データベースシステム 109−45
(1996. 7. 26)

永続オブジェクト指向スクリプト言語を用いた移動計算機向け情報検索機構の実現

Wisut Sae-Tung　大森 匡　星 守
電気通信大学 大学院 情報システム学研究科
情報システム設計学専攻 データベース学講座

移動計算機の普及に伴いユーザはこれを持ち歩き、ネットワーク上で様々な情報を提供するデータベース、いわゆる情報資源から情報検索する機会が非常に増えている。しかし、これらの情報資源はそれぞれが自律的に設計／開発されており、統一的な検索インタフェスを提供していない。このような環境では、ユーザが同時に複数の情報資源を利用するのは難しい。そこで本稿では、移動計算機ユーザに対してこのような情報資源での自由な情報検索を提供するために、永続オブジェクト指向スクリプト言語 Persistent Perl を提案し、これを用いて移動計算機向けの情報検索機構を実現する。

An Information Retrieval Architecture for Mobile Computers Based on a Persistent Script Language

Wisut Sae-Tung　Ohmori Tadashi　Hoshi Mamoru

The University Of Electro-Communications
Graduate School Of Information Systems

Because of the outstanding of hardware technology, a size of computers becomes smaller so that users can take their mobile computers, move from one place to another place within networks and query information from the information resources, database server that provide different kind of multi-media information. Unfortunately, the information resources connecting to the network are developed by different people, in different languages and at different time. Therefore, they do not provide the integrated interface. This causes the difficulties in manipulating information in heterogeneous information resources. To solve this problem, we have designed and developed an environment and a set of tools for mobile computer users to integrate information on the network. These help users easily and freely manipulate information on the heterogeneous information resources.

1 Motivation

In mobile computing environment, users carry their mobile computers, travel around the network and may require to access information from unknown information resources[1]. However, the information stored at such information resources has been developed in diverse languages on a wide varity of hardware and software platforms, for example; relational databases, document retrieval systems, file systems, WWW servers, Java-application servers and so on[2]. Consequently, they often provide different data models and access mechanisms. on this variety, the users can not freely access information on information resources in an integrated manner[2,5]. Therefore, the problem of an integration has become much more challenging. This situation can be shown in figure 1. All the information resources do not provide global and integrated interfaces. Moreover, the access methods that are provided by these information resources do not use the same algorithms as the users want. Therefore, they must use these access methods to get information back to their mobile computers and browse them one by one. This method takes time and consumes network resources.

Recently, remote programming has become a focus of attentions to solve the above problem[4]. Remote programming has flexibility and efficiency by performing much of work locally on information resources and returning results to users only the relevant information. This method thus provides a better solution than a traditional distributed client/server model for the system with a low bandwidth or unreliable network and limited resource capacities in mobile devices.

To solve the above problem, we use the idea of remote programming. We design and implement a set of tools that facilitate users to freely access data on spontaneous information resources in the integrated manner[8,9]. In this paper, after introducing an overview of our system architecture in section 2, we describe problems and necessary functions in script generating in section 3. In section 4, we describe how to solve the problems shown in section 3. In section 4, we describe related work and the philosophy of our approach. Finally, we conclude and discuss our ongoing work.

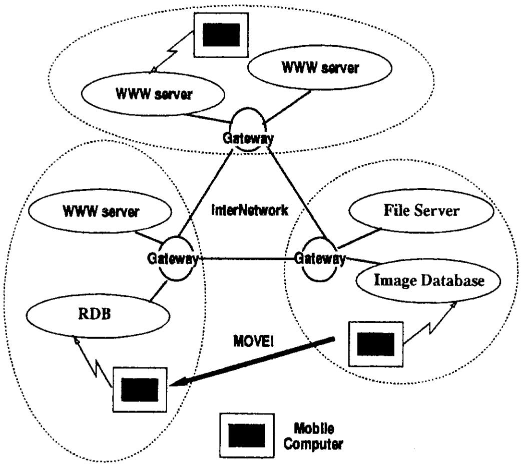

Figure 1: Information Resource and Network structure

2 System Architecture

To understand our system more clearly, we will start from the following example with the most basic processing. It is shown in figure 2. This figure shows the information-retrieval processing between a mobile computer and an information resource. Suppose that the mobile computer user comes across an unknown information resource when he move around the network. In this case, the information-retrieval processing is performed in the following steps:

Step 1 The user does not know data schemas and how to access information in the information resource. Therefore, he sends a request for the class definitions of the provided information to the information resource by an e-mail.

Step 2 Corresponding to the user's request, the wrapper function on the information resource searches for the class definitions and sends them back to the user.

Step 3 The user registers the class definitions sent from the information resource into his data dictionary and then use them to build a filtering script in his own style. This filtering script will be sent to the information resource by e-mail again.

Step 4 The wrapper function on the information resource executes the incoming mail, and then sends the result of the execution back to the user.

As the step shown above, we provide various tools to help a user access an information on the information resource in an integrated manner. The basic components and tools can be divided into the following parts:

1. Wrapper and Persistent Perl[7] Object Model
2. Query Language
3. Mediator on Mobile Computer

The detail of each component will be described in the next subsections.

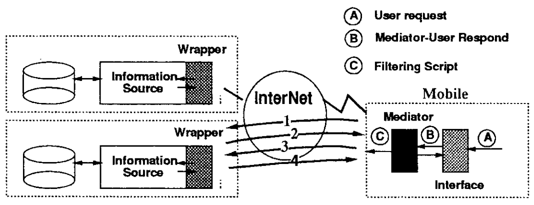

Figure 2: An Information-Retrieval Processing

2.1 Wrapper and Persistent Perl Object Model

A wrapper plays the key role of the integration in our architecture. The wrapper is a group of programs that translates requests and data between native information resource and the Persistent Perl Object Model. In the other words, the native data items and their access method must be described by class definitions of Persistent Perl. They are then registered into a data dictionary. With this mapping information, the wrapper function can perform the following work:

1. providing mobile computer user these class definitions corresponding to his request.
2. When receiving a script program from the user, it translates the script program to the native requests, executes them, converts a result into Persistent Perl object model and send the result of the execution back to the user.

To illustrate the wrapper, consider the class definition shown in figure 3.

The class definition in figure 3 defines a class called *Person*. This class contains a **body** part that specifies a fixed number of arbitrary attributes. In a **method** part, we have to provide the mapping between the native services to the methods in Persistent Perl style. In a **interface** part, we can provide a rule for each domain of arguments of the methods of class definition. The functions of this part will be explained in the next section.

Suppose that the underlying information resource is a relational database and the wrapper program retrieves 2 records of data from the database. The values of the information are shown as follows:

```
Person Data
============
  [967006,Wisut Sae-tung,01-22-1966,M,Eng]
Occupation Data
===============
  [Eng, Engineer]
```

Using Persistent Perl object model, we can represent the above value as follows:

```
persistent class Person
body
  Code          string,
  Name          string,
  BirthDate     string,
  Sex           char,
  Occupation refto Occupation
method
PersonClass($Code, $Name, $BirthDate,
            $Sex, $Dept)
begin
  // coding of constructure function //
end
checkName($name)
begin
  if ($self->{Name} =~ /$name/ {
     return 1;
  }else{
     return 0;
  }
end
interface
...
endclass
```

Figure 3: Class Definition

```
  Person_1->{Code} = '967006',
  Person_1->{Name} = 'Wisut Sae-tung',
  Person_1->{BirthDate} = '01-22-1966',
  Person_1->{Sex} = 'M',
  Person_1->{Ocuppation} = 'Occupation_1',
  Occupation_1->{Code} = 'Eng',
  Occupation_1->{Name} = 'Engineer',
```

Person_1 and *Occupation_1* are Perl's associated variable references. From the point of view of object, they act as object identifiers. In consequence of this, the value of *Person_1*→{*Occupation*}→{*Name*} is 'Engineer'.

2.2 Query Language

Cooperated with Persistent Perl object model, we provide users with an Object-Oriented SQL, or OO-SQL, as an integrated query language for our system. Not only object qualifications, users can also specify their filtering methods as the condition of query to manipulate information as preferred.

```
select S->{Code},S->{Name},
       S->{BirthDate},S->{Sex},
       S->{Occupation}->{Code}
from   S in PersonClass
where  &horoscope(S->{BirthDate},'06/23/1966'}
```

From the example shown above, a user wants to find the information of the person whose characters agree with his by using the horoscope method. This helps the user to be able to customize a filtering process into his own style.

2.3 Mediator on Mobile Computers

To help users rapidly generating a filtering script, we have provided the mechanism on the mobile computer called "mediator". The mediator generates a filtering script of Persistent Perl from OO-SQL command. It collects up some library calls provided in information resources, user's data, user's filtering methods and puts them into a filtering script in Persistent Perl language. The filtering script will be sent to the information resources to be executed for retrieving information.

3 Problems and Necessary Functions in Script Generating

In this section, we will describe about the problems in script generating while mobile computer users are disconnecting from network and describe how to solve these problems.

3.1 Problems in Script Generating

In the previous section, we have described the overall processing model. In our model, mobile computer users can generate a filtering script by using OO-SQL while they are disconnecting from network. They connect to the network, send the filtering script during a short connection and then disconnect from the network. Even though this approach is the best style for the mobile computing environment, there are some additional necessary functions for generating a filtering script. In contrast to the client/server model, mobile computer users cannot check whether they enter data matching with the data required by information resources. This problem arises because their processes are not connecting to the processes on the information resources. Furthermore, this problem always arises when users try to join more than two classes that use different domains or formats for representing the same data.

3.2 Necessary Functions for Generating Script

To solve the above problem, we need some necessary functions for generating a filtering script while users disconnect from information resources. They are listed as follows:

1. Information resources provide users with not only class definition **but also how to use access methods in class definition** . These information are used as rules to resolve data-domain and data-type mismatch of an integration problem.

2. Mobile computers should be able to recognize the above resources and use them to generate a filtering script while disconnected from network.

4 Integration Rule Specification for Mediator

Now, suppose that users want to access to some of the information stored in 2 information resources that represent data using different format. For example, the first one represents date format using Heisei format while the second one represents date format using American format. To understand our system more clearly, let us explain the integration rules in a interface part of a class defintion and how a mediator on mobile computer can use them to generate a filtering script through the following example.

As shown in figure 4, the specification of the interface part can be divied in the following components:

1. *Rule:* This part declares the rule for each domain of the arguments of class-associated methods. In this example, it is the rule for a Heisei Date domain.

2. *External Program:* This part declares a group of functions that are provided by the information resource. These functions can be used to check or decompose the input arguments. This example shows that the information resource provides three checking functions, American, European and Heisei. These functions are used to check the input argument and return a Boolean value.

3. *Comment:* This part declare the message shown to users when they are going to enter data for the argument of the corresponding domain. It is useful for explaining some information for users. For example, data format, how to use this access method and so on.

4. *Default:* This part declare a sample of data of its domain. In this example, the sample data of Heisei date format is "H7/04/23."

This example shows about querying a reservation for both a train and a hotel. User wants to know whether there is an available room of hotel and seat of train to his destination at the certain time. the user requests the class definitions to the information resources and they send back the class definitions to the user. The class definitions of *RoomClass* and *TrainClass* are shown in figure 4 and 5 respectively.

After receiving the class definitions, the user registers them into his database on mobile computer. Suppose that he want to find out an available train from Tokyo to Osaka on 25/12/1995 (DEC 25, 1995) and an available room from 25/12/1995 to 26/12/1995. The user will use the *VacantRoom($startDate Date, $endDate Date)* method of the *RoomClass* and the *VacantTrain($DepartTime Date, $Departure string, $Destination string)* method of the *TrainClass*. The *Room-*

```
persistent class RoomClass
body
  TypeOfRoom              string,
  RoomNo                  string,
  Schedule setof RoomScheduleClass,
method  // access methods for RoomClass
RoomClass($TypeOfRoom string, $RoomNo string)
  // code program
VacantRoom($startDate Date,$endDate Date)
  // code program
interface // interface rules for RoomClass
Rule:
Date($X)     :- chkType($X).
chkType($X)  :- Heisei($X),output($X).
chkType($X)
   :- European($X),output('&Eu2Heisei('.$X.')').
chkType($X)
   :-American($X),output('&Am2Heisei('.$X.')').
External:
    Heisei(input) code is Heisei.pl
    European(input) code is European.pl
    American(input) code is American.pl
Comment: How to use.
Default: H7/04/23
endclass
```

Figure 4: A HotelClass Definition

```
persistent class TrainClass
body
  TypeOfTrain        string,
  TrainNo            string,
  TimeSchedule setof TimeScheduleClass,
method   // access methods for TrainClass
TrainClass($TypeOfTrain string, $TrainNo string)
// coding //
VacantTrain($DepartTime Date,$Departure string,
            $Destination string)
// coding //
interface // interface rules for Trainclass
Rule:
Date($X)     :- chkType($X).
chkType($X)  :- American($X),output($X).
chkType($X)
   :- European($X),output('&Eu2Am('.$X.')').
chkType($X)
   :- Heisei($X),output('&Heiei2Am('.$X.')').
External:
    American(input) code is American.pl
    European(input) code is European.pl
    Heisei(input) code is Heisei.pl
Comment: How to use
Default: 23/02/1996
```

Figure 5: A TrainClass Definition

Class uses Heisei date format, but the *TrainClass* use American date format. For this reason, The date format values supplied to the *VacantRoom* and the *VacantTrain* must be Heisei and American date format respectively.

By acting as an mobile computer assistent, we provide users with an user interface so that the user can form OO-SQL command easily. While communicating with the user through the user interface, the mediator on mobile computer uses rules provided from the information resources to solve the problem shown above. The detail of the user interface are shown in [9]

The execution of the interface rule is similar to Prolog execution process that use depth-first search algorithm to find out the goal of the execution. The *mediator* on mobile computer will invoke a rule when user enters data for the argument whose domain corresponds with the rule. For the above example, when the user enter ' 25/12/1996' for argument *$startDate* of the *VacantRoom* method, the mediator will invoke the *Date* rule, execute and send `&Eu2Heisei('25/12/1996')` as a return value. From the requirement we have described, the result can be shown as below:

```
select R->{RoomNo},T->{TrainNo}
from R in RoomClass,T in TrainClass,
where &VacantRoom(R,&EuToHeisei('25/12/1995'),
                  &EuToHeisei('26/12/1995'))
and  &VacantTrain(T,&HeiseiToAm('25/12/1995'),
                  'Tokyo','Osaka')
and &myBudget(R->{TypeOfRoom},T->{TypeOfTrain},
     's','vip','Shinkansen','Regular')
```

5 Discussion with Related Work

Integration of heterogeneous information resources has become one of important research directions. There are many researches that provide solutions to solve this problem. One of the outstanding projects is the TSIMMIS[2,3] that is developed at Stanford university.

The TSIMMIS provides environment so that users can freely access information stored on the heterogeneous information resources. TSIMMIS provides OEM-QL[2], the language resembles SQL, for users to query information from heterogeneous resources. The *mediator*[3] provides global integrated view based on OEM[3] so that users do not need to recognize where the information is stored.

Even though users can use OEM-QL to query the provided information, they can use only object qualification as query condition. They cannot provide their own algorithms for accessing information. This restricts users to use the same style as SQL command. The other restriction of the TSIMMIS is the known integration view that must be prepared by mediator. In the mobile environment, While disconnecting from network, mobile computer users can not create their own view on the spot by their mobile computers to access information from unknown information resources.

The global view of our solution can be shown in figure 6. In our environment, a user will collect *class definitions, utility programs* provided from information resources and *his own filtering methods* in his mobile computer and move within network (step 1). When the

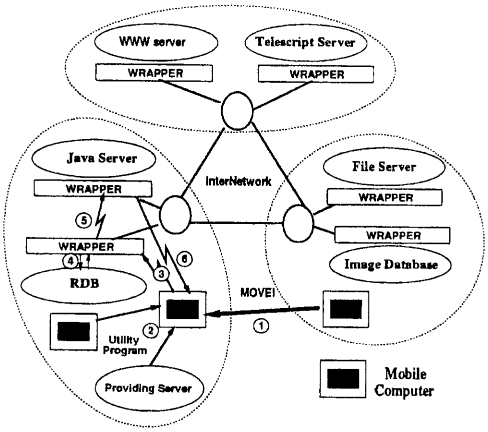

Figure 6: The Global-View Architecture

user comes across unknown information resources, he then uses his filtering methods and the class definitions sent from these information resources to generate his own style of a filtering script while he is disconnecting from network. However, the interface rules provided by each information resource may be insufficient for integration. In this case, the user can receive interface rules and utility programs from other mobile users or servers and uses them to generate the filtering script again (step 2). After finishing script generating, the user will send the filtering script to the information resources to gather the relevant information (step 3). A wrapper will convert the queries of the filtering script into native query commands of the underlying information resource and transforms the result of the query into Persistent Perl object model (step 4). In case that the filtering script must be executed in several information resources to complete its work, it moves to another information resource (step 5). It contacts with the wrapper in that information resource as the same manner shown above. When finishing all works, the result will be sent back to the user (step 6).

As described earlier, our propose has a number of advantages as follows:

1. Information-retrieval process is performed on information resources without intensive network communication.

2. A Mobile computer user can use his own filtering methods, class definitions and rules from information resource or other mobile users to generate a customized filtering script. He does this process on the spot they meet an the unknown information resources during they disconnect from network.

3. A wrapper and Persistent Perl can act as a broker that helps user to access information on heterogeneous information resources in an integrated manner.

Note, a very important difference between our system and TSIMMIS is the operation environment. In mobile computing environment, users move in network and may want to access unknown information resources on the spot by their mobile computers which still have resource capacities and network bandwidth limitations. These characteristic and limitations of mobile computing constrict the way of use that differ from the traditional client/server model.

6 Conclusion and Project Status

In this paper, we have proposed an environment and a set of tools for mobile computer users to integrate information in heterogeneous information resources. We have implemented the Persistent Perl and use it as the communication language in our system. We have presented the wrapper function on information resource, the OO-SQL, the mediator function and the user interface on mobile computer. We describe how to use our tools to access information on heterogeneous environment through the example of the global view of our system. We believe that our proposal is so natural a way of use and it will be an important step toward for achieving information-retrieval in mobile computing and heterogeneous environment.

The initial work consisting of all modules described above have been developed. However, The information resources we used for testing are implemented by Persistent Perl. We are currently implementing The wrapper for Postgres95 database and applications built by Java language.

References
1: T. Imielinski and B. R. Badrinath.: "Data Management for Mobile Computing." SIGMOD Record, 22(1):34-39, March 1993.
2: Chawathe,S. 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.
3: Y. Papakonstantinou, et al.: "Object exchange across heterogenous information sources." In IEEE Int. Conf. Data Engineering '95., pp.251-260, 1995.
4: P. Wayner. "Agents Unleashed" AP PROFESSIONAL 1995.
5: A.Silberschatz, M.Stonebraker, J.D.Ullman ed.: "Database Systems: Achievements and Opportunities", *ACM SIGMOD Record*, Vol.19 (No.4), pp.6-22, 1990.
6: J. Gosling, H. McGilton. The Java Language Environment *A white paper*. Sun MicroSystems Co. Available from http://java.sun.com/ .
7: L.Wall, Perl reference guide for Perl ver.5.
8: 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」電子情報通信学会研究報告 DE95-54, 95 年 10 月.
9: 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 5 2 回 全国大会 7P-6, 96 年 3 月.