Method

Zhiyong PENG　Yahiko KAMBAYASHI

Faculty of Engineering Kyoto University
Sakyo, Kyoto 606, Japan

Data-knowledge coordination model[CHEN93] is suitable for integrating databases(DBs) and knowledge-bases (KBs) under distributed environment. In this paper, we will discuss an implementation method of such a model using an typical object-oriented programming environment - Smalltalk-80. Multiple DBs and KBs are integrated by Smalltalk-80. Interpretive nature of Smalltalk-80 is utilized to realize dynamic links among multiple DBs and KBs. Through these links, data and knowledge in multiply underlying DBs and KBs can be imported into Smalltalk-80 dynamically and selectively according to the problem to be solved. Related data and knowledge are integrated and coordinated through multiply hierarchies constructed in Smalltalk-80. Feasibility of this approach is shown by using a simple implementation example.

1 Introduction

Coordination of data and knowledge is very much important to realize advanced applications. Current systems, however, are usually designed for some predetermined objectives, and it is very difficult to modify objectives. In order to share a system by various applications, we need to develop a flexible data/knowledge system.

For this purpose, a new model called the data- knowledge coordination model[CHEN92] has been developed, which is suitable for integrating multiple DBs and KBs under distributed environment. That is, data are classified by their properties and knowledge by their subjects, and for combining the participating data objects and knowledge subjects in each localized problem domain, context modules based on the object-oriented data model are used. Instead of permitting arbitrary links among data and knowledge, the model permits dynamic links among three kinds of hierarchies; token object hierarchy for data integration, knowledge module hierarchy for subject knowledge integration, and context module hierarchy for data-knowledge coordination. Through such dynamic linking, data and knowledge can be properly tailored and coupled to fit different cooperating contexts. Moreover, the cooperation of data and knowledge are specified abstractly and handled separately from the independent data and knowledge management.

In this paper, an implementation method for such a model is discussed. We use an object-oriented programming environment - Smalltalk-80 [GOLD83a] [GOLD83b] to realize these hierarchies and links. More than one hierarchy can be generated by Smalltalk-80 by object reference and inheritance relationship.

We will use a client/server architecture to realize the model. In a client machine, Smalltalk-80 is used and each server realized on one or more DB and/or KB system. Between a client and a server, a specific protocol named as linkage protocol is defined for the link management and communication. After links are established, a bilateral pipe is built up between the client and each server. DBs and/or KBs become available in Smalltalk-80 through the pipe. For imported information specification, one class or class hierarchy is created in Smalltalk-80. The imported information is taken as the instances of these classes and handled like the conventional objects in Smalltalk-80.

Based on the above linking mechanism, data from multiple DBs are imported into Smalltalk-80 as follows. The classes for the imported data are regarded as **virtual classes**, of which instances are considered to be a view of the imported data. As a view, the imported data values are not replicated and they are manipulated in Smalltalk-80 through methods defined in the virtual classes. For identifying data, we introduce data identifiers which are different from object identifiers (if they exist)in each underlying DB. Knowledge from multiple KBs are also grouped in term of subjects in Smalltalk-80. For each subject knowledge, we define a **subject class** of which instance is regarded as an integrated view of the knowledge with the same subject from multiple KBs. Its instance variables as slots can be filled by the importation facts that are involved in the reasoning based on the subject knowledge, while methods as scripts are used for triggering the reasoning mechanism provided by the linked underlying KB. In this way, the imported data and knowledge are able to coordinate for problem solving. Data are assigned into appropriate slots in subject knowledge, and then reasoning mechanisms will be triggered according to the problem solving logic. The problem solving logic is determined by the application. For this reason, we introduce **context class** as problem solving logic specification, of which instances are used as context modules for data/knowledge coordination.

All of data, knowledge and context modules are represented and manipulated as objects in Smalltalk-80, and therefore they can be organized into many kinds of hierarchies suitable for application requirement. That is, using complex object and class hierarchy integrates data and knowledge from multiple DBs and KBs, and then coordinates the integrated data/knowledge for solving problem cooperatively.

2 Data-Knowledge Coordination Model

Data-knowledge coordination model couples data and knowledge dynamically and is suitable for integrating DBs and KBs under distributed environments. In order to reduce the object duplication and migration, the context module(CM) is introduced to realize independent management and dynamic coupling of data and knowledge under cooperation contexts.

There are the following three kinds of hierarchies, for data integration, knowledge integration and data/knowledge coordination, respectively, as shown in Figure 1[CHEN92].

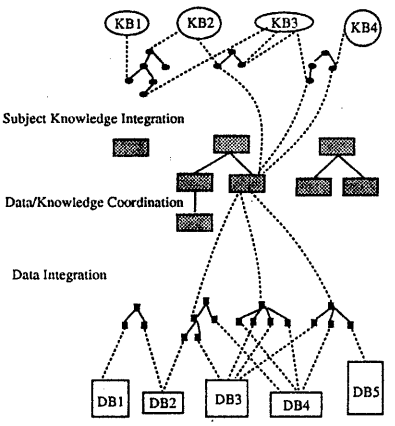


Figure 1. The data-knowledge coordination model

A. Data Object Hierarchy

Data objects are usually distributed in multiple data sources. A complex object may be composed progressively from more primitive ones. These component objects probably have different types and may be stored in multiple DBs. Data object hierarchy offers an integrated view to the complex data with conceptually related components distributed in multiple underlying DBs.

B. Knowledge Module Hierarchy

There are multiple knowledge sources at various levels of specificity and applicability. Knowledge can be fallen into the following three categories:

Database semantics expressing the schema and methods of a general class.

Subject knowledge scattered in various KBs and can be applied to the data objects from multiple participating classes.

Context knowledge specifying how to cooperate the data objects and subject knowledge involved in a specific problem domain or context.

Database semantics for each database are stored in database parts. Context knowledge are involved in context modules and become an integral part of the context

text
module hierarchy. Only subject knowledge are handled in the knowledge modules. In order to utilize the knowledge scattered in multiple underlying KBs effectively, it is important to group them.

As discussed above, knowledge are grouped in terms of subjects. That is, knowledge localized to a subject is grouped into a knowledge module(KM). A KM can be regarded as an integrated view of knowledge from multiple underlying KBs. They can be shared, reused and leveraged by organizing multiple hierarchies. There are three kinds of KM hierarchies: generalization, abstraction and composition.

C. Context Module Hierarchy

Context module (CM) provides a localized and dynamic data/knowledge coupling environment. A CM imports participating objects and subject knowledge from multiple DBs and KBs. It is defined in a specific problem domain, either application or management oriented, by linking multi-source participating data and knowledge dynamically and providing support to their cooperation. For example, a federate DB/KB server can be specified by a CM. Like the knowledge module hierarchies, CMs are also treated as objects and organized into composition, generalization and abstraction hierarchies so that they can be inherited, combined, shared and reused. Furthermore, maintaining multiple CM hierarchies can provide more feasibility for problem solving.

Major advantages of the data-knowledge coordination model are as follows:

Independence: Independent update of data and knowledge is possible.

Dynamic modification: Linkage between data and knowledge can be determined dynamically according to the application requirements.

Cooperation: It can realize a system consisting of multiple underlying DBs and KBs working cooperatively.

3 Implementation of the Model Using Smalltalk-80

Realization of coordination environment is the most important factor to develop application systems suitable for the data-knowledge coordination model. Such an environment should have dynamic linking capability for multiple underlying DBs and KBs, in which the imported data and knowledge can be integrated and coordinated through multiple hierarchies.

In order to examine feasibility of the model, we have developed a system utilizing Smalltalk-80. Smalltalk-80 is selected by the following reasons:

(1) Fully object-oriented language

Both data and knowledge can be represented and manipulated as objects[BAN87][FUJI90] [OTIS91].Smalltalk-80 system is fully object-oriented. Basic concepts such as objects, methods, classes, messages, instances are pervasive and applied uniformly in it. It enables developers to take advantage of object-oriented capabilities, such as inheritance, polymorphism and encapsulation. Therefore, it is a good integration environment for data and knowledge from multiple underlying DBs and KBs.

(2) Programming environment

As a programming environment, it has various powerful development tools, including browser, editor, debugger, decomplier etc., which are highly integrated with the smalltalk language so that user can code without in-depth knowledge of programming language. It also provides a reusable application framework and has incorporated mature class libraries consisting of 400 classes and 7000 methods. The application framework assists developers to generate application module to tie the interface and application logic together. It is completely object-oriented, which enables developers to reuse not only interface but the application logic that defines the application. Applications can be reused individually or as a part of another application. Reusable application framework facilitate easier application maintenance, enhance future development efforts, and ensure high-quality application. This framework is suitable for implementing context modules in data-knowledge coordination model.

(3) User friendly interface

Graphical Interface Builder provided by Smalltalk-80 enables developers to create GUIs quickly with a point-and-click "palette" and "canvas". A standard interface layout, or canvas, is automatically generated for the developers to begin work. Customized canvases may also be designed and stored within Smalltalk-80. Developers use the palette and choose from a wide array of layout tools to design the interface. These tools include a menu builder, icon painter and color tool. This facilities high-quality application development.

(4) Triggering mechanism to be used for client/server architecture.

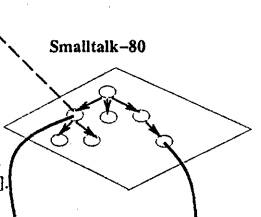
Smalltalk-80 has been equipped with the capability triggering external process. It has class **ExternalProcess** of which subclass, such as class **UnixProcess**, provides a reference to an external process at specified operating system environment (for example, a Unix OS). Its instances are created through primitives(interface is knowledgeable in host operating system) which "fork& exec" new processes as children of the running Smalltalk. After creating one, the main interesting thing to do with an external process is to communicate with it, by sending request, waiting for completion and observing its returned status. The method **pipeConnectionFor:** *aName* **arguments:** *startupArguments* **setProcessDescriptor:** *pdBlock*, along with its variants, is used for such a purpose, which starts a new job on the program called *aName*, passing arguments *startupArguments* and setting up pipes (one communication way between external processes) to the new job. Such an open structure allows developers to combine dynamically the power of multiple underlying DBs and KBs with object-oriented programming technology for client/server applications.

Using the capabilities shown above, multiple underlying DBs and/ or KBs can be mediated easily by linking them with Smalltalk-80 system based on a client/server architecture. Smalltalk-80 is taken as a client and one server is designed for one or more DBs and/or KBs. Between a client and a server, a specific protocol named as linkage protocol is defined for link management and communication. Linkage protocol is bilateral and determined by application characteristics. At the client side, a sole class or class hierarchy is defined, which can be regarded as view of the linked DBs and/or KBs. As instances of such a class, data or knowledge in the linked DB or KB become available in the Smalltalk-80. The methods in the class, which can be invoked similar to conventional Smalltalk methods, primarily provide link management, data/knowledge query, update and manipulation. On the other hand, each server is passive and triggered by invoking the methods defined in the above class(es). It primarily provides information service for an application in Smalltalk-80.

[Example 1] Suppose an application in Smalltalk-80 would ask some ONTOS database tutorialDB whether there exists some data in it. The data is stored as an instance of the class Thing in tutorialDB. The links for such requirement is created as shown in Figure 2. First, a class ViewForDatabase in Smalltalk-80 is created. In this class, three class variables named Connection, Proc and Cmd are defined for storing the argument information necessary for controlling the server. In addition,

an instance variable dataID is defined for identifying the imported data from tutorialDB. The methods, including openLink, fetch, put:, closeLink and query:, are implemented in Smalltalk language. A specific server is developed in C++ based on tutorialDB[ONTOS], which provides the requested information service. It is triggered by the method openLink, then peers query requests from Smalltalk-80, queries data in tutorialDB accordingly; it returns result to Smalltalk-80; finally, it is terminated by the method closeLink.

```
Object  subclass:  #ViewForDatabase
        class variable name:  'Connection Proc Cmd'
        instance variable name:  'dataID'
        pool dictionary:
        category: 'Ontos'
        class methods:
        openLink
        Connection:= UnixProcess pipeConnectionFor: 'rsh'
                                  arguments:#('flute' 'askOntos')
                                  setProcessDescriptor:[ :pd | Proc:= pd
        Cmd:= Connection readAppendStream.
        fetch
        ^Cmd nextHunk.
        put: aString
        Cmd nextPutAll: aString; cr; commit.
        closeLink
        self put:'stop'.
        Cmd close.
        Proc release.
        query: aThing
        | reply |
        self openLink.
        self put:aThing.
        reply:= self fetch.
        (reply='OK') ifTrue:[ ^true ]
                     ifFalse:[ ^false ].
```




```
#include <stream.h>
#include <stdio.h>
#include <string.h>
#include <Database.h>
#include "Thing.h"

main( )   // Server:  askOntos.C

{char request[20];
 OC-open("tutorialDB"); // Open the logical database tutorialDB
 OC-transactionStart(); //Start a transaction
 cin>>request;
 while(strcmp(request, "stop")!=0)
 { Thing* aThing = (Thing* ) OC-lookup(request); //Query Ontos
   if(aThing != Null) cout<<"ok" else cout<<"no"; // Answer query result
   cin>>request;
   OC-transactionCommit(); //Commit the transaction
   OC-close(); //Close the database
 }
```

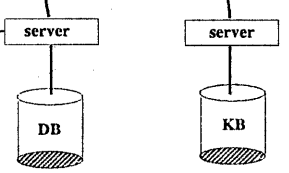


Figure 2. Dynamic link between Smalltalk-80 and the underlying DB and KB

The linkage protocol is also responsible for the format conversion between Smalltalk-80 and the DBs and/or KBs. The above example is only concerned with string format conversion between Smalltalk and C++, which has already been provided by Smalltalk-80. If the more complex format conversion is required, the linkage protocol should be extended with the more complex conversion capabilities based on the owned string format conversion. For example, floating numbers possibly in different precision can be communicated between two systems by converting them into strings at first, then transferring them as strings, and finally compiling them into the available format.

The links between Smalltalk-80 and multiple underlying DBs as well as KBs are depending on application requirements as shown in Figure 3. For different applications, we may have different linking combinations. Through these links, data and knowledge can be employed in Smalltalk-80 selectively. The related data and knowledge from multiple underlying DBs and KBs will appear in the same form as the conventional objects in the Smalltalk-80 while their syntax mismatches are filtered by the corresponding linkage protocols. Furthermore, their semantic heterogeneity will be tuned through relationships and constraints defined in the corresponding classes or the multiple hierarchies constructed in the Smalltalk-80. These integrated data and knowledge are coordinated dynamically in the context module objects.

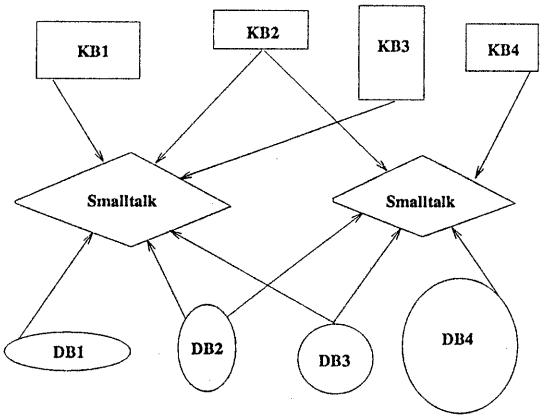


Figure 3. Realization of the data-knowledge coordination model

4 Integration of Data from Multiple Underlying DBs

As shown in Section 3, data distributed in multiple underlying DBs can be imported into Smalltalk-80 by building one-one correspondence between one data item in DB and an object in Smalltalk-80, where data are managed by the underlying DBs and manipulated as objects in Smalltalk-80. An object is regarded as a view of its corresponding data whose actual values are stored in the underlying DB and data are manipulated through methods that are implemented by the linkage protocol. Thus, the actual data can be updated independently within the underlying DB, and update effect can be reflected into Smalltalk-80 immediately through methods.

For such a purpose, Smalltalk-80 provides an unique identifier for each importation data. The data identifier provided by a server can be different from the object identifier provided by Smalltalk-80. The former is used by the server for viewing data in DB while the later for manipulating the corresponding data through methods defined in Smalltalk-80.

Although the identifying mechanisms are realized independently, the data with the same data identifiers from different underlying DBs can be differentiated because they are assigned with different object identifiers in Smalltalk-80. For object-oriented database(OODB), its identifying mechanism can be employed directly by the server. However, the other kind of databases such as relational database (RDB) should be provided with some mechanism through which the imported data from it can be assigned an unique data identifier. Such a mechanism is developed as a part of server and its realization completely depends on the underlying DB.

Using data identifying mechanism, an underlying DB is linked to Smalltalk-80 by defining a class or class hierarchy in Smalltalk-80 and implementing a server based on the underlying DB. The class(es) is named as virtual class(es) and the imported data are represented as its instances(virtual objects). In the virtual class, only one instance variable is defined for storing the imported data identifier. The stored data identifier is provided by the identifying mechanism and with it the actual data within the underlying DB can be manipulated through the methods defined in the virtual class. The one-one correspondence between the imported data and virtual object is shown in Figure 4.

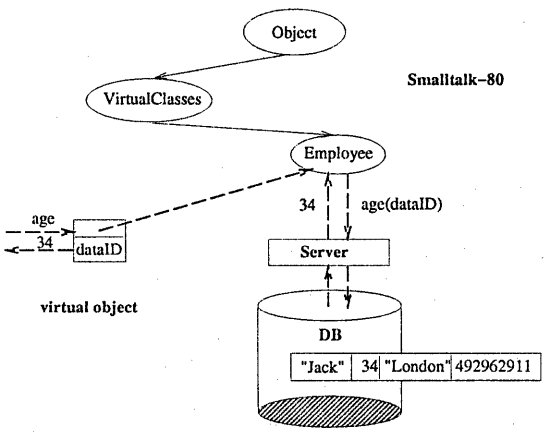


Figure 4. Correspondence between data in DB and object in Smalltalk-80

Data identifier is completely localized with the underlying DB. When its corresponding data is required by the application, it will be imported into Smalltalk-80 and correlated an object of the class designated by the application. An instance of that class is created and the data identifier is stored in the instance variable of that newly created object. Thus, the data identifier and object identifier are correlated according to the importation request. When the data has been handled and become unnecessary, the correlated object will expire and the data identifier will discorrelated with the expired object identifier.

For each class, the data identifier can correlated only one object. In order to avoid duplicating the same imported data with different virtual objects of the same class, a virtual object dictionary is defined for each virtual class. Depending on such a dictionary, the data will be imported in the following way. The imported data is first assigned an unique identifier by server; and then the dictionary is searched to see whether there has been a virtual object with the same data identifier. If there exists, it shows that this data has already been imported as an instance of that class. The found object should be picked up and manipulated as the view of the imported data. Otherwise, a new virtual object will be created and put into the dictionary. Thus, the one-one correspondence between the imported data and virtual object of the same class can be guaranteed in Smalltalk-80.

Because the same data can be viewed differently by different applications, namely as an object of different classes. The above importation method only limit the same data to correlate with just one instance of the same class rather than multiple instances of different classes. Therefore, data identifier can be correlated with more than one instances of different classes. Thus the same data can be used by different applications as instances of different classes simultaneously.

In order to manipulate the imported data, some access methods must be implemented through the linkage protocol. They are defined in the virtual class and invoked similar to the conventional methods in Smalltalk-80, namely through passing message to object. As a simple example, suppose an employee record (name:Jack age:34 address:London telephone:492962911)in some underlying DB might be manipulated by some application. In the above process, the data would be imported into Smalltalk-80 as shown in the Figure 4. In order to access its attributes, a pair of methods have been defined for each one, such as age and age:, which can be viewed as reading and writing actions within the underlying DB. Issuing age means to obtain age attribute value 34 from the underlying DB while issuing age: to update it. These actions are included in the transaction triggered by the server so that they can't violate with the other processing processes depending on local concurrency control mechanism provided by the underlying DB.

All the involved data from multiply underlying DBs are imported as virtual objects in the above way. Therefore, utilizing Smalltalk-80, these data can be extended and integrated in the following three basic methods:

1) Extending data with additional attribute

The attributes of data from an underlying DB may be not sufficient for the application requirement. Consider the above example again. Suppose the application would give all involved employee some privilege for using resources in company. For this situation, we can create a new class PriviledgedEmployee as a subclass of virtual class Employee(Figure 5), in which an additional instance variable is defined to hold this new attribute value. Such an attribute is only application-oriented and without any effect on the actual data in the underlying DB. The value stored in the additional instance variable may be an object of a basic class or other virtual class.

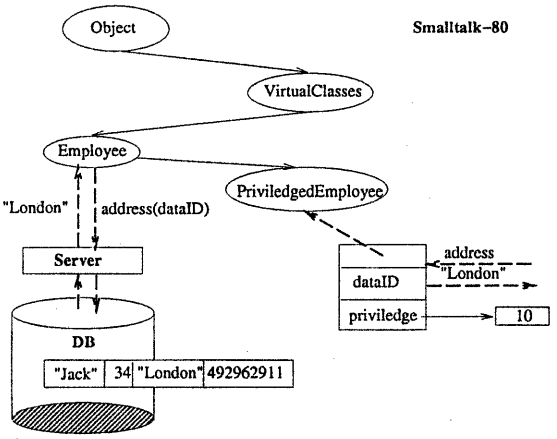


Figure 5. Extend data with additional attribute

2) Combining data from multiple underlying DBs

The attributes of the same entity may be dispersed in different underlying DBs. For example, a school has components for "school map" and "school description" stored in different underlying DBs. The data in these two underlying DBs can be imported through two virtual classes defined in Smalltalk-80. In order to employ the

imported data from different underlying DBs as an integrated data, we can create a new class with two instance variables which reference the component data(Figure 6). In this combination class, methods can be used to guarantee the global consistency among local data, by propagating the effect of data modification.

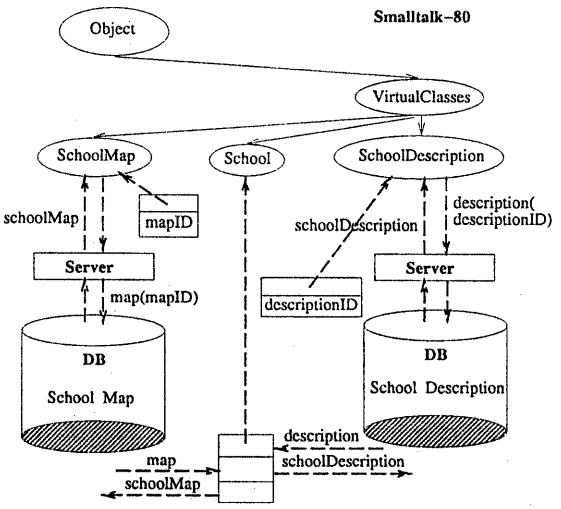


Figure 6. Composition data from different DBs

3) Building hierarchy for data from multiple underlying DBs

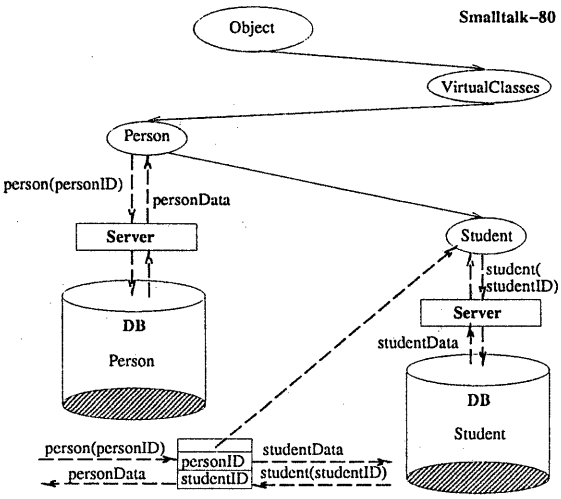


Figure 7. Data hierarchy from different DBs

Some relationship such as the generalization - specialization may exist between data from multiple underlying DBs. For example, the general personal data(name, age, birth place, etc.) can be obtained from DB in public office while the special data as student(name, major, course, score, etc.) from DB in his studying school. If some application requires both of these data, they need to be imported into Smalltalk-80 simultaneously. Obviously, student can be regarded as a person with some special status. Therefore, the corresponding virtual classes should be organized into super-sub class hierarchy in Smalltalk-80, namely Person as superclass and Student as subclass(Figure 7). Unlike the usual virtual class, Student has two data identifiers, one is its own and the other inherited from Person. Of course, these two data identifiers should be kept consistent through the methods defined in Person and Student.

The above basic methods can be combined and used repeatedly so that data from multiple underlying DBs can be extended and integrated at higher level. These data tailoring are completely application-oriented.

5 Integration of Knowledge from Multiple Underlying KBs

Integrating knowledge from multiple underlying KBs by Smalltalk-80 is realized by introducing subject class to define an integrated view of the knowledge. A subject class consists of slots and scripts, where slots as instance variables are used for representing importation facts provided by the underlying DBs and scripts as methods for triggering reasoning based on knowledge within the underlying KBs. An underlying KB consists of one or more subjects. On the other hand, knowledge with the same subject may be scattered in multiple KBs. For each KB, one or more subject classes are defined in Smalltalk-80 and they are linked in the way discussed in Section 3. In Smalltalk-80, the subject classes from different KBs with the same subject will be integrated by means of complex objects. In such a way, knowledge within the linked underlying KBs can be employed by invoking script defined in subject classes. The script will trigger operations on the corresponding underlying KB as if some user is utilizing the KB. Only difference is that importation facts are provided by slots through linkage protocol rather than by a user.

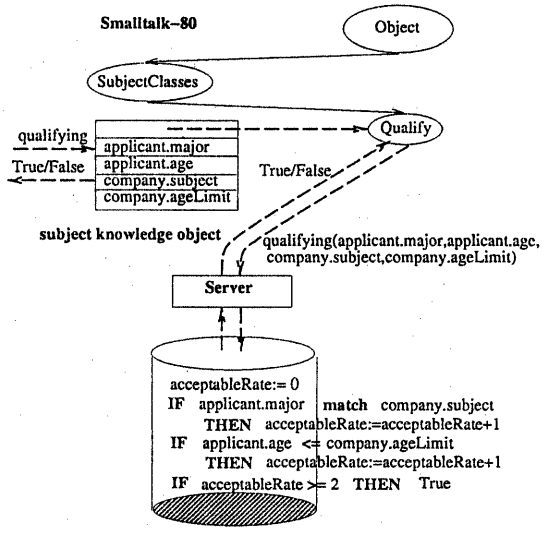


Figure 8. Correspondence between knowledge in KB and object in Smalltalk-80

Subject class is just subject knowledge specification and should be instantiated by creating an instance and

assigning actual data into its slots when the subject knowledge is involved in some application.

[Example 2] Consider the example shown in Figure 8. Suppose an underlying KB, which justifying an applicant **whether** can be accepted by some designated company, has been linked to Smalltalk-80 through subject class Qualify. Since using the underlying KB requires providing factual data(applicant's major, age and company's subject, age limitation), four slots as instance variables (applicant.major, applicant.age, company. subject, company.ageLimit) are defined in Qualify for storing these importation facts. In addition, a script to qualify methods is implemented to trigger the reasoning based on rules in the underlying KB. When a subject knowledge object instantiated by Qualify has received the message qualifying from some application in Smalltalk-80, the corresponding method in Qualify will be invoked. The method picks up the factual data stored in slots by the application and interact with the linked underlying KB's server through the linkage protocol. The reasoning result will be replied to the application through the link.

The above way might make the application employ knowledge within the underlying KB as if these knowledge would be conventional objects in Smalltalk-80. Like the imported data, these subject knowledge objects can be extended and integrated in the following basic methods:

1) Aggregating subject knowledge from multiple underlying KBs

Like complex data, complex subject knowledge can also be aggregated from simpler ones. This can be done by creating a new subject class with several instance variables which reference the component subject knowledge. In order to encapsulate component subject knowledge, some necessary methods are defined in the created subject class as uniform accessing protocol for all slots and scripts in component subject class. The example in the next section will give a detail illustration about such an aggregating way.

2) Classifying subject knowledge into generalization hierarchy

Subject knowledge from multiple underlying KBs can be classified according to specialization -generalization relationship among them. The subject knowledge about driving a car is viewed as the specialized subject knowledge about driving a vehicle. If both of them are required to be employed in Smalltalk-80, the super-subclass inheriting mechanism provided by Smalltalk-80 enables these two subject classes to be organized into generalization hierarchy, namely subject class CarDrive being defined as a subclass of VehicleDrive. Depending on the hierarchy, only slots and scripts special for CarDrive are implemented in it and others are shared by inheriting them from VehicleDrive.

3) Classifying subject knowledge into abstraction hierarchy

Subject knowledge abstraction is an extension to generalization. The abstraction hierarchies are organized in the same way as generalization hierarchies. However, in the abstraction hierarchy, the scripts in the superclass are based on the more abstract importation facts and knowledge than those in the subclass. For these scripts, instead of inheriting from the superclass, we have to reimplement them in the subclass. For example, The script seeDoctor based on the rule $sick(X) \rightarrow visit(X,Y) \& doctor(Y)$ can be viewed as the abstraction of the script with the same name seeDoctor which is based on the rules $injury(X) \rightarrow visit(X,Y) \& surgeon(Y)$ and $flu(X) \rightarrow visit(X,Y) \& physician(Y)$. In the subclass, the script would not be shared with the superclass and while it be reimplemented with the latter concrete rules. Smalltalk-80's dynamic binding mechanism might ease the realization of such a hierarchy. In Smalltalk-80, duplicating method definitions with the same message format is allowed, and thus the script for the same objective can be implemented at different abstraction levels along the path from superclass to subclass.

In general, the subject knowledge defined in the superclass can be applied at higher levels and in wider ranges than in the subclass.

6 Data-Knowledge Coordination

As discussed in the previous sections, data from multiple underlying DBs are imported to Smalltalk-80 as the instances of virtual classes and integrated by virtual class hierarchies. Knowledge in multiple underlying KBs, on the other hand, can be employed by sending script message to the instances of subject classes defined in Smalltalk-80 and the grouped subject knowledge can be integrated into subject class hierarchies.

In order to coordinate these integrated data and knowledge, context classes are used in Smalltalk-80. In context classes, instance variables are defined to reference the related data and subject knowledge, and methods are used to implement dynamic logical relationship between data and subject knowledge. Its instance is referred to as context module and realizes data-knowledge coordination, as shown in Figure 9.

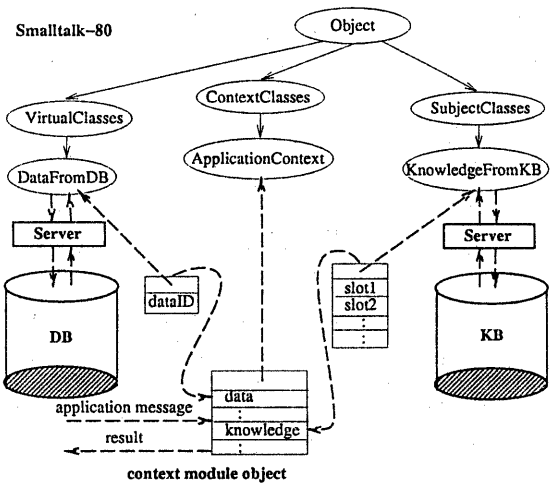


Figure 9. Data and knowledge coordination based on context module object

In Smalltalk-80, multiple context classes might be created for solving various problems. The same data and subject knowledge may be involved in multiple context modules. In addition, the involved actual data and subject knowledge in context module frequently change along with problem solving progress.

The methods in context class primarily realize context knowledge which are controlling importation and coordination of data and subject knowledge. Context knowledge is application oriented and defined in a specific problem domain. For a complex problem, single context module may be not sufficient and a set of context modules should be maintained. Like data and subject knowledge, these context modules can also be organized into multiple hierarchies in the following methods:

1) Composing CMs for solving the problem overriding multiple domains

Assuming that multiple context modules have been designed in Smalltalk-80 for various problem domains.

If a new problem overriding these problem domains occurs, it can be solved by composing the existing context modules rather than designing one from scratch. A composition context class is created with several instance variables. Among these instance variables, some are used to reference the existing component context modules and the others for additional necessary data and subject knowledge which are not involved in the component context modules. Methods in the composition context class primarily realize knowledge about coordination among the existing component CMs, additional data and knowledge, problem solving logical overriding multiple domains.

2) Specializing CM for solving the problem in the more special domain

Solving special problem requires special additional data and knowledge. It can be done by reusing the corresponding general CM, namely creating a subclass of the general context class. The subclass inherits all context knowledge from its superclass, including general data and knowledge importation, their coordination, and general problem solving logic. In the subclass, only these are defined, including its own instance variables for referencing the involved special data and subject knowledge, and methods for coordination not only between special data and subject knowledge but also between special and general ones, especial for realizing the more special problem solving logic.

3) Abstracting CM for handling problem at a higher level and in a wider range

Unlike generalization, abstracting CM provides more abstract data and knowledge coordination. In Smalltalk-80, abstract context class might be created as superclass and concrete one as subclass. In the subclass, instead of inheriting from superclass, the definition for the related data and subject knowledge as well as their coordination are reimplemented in the more concrete form. The instance variables defined in the superclass are reused in subclass but they reference data and subject knowledge with the more concrete representation. Controlling their importation depends on the problem being abstract or concrete. In addition, almost all methods for the same objective are reimplemented with the same invoking way but applicable at different level and in different range. This enables some unsolvable problems to become solvable at a higher level and in a wider range depending on the dynamic binding mechanism in Smalltalk-80.

The above three basic methods can be used repeatedly and combined, and multiple complex CM hierarchies can be realized. Through their cooperating, various complex and even some unanticipated problems can be solved in Smalltalk-80 which mediates multiple underlying DBs and KBs dynamically.

7 An Example

In this section, a simple implementation example is shown to illustrate how to develop an application system suitable for data-knowledge coordination model. This application system provides student with service for finding a suitable company. It is simple but requires multiple underlying DBs and KBs cooperation. In general, finding company for a student should first know data of both student and company. With these data, whether a student can be accepted by some designated company needs qualifying knowledge. If the student can be qualified as an employee by that company, he might ask the company about his salary and then the company requires salary knowledge to answer him. Finally, the student makes a decision by comparing salary provided by all acceptable companies, and of course, the highest is his choice.

As will be seen later, our proposed way is suitable for implementing such an application system with increased feasibility. These DBs and KBs are linked to Smalltalk-80, related data and knowledge are integrated for coordination by constructing necessary hierarchies and these hierarchies may be extended for some unanticipated situation. This is shown in Figure 10 and the detail explanation will be given as follow. This simple application system based on Smalltalk-80 has been implemented by mediating three ONTOS databases and two simulated production rule based Knowledge bases.

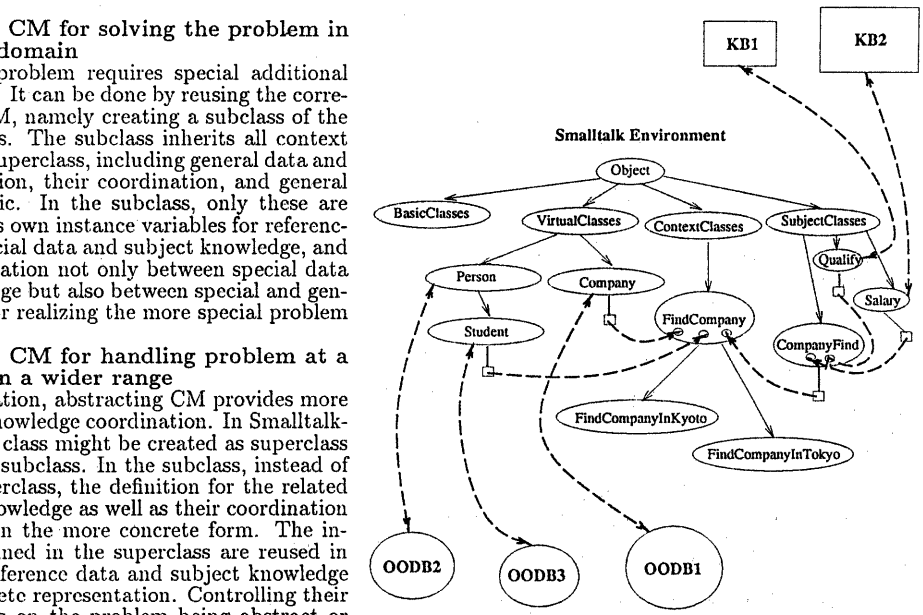


Figure 10. An example for data–knowledge coordination

Data integration

The data necessary for the application system are stored in three ONTOS databases OODB1, OODB2 and OODB3. The company data within OODB1 are composed of attributes (string subject; int ageLimit; basicSalary; string address;). In order to make them available in Smalltalk-80, a virtual class Company is created and its instance is regarded as the view of the imported company data. The data identifier stored in the instance variable CompanyID is provided by OODB1 based on the ONTOS's identifying mechanism[ONTOS].

On the other hand, a corresponding sever based on OODB1 is developed in C++ to perform actual data manipulation, which is managed and employed by the methods defined in the virtual class Company. These methods are realized Through the linkage protocol for link management(openLink, closeLink), communication (fetch, put:), data importation(fetchAll, fetchWithCondition:), attribute reading(subject, ageLimit, basicSalary, address) and attribute writing(subject:, ageLimit:, basicSalary:, address:).

In addition, a dictionary VOdictionary is defined as one of the Company's class variables. The dictionary is stored with all objects corresponding to the imported company data. If a company data need be manipulated by the application, its data identifier will be provided by the server. And then, the VOdictionary is searched in term of this data identifier. If there is no such object in

it, the virtual class Company will be instantiated by creating an object as the data's view and the object should be put into the VOdictionary.

Sending attribute reading message to the object can obtain its corresponding attribute value. Sending attribute writing message can update its corresponding attribute value. These operations are included in a transaction triggered by the server and therefore can't cause any inconsistency state in OODB1. Besides, data identifier always keeps unchanged so data in OODB can be updated independently and also it can be manipulated in Smalltalk-80 simultaneously.

The data within OODB2 and OODB3 are composed of attributes (string name; int age) and (string name; string major; string degree), respectively. They are linked to Smalltalk-80 in the same way as OODB1. Virtual classes, which are Person with instance variable personID and Student with instance variable studentID, are created in Smalltalk-80. Corresponding to them, two servers are implemented based on OODB2 and OODB3. Obviously, a student can be regarded as a person with some special status. For such a situation, the data from OODB2 and OODB3 should be integrated by organizing them into a generalization hierarchy, namely taking Student as a subclass of Person. The integrated data view is an instance of Student which has its owned data identifier studentID as the view of data from OODB3 and inherits another data identifier personID from Person as the view of data from OODB2(Figure 7).

In order to overcome semantic heterogeneity of data from OODB2 and OODB3, a dictionary is defined in Student to store the equivalence relationship between name attribute of data from OODB2 and OODB3. In addition, some necessary methods are implemented to guarantee the integrated data semantic consistency. In this application, suppose their semantic consistency constraint is that only data with the same or equivalent name from OODB2 and OODB3 can be integrated. For example, when a student data need be imported from OODB3, a method is needed to see whether there is a corresponding person data with the same or equivalent name in OODB2. If there exists, the corresponding person data should also be imported together with the student data and then they are integrated as an instance of the class Student. Otherwise, the required student data is thought to be incomplete and this importation request should be rejected.

Knowledge integration

The knowledge within KB1 are employed to justify whether a student can be accepted by some designated company. The judgment requires importation fact about the major and age of applicant as well as the subject and age limitation of company. The rules in KB1 show that only if the applicant's major is consistent with the company's subject and his age is not greater than the company's age limitation, he can be accepted. The link way between Smalltalk-80 and KB1 has been illustrated in [**Example 2**]. In the same way, the knowledge within KB2 can be employed in Smalltalk-80 by creating another subject class named as Salary and linking KB2 to Smalltalk-80 through the subject class. Different from KB1, the knowledge within KB2 are primarily for answering student about his obtainable salary.

In the subject class Salary, three instance variables named as applicant.age, applicant.degree and company.basicSalary are defined as slots since the reasoning based on the rules within KB2 requires these importation facts, which include the student's age, degree as well as the company's basic salary. In addition, a method named as salary is implemented as the script for triggering reasoning based on the rules within KB2.

Both of these two subject knowledge can be regarded as the component subject knowledge of the subject knowledge about finding company in our example. For this reason, they should be composed into a composition subject knowledge, which is defined by the subject class CompanyFind. In the composition subject class CompanyFind, two instance variables are defined for referencing the component subject knowledge Qualify and Salary.

In order to access slots defined by the component subject class, a pair of methods are implemented for each component slot. As an example, the method applicant.age is used to read the corresponding slot value from the component subject knowledge. Applicant.age: newValue is to update the slot value in the component subject knowledge. Since the slot applicant.age is duplicated in Qualify and Salary, the update method should change the duplicated component slots simultaneously. Besides, the scripts in Qualify and Salary are abstracted by the methods in CompanyFind, which delegate the corresponding scripts in Qualify and Salary to trigger the reasoning based on rules in KB1 and KB2. In this way, CompanyFind encapsulates Qualify and Salary so that the knowledge within KB1 and KB2 can be employed in Smalltalk-80 as an integrated view.

Data and knowledge coordination

Now the related data and knowledge from multiple underlying DBs and KBs have been integrated based on Smalltalk-80. It is time to design context module to coordinate them for solving the problem proposed by our example. For this purpose, a context class FindingCompany is created in Smalltalk-80 as the context module specification. It has three instance variables which reference the related data and subject knowledge.

In order to coordinate these data and subject knowledge, the context knowledge are implemented by the methods defined in FindingCompany. Some of these methods are for assigning data to the appropriate slots in subject knowledge. For example, the method for instantiating FindingCompany is used to import related subject knowledge and data as instances of CompanyFind, Student and Company, and couple them accordingly, such as assigning the age of the student into the applicant age slot of the subject knowledge about finding company.

The others are primarily for realizing problem solving logic. In our example, the application system first let student input his name, then it queries OODB2 and OODB3 in term of his name. If without any data about him in OODB2 and OODB3, the system prompts him correct input or allows him to give up continual try. If not so, the found data is imported into the working context module. And then, the context module imports company data one by one from OODB1. For each company, the reasoning based on rules in KB1 is first triggered to justify whether he can be accepted. If he is returned with OK, then the reasoning based on rules in KB2 should be triggered to ask his obtainable salary from that company. Finally, the reasoning result is stored for the comparison afterwards. After all of companies have been enumerated in the above way, the final result is shown to the student. If more than two companies can accept him, the one that can provide the highest salary is suggested by the application system.

The context class FindingCompany has achieved the objective proposed by our example. However, student may put forward some unanticipated requirements such as wish to find a suitable company in Kyoto. It is obvious that this problem can be regarded as more concrete case of the above one. Therefore, it can be solved by creating a new context class FindingCompanyInKyoto as subclass of FindingCompany. In FindingCompanyInKyoto, only method about importing company data into working context module need be reimplemented so that it only imports the data about companies in Kyoto.

Using dynamic binding mechanism in Smalltalk-80, once the procedure finding company in Kyoto fails, the

finding procedure can be triggered at the more abstract level namely FindingCompany. Thus, even if the student can't find a suitable company in Kyoto, he may be given a suggestion that some company in other city is suitable for him.

If the student wish to find a company in Tokyo, he can be satisfied by creating another new context class FindingCompanyInTokyo in the same way as FindingCompanyInKyoto. This shows the feasibility of our proposed realization way.

8 Concluding Remarks

In this paper, we have presented an implementation way of the data-knowledge coordination model. It utilizes an object-oriented programming environment-Smalltalk-80 as dynamic coordination environment of data and knowledge from multiple underlying DBs and KBs, and emphasizes feasibility of the developed application system. In this way, multiple underlying DBs and KBs are linked to Smalltalk-80 according to application requirement. Data and knowledge are imported into and employed in Smalltalk-80 in the form of data and knowledge view. They are extended, integrated and coordinated through constructing multiple hierarchies by using object reference and inheritance relationship in Smalltalk-80. A simple application system has been implemented in order to illustrate the feasibility of this approach.

Recently, coupling data and knowledge has received much attention. Several approaches have been proposed in this research field[CHEN91a][CHEN91b] [PEDE91] [WEIS91][ZOBA87]. Compared with them, utilizing Small talk-80 as data-knowledge coordination environment can provide application with the more feasibility. Especially, identifying data with two different identifiers(data and object identifiers) will loose the tie (A data belongs to only one class), which is one of the main causes of view, object migration and schema modification problems. In addition, using virtual class and subject class as data and knowledge view in Smalltalk-80 facilities independent management and dynamic link of data and knowledge, which are discounted by the previous methods.

Smalltalk-80 has been employed as an interface of an object-oriented database GemStone [BUTT91]. Such an integration is fixed and can't support data integration from multiple underlying databases. Integrating data from multiple database with virtual view has been discussed in [MOTR87], but it is only suitable for data with the same functional model. Our approach is not only for heterogeneous data integration but also for data-knowledge coordination from multiple underlying DBs and KBs.

Since only single inheritance can be supported by Smalltalk-80, enhancing it with multiple inheritance and the efficiency of this way are our future research topics.

References

[BAN87] J. Banerjee et al, Data model Issues for Object-Oriented Applications. ACM Transaction on Office Information Systems, No.1, 3-26, Jan. 1987.

[BUTT91] Paul Butterworth, Allen Otis, Jacob Stein, The GemStone Object Database Management System. Communications of the ACM, Vol.34 No.10, Oct. 1991.

[CHEN91a] Qiming Chen, Yahiko Kambayashi, Nested Relation Based Database Knowledge Representation. Proc. of the 1991 ACM SIGMOD International Conference on Management of Data Denver, May 1991.

[CHEN91b] Qiming Chen, Yahiko Kambayashi, Unifying Data Grouping and Knowledge Grouping through Nested Relational Based Knowledge Representation. Proc. of the IEEE Fifth International Computer Software & Application Conference Tokyo, Japan, September 1991.

[CHEN92] Qiming Chen, Yahiko Kambayashi, Coordination of Data and Knowledge Base Systems under Distributed Environment, IFIP, DS-5 Semantics of Interoperable Database Systems. Lorne, Victoria, Australia, November 16-20 1992.

[FROS86] Richard Frost, Introduction to Knowledge Base Systems. Collins Professional and Technical Books 1986.

[FUJI90] Shigeru Fujimura, Shoji Tomita, Noboru Iima and Akira Suzuki, AUK: Shell for Building Intelligent System Based on Object-oriented Knowledge Representation auk. Transaction of Information Processing Society in Japan, Vol.31 No.1 Jan. 1990.

[GOLD83a] Goldberg, Addele and David Robson, Smalltalk-80: The Language and Implementation, Addison-Wesley, 1983.

[GOLD83b] Goldberg, Addele Smalltalk-80: The Interactive Programming Environment, Addison-Wesley, 1983.

[MOTR87] Amihai Motro, Superviews: Virtual Integration of Multiple Databases. IEEE Transaction on Software Engineering, Vol. SE-13, No. 7, July 1987.

[ONTOS] ONTOS Developers Guide.

[OTIS91] Allen Otis, A Reference Model for Object Data Management. Computer Standards & Interfaces 13 (1991) 19-32 North-Holland.

[PEDE91] Pedersen, Cet al, Data and Knowledge Bases as an Integral Parts of a Distributed Object Infrastructure. Proc. of IEEE International Workshop on Interoperability of Multidatabase Systems, 1991.

[ULLM88] J. D. Ullman, Principle of Databases and Knowledge-Based Systems. Vol. 1, Computer Science Press, 1988.

[WEIS91] Weishar, D. and L. Kerschberg, An Intelligent Heterogeneous Autonomous Database Architecture for Semantic Heterogeneity Support. Proc. of IEEE International Workshop on Interoperability of Multidatabase Systems, 1991.

[ZOBA87] A. AL-Zobaidie J.B. Grimson, Expert Systems and Database Systems: how can they serve each other?. Expert System, February 1987. Vol. 4, No.1.