Using SMS Gateway to Provide Generic and Personalized Services by Abstracting and Specializing SQL Queries

[1]*MUHAMMAD SALEEM*, [2]*IQBAL QASIM*, [3]*ATA-UR-REHMAN*

[1]*Digital Enterprise Research Institute, National University of Ireland,* ***Ireland***

[2]*Department of Computer Science and Engineering, Hanyang University,* ***South Korea***

[3] *Department of Electronics, Politecnico di Torino, Torino,* ***Italy***

muhammad.saleem@deri.org, qasim@hanyang.ac.kr, ata.rehman@polito.it

Abstract

This paper proposes a query abstraction mechanism which allows web/mobile-service administrator to formulate a skeleton of a sequence of SQL queries by parameterizing holes, later being filled by end-users. The mechanism is generic as the administrator can use it to register multiple services, and expandable as the existing service can be specialized, to automatically generate new kinds of personalized services. An end-user's input, when given in its entirety, initiates the automatic generation of appropriate SQL queries suitable for the user's requested service. A personalized service can be devised by designating the end-user's input parameters into static or dynamic. When static input arguments are given, a specialized skeleton service with respect to the given input is created. The mechanism is implemented to be used in systems for web/mobile-based information and transaction services.

Keywords**:** *Query formulization, Generic information/transaction services, Skeleton queries, personalized services*

1 Introduction

Many big firms greatly depend upon the functions of online databases as their daily business operations cannot be accomplished without the availability of databases. Remote database information and transaction services are considered as a lifeline of their profitability in today's highly mobile society. These remote services make life easier for human being. Instead of waiting in long queues, operations can be done remotely any time anywhere [26], [27].

The increasing demand for new remote services makes it hard to develop a standalone system for every new Web/SMS-based information or transaction service. Programmers are interested in developing systems which can provide multiple services using a single system and also take the extensibility and re-usability into consideration [1]. Currently, the extendable generic systems presented in [1-3] only provide generic data retrieval services and cannot be used for data insertion, deletion and updating services such as bill payments, reservation, etc. Moreover, generating dedicated, personalized and user-specific services by decomposing the root service has been a source of recent research interest. This service personalization generates records according to the user requirements with minimum effort and time.

Keeping in view the need for generic web/mobile-based database services; we propose an efficient mechanism for generic information and transaction services using a single extendable system. Administrator formulates Skeleton SQL queries for various information/transaction services using a graphical interface. A skeleton query is the template of the original SQL query containing some holes/gaps to be filled from the input provided by users. Each registered service can contain a sequence of SQL queries. For a specific service, a user has to provide certain information either by sending an SMS message in a predefined format or by using a website input form.

The structure of every service is exactly similar to a typical programming language function having unique service number as function name, user provided inputs as function arguments and all the skeleton queries as function body. All the services data is stored in a repository as functions. For a specific service, a user input is forwarded to the proposed extendable system for necessary queries execution. Based on the user input, a proper service function is invoked from the service repository with proper arguments values selected from the user input. The skeleton queries are then transformed to the actual SQL queries and execute one by one against a specific database. After successful queries execution, a prompt reply is forwarded to the user.

We used the concept of partial execution of queries for personalized services generation; in which part of the service queries are already constructed using the static information provided by the user. With this concept, personalized services [4-8] are provided to the users with minimum possible user-provided data. Once a mobile/web-service administrator registers a new service (root service), end user can personalize (sub service) this by specifying his/her static inputs. Each personalized service is dedicated to a single end user. Every user-specified static input fills a specific hole/gap permanently in the skeleton queries of the root service. This means that a specific end-user is not required to provide static inputs for his/her personalized service. A personalized service is automatically generated by the end user rather than mobile/web-service administrator. Service personalization makes the system highly scalable

as the mobile/web-service administrator can register any number of root services and each end-user can personalized a root service any number of times. The details of service personalization are given in Section 4.

In case of the SMS based input, a user will send an SMS in a predefined format to mobile Gateway which will forward it to the extendable generic system for necessary query execution. After the successful operation, a prompt reply will be forwarded to the user in reverse order. While, for the web based input, a user will fill an input form available on a specific services website. The proposed system is more worth for the SMS based services because of the no restriction of internet connectivity.

1.1 Motivating Example

Consider a transaction service "Bus Ticket Reservation and Payments" in which passengers reserve and pay for various bus tickets either through SMS or using web pages. For a specific reservation, a passenger provides ID, password, starting and destination locations, and reservation date. Every passenger needs to register a specific account and should have enough money in his/her account for the reservation payment. The ERD (Entity Relationship Diagram) for the given system is shown in Figure 1.

Table *Ticket_Details* stores complete information about various available, reserved tickets, *Passenger_Account* stores passengers' data and account information, and *Pass_TravelHistory* stores all the traveling history of passengers for querying in the future. The column *Ticket_Status* has a default value as *Available*. If the passenger query is matched to an available ticket, the reservation takes place by changing the status to *Reserved* and inserting passenger Id into *the Ticket_Details.Passenger_Id* column.

The step- by- step functionalities for the service are as under:

1. For a specific ticket reservation and payment, the registered passenger provides *ID, Password, Starting Terminal Name, Destination Terminal Name, and Reservation Date* as inputs. These inputs can either be provided through SMS in a proper format or using a specific input form in a web page.
2. All the Passengers must be authenticated before making any type of reservation. The authentication should be based on passenger *ID, Password*.
3. The available total amount *Available_Amount* in the passenger account must be large enough to pay for a specific ticket, which satisfies the passenger input requirements.
4. After the completion of step 2 and 3, a specific ticket in a particular bus should be reserved for a passenger. The reservation must take place according to the supplied input conditions. If multiple condition matches, only one ticket will be reserved, prioritised by bus timing.

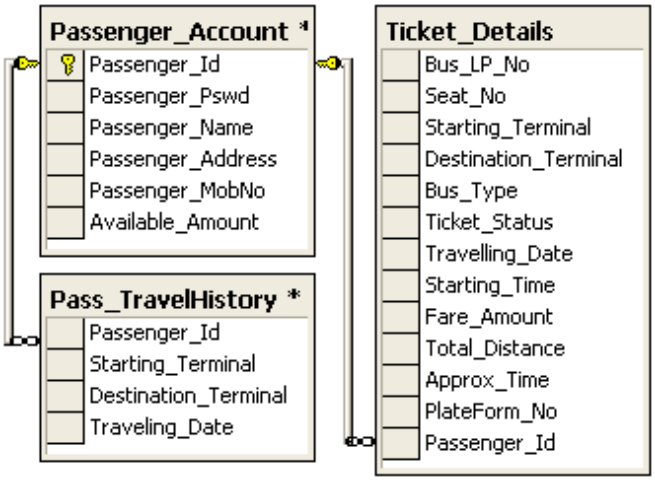

Figure 1 ERD of the ticket reservation

5. The customer *Available_Amount* amount must be updated after the successful payment for the specific journey ticket.
6. The travelling details should be stored in the *Pass_TravelHistory* so that in the future a passenger can query for his/her journey details.

After proper authentication, ticket reservation and payment, the response to the passenger about reservation will be *Seat no, Travelling date, Departure time, Platform no, Fare amount, Total distance, approximate time* and the *remaining amount* in the account.

For a successful bus ticket reservation and payment operation, each of the steps from 2 to 7 requires an SQL query to be executed against the given database. Step 2, 3 and 7 require SQL Select queries, step 4 and 5 requires *SQL Update* queries and step 6 require an *SQL Insert* query.

A specific passenger's input data *"MobileService#1 saleem greatwazir ansan suwon 3/10/2010"* means a passenger *saleem* having password *greatwazir* wants to reserve a bus ticket from *Ansan* to *Suwon* on *3/10/2010*.

Since the step 4, step 5 updates and step 6 insert records in to the database; the generic systems [1-3] are not able to provide such type of transaction services. The main aim of the proposed system is to provide services in which multiple insert, delete and update query processing is required for every new user input and let the end user able to further personalize any root service.

In stand alone system's implementation, all the required service queries are written by a programmer and are fixed in a specific program location. While in the proposed system, all the service queries are graphically generated by the administrator and are stored in the repository. The corresponding queries are then dynamically retrieved from the repository and are executed based on the input data. Hence in the traditional way of implementation, query execution is fixed while it is dynamic in the proposed system.

The paper is organized as follows. Section 2 highlights related work, Section 3 define service

abstraction as series of SQL queries, Section 4 describes the concept of personalized service registration and its details, Section 5 introduces the visual tool for mobile-services generation, Section 6 define proposed architecture for implementing this generic queries processing system, Section 7 shows a typical user input processing, Section 8 contains practical example of information's service for which this model can be used and Section 9 describes the conclusion.

2 Related Work

Authors [1, 2] developed a generic application, which can be used to provide more than single service. The system administrator is able to register new messaging services in a simple way without any programming or design changes. The generic application dynamically communicates with databases and extracts information based on the contents of the SMS. However, the system only provides information services and cannot be used for transactions like inserting, updating or deleting records. These systems only relies on execution of generic *SQL-Select* queries and cannot consider the generic execution of *SQL-Insert*, *Update* and *Delete* queries. Also these systems does not support personalized service registration.

Ahmad and Kareem [3] design a database language called Free-Form; which enable user to issue any type of query. The users are able to formulate new queries and get information according to their interest. This model is a very good alternative for those stand alone systems which only provide minimal querying capabilities. The possible queries that can be formulated on these systems are mostly pre-determined by the developers. Such a measure tends to limit the usage of these systems. It leaves no room for users to issue any query of interest. However, this work does not address the issues of [1] as well. It allows users to formulate only *SQL-Select* queries and service personalization is not supported. Furthermore, database experts believe that it is hard for an ordinary user to formulate a desired complex query using only schema information.

Jarir et al. [4] proposed an SOA based architecture for personalized on-the-fly web services for web information extraction. Web extraction services can be modified on-the-fly without stopping the current process. Georgia and Yannis [5] developed a personalized framework for database systems based on the user-interests shown in the profile. Some of the other research related to personalized services generation is shown in [6-8], [28]. However, none of the above described the SMS based personalized services generation.

In SOA, there are standards like, Web Service Definition Language (WSDL) [21], Business Process Execution Language (BPEL) [22], WS-Coordination [23], WS-Atomic Transaction [24] and WS-Business Activity [25] used to describe transactional services and their compositions. Some of them like WSDL provides graphical user interface to modify XML files used for a specific service. These standards are very useful for generating information and transactional services. However, we believe that it is not trivial to generate and control large number of personalized services with the above standards.

Other notable applications based on SMS/Web technology are hospital search and appointments [9, 10], Regional information services [11], Public Transport Service [12, 13], Mobile commerce and Banking [14, 15], Sales Reporting [16], Mobile-Quiz [17], SMS Blogging [18], Real time information exchange [19] and Payments [20]. Each of these applications provides specific information and did not take the extensibility and re-usability into consideration. Thus if system administrators want to add some new functionalities to an existing system, they have either to make changes to the existing system or rebuild the entire system from scratch [1].

3 Service Abstraction

We regard each mobile service as a procedural abstraction of a sequence of database queries. Each mobile service is treated as a *procedure* having a unique identifier. The mobile service, like usual procedure in programs, can have zero or more formal parameters that are to be bound to the values supplied later by mobile-service users. The body of a mobile service consists of a sequence of database queries. Mobile-service declarations are prepared by a mobile-service administrator. A mobile-service user can invoke a mobile service by supplying a service name followed by input data corresponding to formal parameters of the service. Then the input values are bound to formal parameters and the queries in the body are sequentially executed in that environment. The following is an example of a mobile-service declaration for the motivating example.

```
MobileService#1
    Input Parameters:
        PId: string
        Paswd: string
        StrtTerm: string
        DestTerm: string
        TravDate: string
   {
   --- Step 2 (Skeleton SQL-Select Query) ---
Select *from Passenger_Account where
Passenger_Id = PId and passenger_Pswd = Paswd
;

  --- Step 3 (Skeleton SQL-Select Query) ---
Select *from
Passenger_Account,Ticket_Details where
Passenger_Account.Passenger_Id = PId and
Passenger_Account.Passenger_Pswd = Paswd
and Ticket_Details.Starting_Terminal =
StrtTerm and
Ticket_Details.Destination_Terminal =
DestTerm and Ticket_Details.Travelling_Date
= Convert(datetime, TravDate)and
Convert(int,Passenger_Account.Available_Am
ount)>=
Convert(int,Ticket_Details.Fare_Amount);
```

```
 --- Step 4 (Skeleton SQL-Update Query) ---
Update Top(1) Ticket_Details  set
Ticket_Status = 'Reserved',Passenger_Id =
PId where Ticket_Details.Ticket_Status =
'Available' and
convert(int,Ticket_Details.Seat_No)
=(Select TOP 1 Ticket_Details.Seat_NO from
Ticket_Details where
Ticket_Details.Starting_Terminal = StrtTerm
and Ticket_Details.Destination_Terminal =
DestTerm and Ticket_Details.Travelling_Date
= Convert(datetime, TravDate)and
Ticket_Details.Ticket_Status = 'Available')
order by  Starting_Time;

--- Step 5(Skeleton SQL-Update Query) ---
Update Passenger_Account set
Passenger_Account.Available_Amount =
Passenger_Account.Available_amount -
Convert (int,(Select top 1
Ticket_Details.Fare_Amount from
Ticket_Details where
Ticket_Details.Passenger_Id  = PId and
Ticket_Details.Starting_Terminal = StrtTerm
and Ticket_Details.Destination_Terminal =
DestTerm and Ticket_Details.Travelling_Date
= Convert(datetime,TravDate)and
Ticket_Details.Ticket_Status = 'Reserved'))
Where Passenger_Account.Passenger_Id  = PId
;

 --- Step 6 (Skeleton SQL-Insert Query) ---
Insert into Pass_TravelHistory values
(PId,StrtTerm, DestTerm, TravDate);

 ---Step 7 (Skeleton SQL-Select Query) ---
Select Seat_no,Travelling_Date,
Starting_Time, PlateForm_No,Fare_Amount,
Total_Distance,Approx_Time,Available_Amoun
t from Ticket_Details, Passenger_Account
Where Passenger_Account.passenger_id = PId
and Travelling_Date = Convert(datetime,
TravDate) and ticket_status = 'Reserved'

            }
```

Each query in the body of a mobile-service declaration is called a skeleton query because some parts of the query are filled with a formal-parameter variables. For the skeleton query of step 3, there are two possible executions; either a direct bus from *Ansan* to *Suwon* does not exist at the given date or the available amount in the passenger account is not sufficient. Here we are assuming that the bus service always exist between the two terminals. However, we can also divide the above query into two: first will check for the availability of the bus service between the source and destination and second will check for the sufficient amount in the passenger account. If any of the queries is failed then a corresponding response will be forwarded to the passenger. Also, for simplicity we are not considering the timing constraint of ticket reservation.

If a user wants to use this service and to know the availability of a bus from Ansan to Suwon on March 10, 2010, then she can invoke this service by supplying all actual arguments “MobileService#1 saleem greatwazir ansan suwon 3/10/2010”.

If a user wants to personalize the service in her mobile phone, she can do so by specializing the service with respect to her identifier. This specialization can be carried out by invoking the service with only the first argument, say “MobileService#1 saleem _ _ _ _ ”. The underscore ,_ , means that the argument is not going to be supplied this time. Then the specialized service is named as “MobileService#1_saleem” and every instance of PId in the body is replaced by ‘saleem’ as followed.

```
MobileService#1_saleem
    Input Parameters:
         Paswd: string
         StrtTerm: string
         DestTerm: string
         TravDate: string

 {
   --- Step 2 (Skeleton SQL-Select Query) ---
Select *from Passenger_Account where
Passenger_Id = 'saleem' and passenger_Pswd =
Paswd ;

  --- Step 3 (Skeleton SQL-Select Query) ---
Select *from
Passenger_Account,Ticket_Details where
Passenger_Account.Passenger_Id = 'saleem'
and Passenger_Account.Passenger_Pswd =
Paswd and Ticket_Details.Starting_Terminal
= StrtTerm and
Ticket_Details.Destination_Terminal =
DestTerm and Ticket_Details.Travelling_Date
= Convert(datetime, TravDate)and
Convert(int,Passenger_Account.Available_Am
ount)>=
Convert(int,Ticket_Details.Fare_Amount);

 --- Step 4 (Skeleton SQL-Update Query) ---
Update Top(1) Ticket_Details  set
Ticket_Status = 'Reserved',Passenger_Id =
'saleem' where Ticket_Details.Ticket_Status
= 'Available' and
convert(int,Ticket_Details.Seat_No)
=(Select TOP 1 Ticket_Details.Seat_NO from
Ticket_Details where
Ticket_Details.Starting_Terminal = StrtTerm
and Ticket_Details.Destination_Terminal =
DestTerm and Ticket_Details.Travelling_Date
= Convert(datetime, TravDate)and
Ticket_Details.Ticket_Status = 'Available')
order by  Starting_Time;

--- Step 5(Skeleton SQL-Update Query) ---
Update Passenger_Account set
Passenger_Account.Available_Amount =
Passenger_Account.Available_amount -
Convert (int,(Select top 1
Ticket_Details.Fare_Amount from
Ticket_Details where
Ticket_Details.Passenger_Id  = 'saleem' and
Ticket_Details.Starting_Terminal = StrtTerm
and Ticket_Details.Destination_Terminal =
DestTerm and Ticket_Details.Travelling_Date
= Convert(datetime,TravDate)and
Ticket_Details.Ticket_Status = 'Reserved'))
```

```
Where Passenger_Account.Passenger_Id  =
'saleem' ;

 --- Step 6 (Skeleton SQL-Insert Query) ---
Insert into Pass_TravelHistory values
('saleem',StrtTerm, DestTerm, TravDate);

 ---Step 7 (Skeleton SQL-Select Query) ---
Select Seat_no,Travelling_Date,
Starting_Time, PlateForm_No,Fare_Amount,
Total_Distance,Approx_Time,Available_Amoun
t from Ticket_Details, Passenger_Account
Where Passenger_Account.passenger_id =
'saleem' and Travelling_Date =
Convert(datetime, TravDate) and
ticket_status = 'Reserved'

}
```

The personalized service is deposited at the server and can be invoked by the requested user over and over again. That is, the user can now invoke her personalized service "MobileService#1_saleem" with 4 arguments instead of the general service "MobileService#1" with 5 arguments. If the user does not want to type the password every time she uses this personalized service, she can even register the service with respect to both identifier and password, say "MobileService#1 saleem greatwazir _ _ _". From then on, she can use her service "MobileService#1_saleem+greatwazir" with 3 arguments over and over again.

4 Service Personalization

Consider the motivating example again: if a specific passenger frequently uses the bus service from *Ansan* to *Suwon*, she can further specialize the personalized service with respect to the route by invoking "MobileService#1_saleem (_ , ansan, suwon, *_)". This invocation asks the server to create a specialized service for Saleem with Starting_*Terminal and Destination_Terminal fixed as Ansan and Suwon, respectively. Then the specialized service generated by the server is as follows:

```
MobileService#1_saleem+suwon+ansan
    Input Parameters:
         Paswd: string
         TravDate: string

 {

--- Step 2 (Skeleton SQL-Select Query) ---
Select *from Passenger_Account where
Passenger_Id = 'saleem' and passenger_Pswd =
Paswd ;

  --- Step 3 (Skeleton SQL-Select Query) ---
Select *from
Passenger_Account,Ticket_Details where
Passenger_Account.Passenger_Id = 'saleem'
and Passenger_Account.Passenger_Pswd =
Paswd and Ticket_Details.Starting_Terminal
= 'ansan' and
Ticket_Details.Destination_Terminal =
'suwon' and Ticket_Details.Travelling_Date
= Convert(datetime, TravDate)and
Convert(int,Passenger_Account.Available_Am
ount)>=
Convert(int,Ticket_Details.Fare_Amount);

 --- Step 4 (Skeleton SQL-Update Query) ---
Update Top(1) Ticket_Details  set
Ticket_Status = 'Reserved',Passenger_Id =
'saleem' where Ticket_Details.Ticket_Status
= 'Available' and
convert(int,Ticket_Details.Seat_No)
=(Select TOP 1 Ticket_Details.Seat_NO from
Ticket_Details where
Ticket_Details.Starting_Terminal = 'ansan'
and Ticket_Details.Destination_Terminal =
'suwon' and Ticket_Details.Travelling_Date
= Convert(datetime, TravDate)and
Ticket_Details.Ticket_Status = 'Available')
order by  Starting_Time;

--- Step 5(Skeleton SQL-Update Query) ---
Update Passenger_Account set
Passenger_Account.Available_Amount =
Passenger_Account.Available_amount -
Convert (int,(Select top 1
Ticket_Details.Fare_Amount from
Ticket_Details where
Ticket_Details.Passenger_Id  = 'saleem' and
Ticket_Details.Starting_Terminal = 'ansan'
and Ticket_Details.Destination_Terminal =
'suwon' and Ticket_Details.Travelling_Date
= Convert(datetime,TravDate)and
Ticket_Details.Ticket_Status = 'Reserved'))
Where Passenger_Account.Passenger_Id  =
'saleem' ;

 --- Step 6 (Skeleton SQL-Insert Query) ---
Insert into Pass_TravelHistory values
('saleem','ansan', 'suwon', TravDate);

 ---Step 7 (Skeleton SQL-Select Query) ---
Select Seat_no,Travelling_Date,
Starting_Time, PlateForm_No,Fare_Amount,
Total_Distance,Approx_Time,Available_Amoun
t from Ticket_Details, Passenger_Account
Where Passenger_Account.passenger_id =
'saleem' and Travelling_Date =
Convert(datetime, TravDate) and
ticket_status = 'Reserved'

}
```

Then the user can invoke her own specialized service whenever she wants to travel from Ansan to Suwon, say "MobileService#1_saleem+ansan+suwon greatwazir 3/10/2010". Such type of dedicated services can greatly increase the importance of the system. Based on the existing services, the user will be able to generate new personalized services according to their wishes. The input format errors will be greatly reduced and users will get various services with the less possible inputs.

With this idea each root service can have more than one personalized services. Each personalized service is dedicated to only one user as it is generated by a specific user according to its wishes. All the personalized services

must have a single root service.

Beside user facilitation, personalized services also reduces the communication cost and possibility of security attacks such as SQL injections as each of the above metric is directly proportional to the number of user-supplied arguments.

5 A Visual Tool for Generating Mobile Services

In this section, we demonstrate a visual tool that assists a mobile-service administrator to compose a new mobile service consisting of a sequence of SQL queries. Each SQL query is generated by the visual tool. For each of the available database, the whole database schema is graphically made available in drop-down lists and menus. Names representing formal parameters of the mobile service should be prepared by the administrator beforehand. Then the mobile-service administrator is only required to pick a specific attribute from the list and assign proper value or name to construct SQL queries just as shown in Figure 2. Each attribute value can be; a constant string, SQL function, another attribute, output of an SQL query or a formal-parameter name. In this way, the complexity of understanding data structures is moved to the graphical tool.

After generating all the queries, the administrator builds the final service by sequentially concatenating the queries. The graphical tool arranges all the formal parameters and generated queries in the specified service format and stores service data in the repository. Figure 2 shows the screen shot of the graphical tool used for new service registration.

The mobile-service administrator is able to use various filters (e.g, + , * , > , <= , != etc), SQL functions (e.g, Int, Convert, Max etc), query pipes (query within query), and various table joins for multiple-table records retrieval in graphical way. Using the query pipes, queries within query can be formulated such that the output of one query can be used as an input value for another attribute.

The graphical way of queries formulization significantly facilitates the administrator in terms of database schema understanding, fast query formulization, and also avoids the possibility of syntax errors while manually typing the service queries.

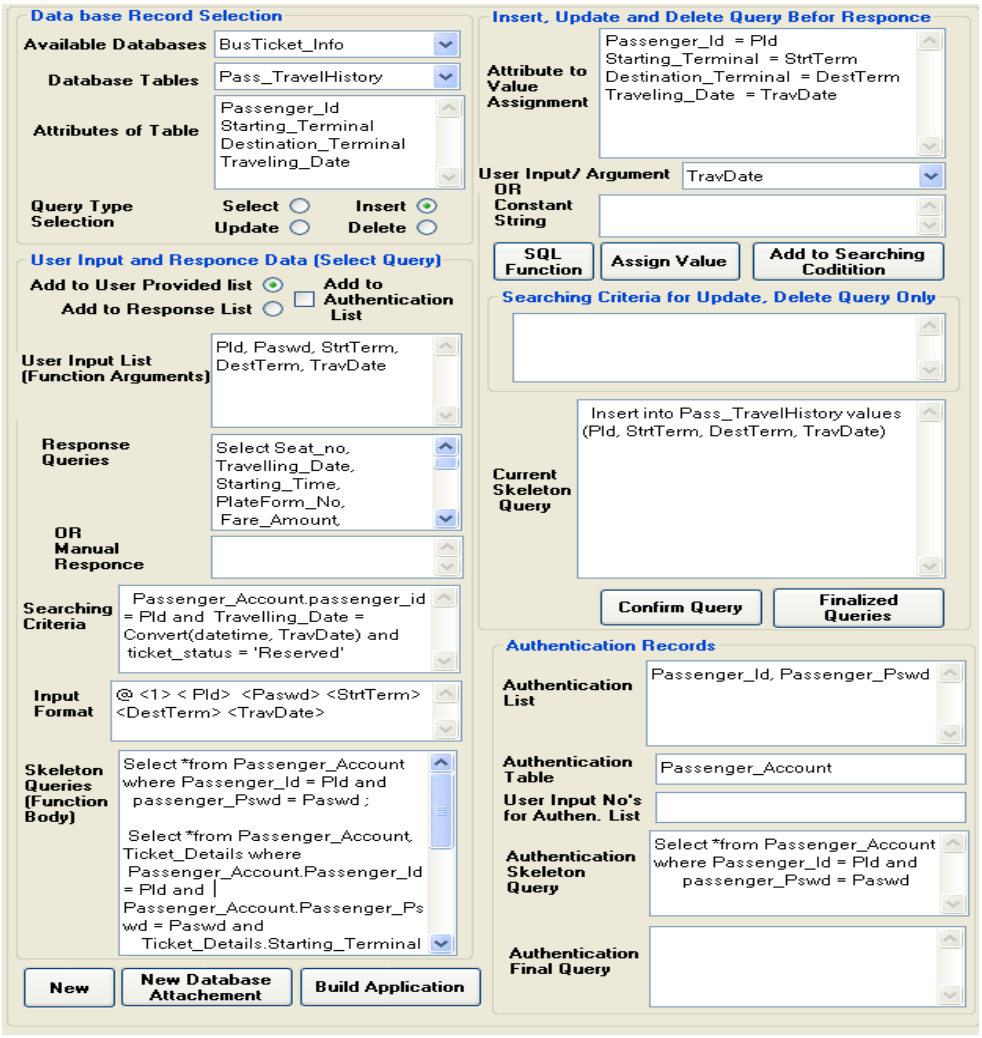

Figure 2 Screen shot of part of the graphical services generator tool

6 Proposed System Architecture and Implementation

Figure 3 shows the proposed architecture for providing information and transaction services. The *Generic Mobile-Services Provider* becomes the main entity, communicating with the user as well as database systems. This application runs on the server computer and provides graphical interface to the mobile-service administrator for new service registration. The application also processes the user input for the generation and execution of final queries. The details about the main entities of our proposed system are explained below.

Mobile Services Generator (MSG) – New services are registered through MSG. Before users can get any service, the mobile-service administrator must register it first. MSG provides a graphical interface to the mobile-service administrator to make Skeleton queries and also automatically generates the user input format. The fixed input format is only mandatory for the SMS based services. For web based service, an automatic user

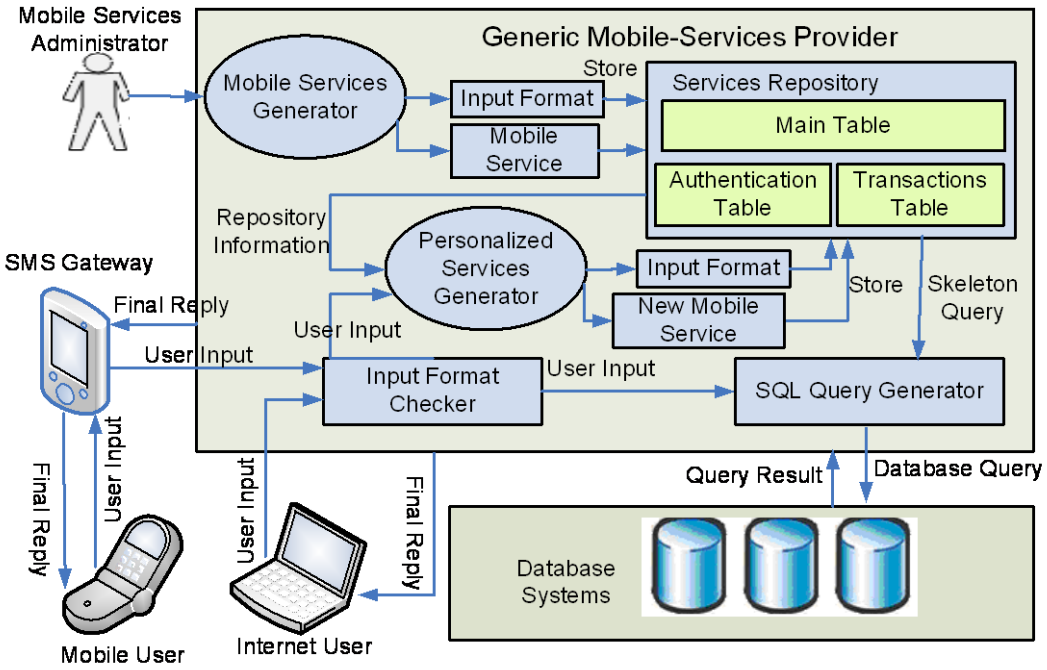

Figure 3 Proposed system architecture

input form is created, and made available through a specific website.

The general format of the user input is given as: *<Service_No> <User-Provided_List>*. Each of the available services provided by the generic mobile-services provider is uniquely identified by *Service_No*. Typically *Service_No* starts from 1 and increases by 1 for every new service added by the mobile-services administrator. *User-Provided_List* contains those input data which user has to provide.

Input Format Checker – Checks if the number and types of user-supplied input exactly matches those of required input parameters for a specific service. It also checks a user request for a personalized service. For personalized service, the user input is forwarded to the Personalized Services Generator.

Personalized Services Generator (PSG) – The PSG is used to generate new personalized services having specific skeleton queries and input format. The PSG modifies the original mobile-service skeleton queries, by assigning static values provided by the user to specific input parameters. The newly personalized service is then stored in the repository.

Services Repository – The MSG stores all the service data, input format for each of the available services into Services repository. The service repository consists of three tables storing various information and skeleton queries of all the available services. For our motivating example, the repository tables after a successful registration are shown in Table 1~3.

In Table 1, the Authentication column identifies whether or not the user authentication is necessary for a specific service. *Trans_Qry* specifies the necessity of transactions like "*SQL insert, SQL update, SQL delete*". *Simple_Resp* shows whether the final reply is a simple string message or result of the some query execution. In our said example, the final reply is an SQL Select query. So the *Final_Reply* column contains a skeleton query instead of a simple string message. The *DB_Name* column is used for the dynamic link to the concerned database and the corresponding queries processing.

The authentication table stores those SQL-Select queries which are used for authentication or specific condition checking. Transactions table stores skeleton queries for insertion, deletion and updating.

After adding a new service, the system administrator must convey the input format to all users along with the detailed description of input formats. The system administrator may use website or TV advertisement. The user can also send a query SMS to the server application, asking for the format of the SMS. The server application then replies with an SMS containing the exact format.

SQL Query Generator (SQG) – Translates skeleton queries into the actual SQL queries by replacing each holes/gaps in the skeleton query with proper user input. The SQL Query Generator takes user-input, skeleton query as an input, and generates the corresponding final SQL query as output.

For the motivating example, after the user supplied inputs "MobileService#1 saleem greatwazir ansan suwon 3/10/2010**"**, the corresponding results back to the user are given below.

Seat_no = 3
Travelling_Date = 3/10/2010
Starting_Time = 11:00
PlateForm_No = 7
Fare_Amount = 4500 KW
Total_Distance = 30 Km
Approx_Time = 45 Min
Available_Amount = 10000

7 User Input Processing

For a specific service, the user supplied input is first checked by input format checker for the required number of supplied inputs. Then it is forwarded to the SQL query generator, making the final SQL query using the skeleton query and input data. The successful execution of required service queries results in a reply to the user. The user input processing is shown in the flow chart given in Figure 4.

Table 1 Main service records

Service_No	DB_Name	Input_Format	Args	Final_Reply	Authenticaon	Trans_Qry	Simp_Resp
1	BusTicket_Info	@ * 1 * Passenger_id * Passenger_Psw d * Starting_Termi nal * Destination_Te rminal * Traveling_Dat e * Bus_Type * From_Time * To_time	PId, Paswd, StrtTerm, DestTerm, TravDate, ServType, FromTime, ToTime	Select Seat_no, Travelling_Date, Starting_Time, PlateForm_No, Fare_Amount, Total_Distance, Approx_Time, Available_Amount from Ticket_Details, Passenger_Account Where Passenger_Account.passenger _id = PId and Travelling_Date = Convert(datetime, TravDate) and ticket_status = 'Reserved'	Yes	Yes	No

Table 2 Authentication

Service_No	Skeleton_Qry	Failure_Message
1	Select *from Passenger_Account where Passenger_Id = PId and passenger_Pswd = Paswd	Authentication Fail
1	Select *from Passenger_Account, Ticket_Details where Passenger_Account.Passenger_Id = PId and Passenger_Account.Passenger_Pswd = Paswd and Ticket_Details.Starting_Terminal = StrtTerm and Ticket_Details.Destination_Terminal = DestTerm and Ticket_Details.Travelling_Date = Convert(datetime, TravDate) and Convert(int,Passenger_Account.Available_Amount) > = Convert(int,Ticket_Details.Fare_Amount)	The available amount is not sufficient to pay the current bill

Table 3. Transactions

Service_No	Skeleton_Qry
1	Insert into Pass_TravelHistory values (PId, StrtTerm, DestTerm, TravDate)
1	Update Top(1) Ticket_Details set Ticket_Status = 'Reserved', Passenger_Id = PId where Ticket_Details.Ticket_Status = 'Available' and convert(int,Ticket_Details.Seat_No) = (Select TOP 1 Ticket_Details.Seat_NO from Ticket_Details where Ticket_Details.Starting_Terminal = StrtTerm and Ticket_Details.Destination_Terminal = DestTerm and Ticket_Details.Travelling_Date = Convert(datetime, TravDate) and Ticket_Details.Ticket_Status = 'Available') order by Starting_Time
1	Update Passenger_Account set Passenger_Account.Available_Amount = Passenger_Account.Available_amount - Convert (int,(Select top 1 Ticket_Details.Fare_Amount from Ticket_Details where Ticket_Details.Passenger_Id = PId and Ticket_Details.Starting_Terminal = StrtTerm and Ticket_Details.Destination_Terminal = DestTerm and Ticket_Details.Travelling_Date = Convert(datetime, TravDate) and Ticket_Details.Ticket_Status = 'Reserved')) Where Passenger_Account.Passenger_Id = PId

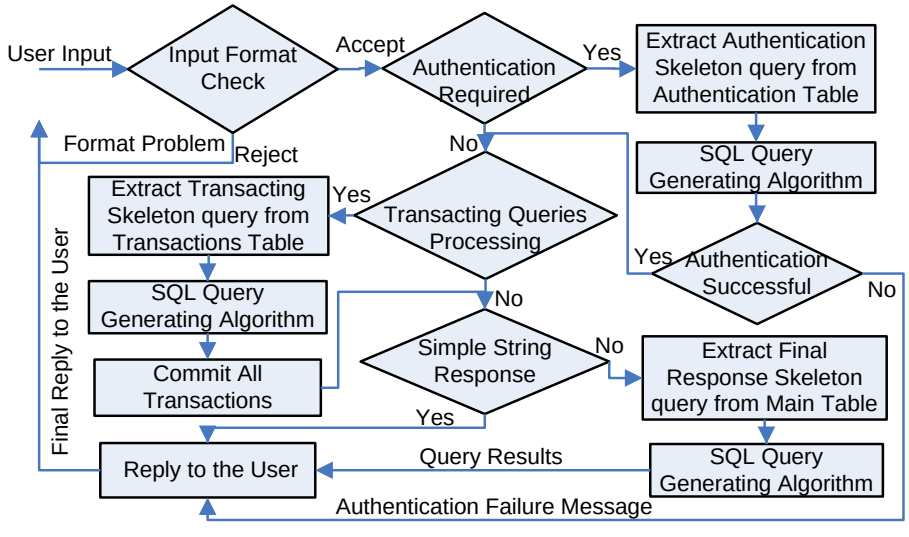

Figure 4 Flow chart of user input processing

After format verification, the user input is checked for certain authentication, if required. For each service, the Authentication column of the Main table stores value either, "*Yes*" or "*No*". If the column value is "Yes", then all the concerned skeleton queries stored in the authentication table are extracted and passed individually through SQL Query Generator for final query generation. If a specific query execution does not satisfy the required condition, the corresponding failure message stored in the Authentication table is forwarded back and the user input processing is stopped. After successful authentication, the *Trans_Qry* column of the Main table is checked. If its value is "*Yes*", all the corresponding transacting skeleton queries ("*SQL insert, update, and delete*") are passed through SQG separately for final query generation. After the successful completion of all transactions, the *Simp_Resp* column of the *Main* table is checked. If its value is "*Yes*", the simple string message stored in the *Final_Reply* coulumn is forwarded back to the user. For the value "*No*", the skeleton

query stored in *Final_Reply* column is passed through SQG and corresponding results are forwarded back to the user.

8 Case Study

Consider the ERD of Figure 5, with information about the students' grades of different exams. Suppose that the administrator wants to add a new service *Exam Result Checking* in which a student provides her/his student identification number, password, and exam name as input and gets the corresponding grade in response. The initial step-by-step requirements set up by the system administrator would be as follows:

1. Every user must be authenticated before getting the result. The authentication should be based on St_id and St_pswd.
2. Every user must provide her/his student identification number, password, and exam name in a proper input format.
3. After proper authentication, the response to the student will be subject code, marks obtained, corresponding grade, and GPA.
4. Once a student checks his result, the Checking_History table must be updated. The history lets the administrator know how many students are still to check the result. For every student identification number and exam name, the Is_Result_Checked field initially contains a default value "No". When a student checks his/her result, the value should be updated from "No" to "Yes" with corresponding date and time entry.

Once the administrator builds the service successfully using MSG, the format of the input is like: <Service_id> <St_id> <St_pswd> <Exam_name> where <Service_id> is a unique service identifier for the grade checking service. The MSG will generate three Skelton queries to be executed against the given database to fulfill the requirements. The first query authenticates a specific student. If the authentication is successful, the next query updates the *Student_Checking_History* table, and then the final query collects the requested grade and sends back to the student. The *Authentication*, *Transactions* and *Main* table stores the first, second and third query, respectively. The corresponding mobile service is given as follows:

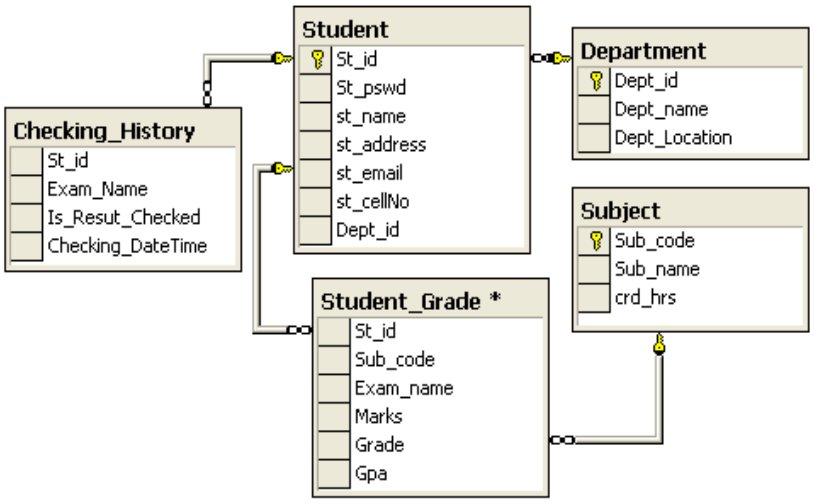

Figure 5 ERD of exam

```
MobileService#3
    Input parameters
        StId: string
        Paswd: string
        ExmName: string
{
  --- Skeleton SQL-Select query (For
Authentication)---
Select *from Student where Student.St_id
= StId and  Student.St_pswd = Paswd;

 --- Skeleton SQL-Update query (To
Update History) ---
Update  Student_Checking_History  set
Is_Resut_Checked     =     'yes'   and
Checking_DataTime = GetDate() where
Student_Checking_History.St_id = StId
and Student_Checking_History.Exam_Name
= ExmName ;

    --- where GetDate() is an SQL
function for the current  system date and
time. The SQGS system provides a list of
SQL functions to be used in a specific
skeleton query.---

---Skeleton SQL-Select query (for final
grades collection) ---
  Select Student_Grade.Sub_code,
Student_Grade.Marks,Student_Grade.Grad
e, Student_Grade.Gpa from Student,
Student_Grade where Student.St_id =StId
and Student_Grade.Exam_name = ExmName
and Student.St_id = Student_Grade.St_id
}
```

8.1 User Input and Corresponding Final Queries

For example, when a user provides the input: *MobileService#3 2008553025 jam342s*, the final queries are given as under:

```
Select *from Student where
Student.St_id = '2008553025'
and Student.St_pswd = 'jam342s'

Update
Student_Checking_History set
Is_Resut_Checked  = 'yes' and
Checking_DataTime = GetDate()
where
Student_Checking_History.St_id
= '2008553025'and
Student_Checking_History.Exam_
Name  = 'fall09'
```

```
Select Student_Grade.Sub_code,
Student_Grade.Marks,
Student_Grade.Grade, Student_Grade.Gpa
from Student, Student_Grade where
Student.St_id  = '2008553025'and
Student_Grade.Exam_name  = 'fall09' and
Student.St_id = Student_Grade.St_id
```

8. 2 Results

Sub_code = cse1
Marks = 98
grade = A
GPA = 4.5
and
Sub_code = cse2
Marks = 94
grade = A0
GPA = 4
and
Sub_code = cse3
Marks = 74
grade = C
GPA = 3

9 Conclusion and Future Work

In this article, we proposed a method of skeleton SQL queries formulization to provide generic information and transaction services. The proposed architecture is fast, easy to understand and scalable. Also the user is able to personalize a main service according to his/her requirements. With service personalization, the numbers of user-inputs are reduced. Consequently, the processing time is increased and only user specific records can be retrieved. Since this system can also provide transaction services due to which only the administrator is allowed to formulate various queries. Also, it is hard for a non-database expert user to formulate complex SQL queries using only the schema information.

As a future work, we are planning to propose a skeleton SPARQL queries formulation language to provide generic resource description framework (RDF) services. The SPARQL will be used as query language and RDF datasets will be used as data source at the endpoints. Moreover, as the proposed system generates services in which all types of database operations are possible, proper authentications and secure database transactions are a must. The existing ID-and-password-based security and authentication is not enough. If the user ID and password are compromised in some way, then any user can do various transactions using someone else's account. We also plan to use proper public and private key management and input encryption to ensure the proper authentication and secure transaction.

Acknowledgment

This work was sponsored by 'Higher Education Commission (HEC), Govt. Of Pakistan' under the scholarship program titled: MS Level Training in Korean Universities/Industry.

References

[1] Saleem, Muhammad, and Kyung-Goo Doh. "Generic Information System Using SMS Gateway." Computer Sciences and Convergence Information Technology, 2009. ICCIT'09. Fourth International Conference on. pp.861-866, 2009.

[2] Saleem, Muhammad, Ali Zahir, Yasir Ismail, and Bilal Saeed. "Enhanced generic information services using mobile messaging." In Advances in Grid and Pervasive Computing, pp. 510-521. Springer Berlin Heidelberg, 2010.

[3] Rohiza Ahmad, Sameem Abdul-Kareem, Free-Form Query for Cell Phones, J. of World Academy of Science, Engineering and Technology, Issue 59, 2009.

[4] Zahi Jarir, Mohamed Quafafou, Mahammed Erradi, Personalized Web Services for Web Information Extraction, J. of Web Services Practices, Vol. 5, No.1, 2010, pp. 22-31.

[5] Koutrika Georgia, Ioannidis Yannis, Personalization of Queries in Database Systems, Proc. 20th International Conference on Data Engineering (ICDE), 2004, pp.597-608.

[6] Jarir Zahi, Mohammad Erradi, Dynamic Personalization in Component-based Application, Asian Journal of Information Technology, Grace Publications Network, 2004, Vol. 3, No. 9, pp. 796-800.

[7] Ladjel Bellatreche, Arnaud Giacometti, Dominique Laurent, A personalization framework for OLAP queries, Proc. 8th ACM International Workshop on Data Warehousing and OLAP, Bremen, 2005, pp. 9-18.

[8] Irene Garrigos, Jaime Gomez, Cristina Cachero, Modelling Dynamic Personalization in Web Applications, Proc. Interventional Conference on Web Engineering, Spain, 2003, pp.472-475.

[9] Kevin Hung, Yaun-Ting Zhang, Implementation of a WAP-Based Telemedicine System for Patient Monitoring, IEEE Transaction on Information Technology in Biomedicine, Vol 7, June, 2003, pp.101-107.

[10] Tyrone Edwards, Suresh Sankaranarayanan, Intelligent Agent based Hospital Search and Appointment system, Proc. 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, Seoul, South Korea, November, 2009, pp.561–567.

[11] Duklon Stenett, Suresh Sankaranarayanan, Personal Mobile Information System, Proc. 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, Seoul, South Korea, November, 2009, pp.561–567.

[12] Lim Tai Ching, A/P H K Garg, Designing SMS Applications for Public Transport Service System in Singapore, Proc. The 8th International Conference on Communication Systems, Singapore, Vol. 2, 2002, pp. 706–710.

[13] Carl Collins, Amy Grude, Matthew Scholl, Robert Thompson, Txt Bus: Wait Time Information on Demand, Proc. Conference on Human Factors in Computing Systems, New York, USA, 2007, pp. 2049–2054.
[14] Hua Wang, Xiaodi Huang, Goutham Reddy Dodda, Ticket-based Mobile Commerce System and its Implementation, Proc. 2nd International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems, Spain, 2006, pp.119–122.
[15] Md. Subrun Jamil, Fouzia Ashraf Mousumi, Short Messaging Service (SMS) Based m-Banking System in Context of Bangladesh, Proc. 11th International Conference on Computer and Information Technology (ICCIT), Bangladesh, 2008, pp.599–604.
[16] Ke Wan, An SMS-based Sales Reporting System for Fashion-clothes Franchising Company, Proc. Engineering Management Conference (IEMC), Managing Technologically Driven Organizations: The Human Side of Innovation and Change, New York, USA, 2003, pp.330–334.
[17] Mohammad Shirali Shahreza, M-Quiz by SMS, Proc. 6th International Conference on Advance Technologies, Tehran, Iran, 2006, pp.726–729.
[18] Archana Prasad, Sean Olin Blagsvedt, Kentaro Toyama, SMSBlogging: Blog-on-the-street Public Art Project, Proc. 15th International Conference on Multimedia, Germany, 2007, pp.501–504.
[19] Felix-Robinson Aschoff, Jasminko Novak, The Mobile Forum: Real-time Information Exchange in Mobile SMS Communities, Proc. Conference on Human Factors in Computing Systems (CHI), Italy, 2008, pp. 3489–3494.
[20] Quratulain Aziz, Payments through Mobile Phone, Proc. 6th International Conference on emerging Technologies (ICET), Peshawar, Pakistan, November, 2006, pp.50-52.
[21] Web Services Description Language (WSDL) 1.1, URL: http://www.w3.org/TR/wsdl as on Friday, August 17, 2012.
[22] Web Services Business Process Execution Language (WSBPEL),URL: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel as on Friday, August 17, 2012
[23] Web Services Coordination (WS-Coordination), URL: http://docs.oasis-open.org/ws-tx/wscoor/2006/06 as on Friday, August 17, 2012.
[24] Web Services Atomic Transaction URL: http://docs.oasis-open.org/ws-tx/wsat/2006/06 as on Friday, August 17, 2012.
[25] Web Services Business Activity, URL: http://docs.oasis-open.org/ws-tx/wsba/2006/06 as on Friday, August 17, 2012.
[26] Shangguang Wang, Qibo Sun, Hua Zou, Fangchun Yang, Web Service Selection Based on Adaptive Decomposition of Global QoS Constraints in Ubiquitous Environment, J. of Internet Technology, Vol. 12 No. 5, 2011, P.757-768 .
[27] Valliyammai, Thamarai Selvi, Mobile Agent Based Resource Monitoring for Job Submission in Grid with Virtual Database, J. of Internet Technology, Vol. 13 No. 3, 2012, P.445-452.
[28] Rita Kuo, Chang-Kai Hsu, Maiga Chang, Jia-Sheng Heh, A Personalized Webpage Reconstructor Based on Concept Lattice and Association Rules, J. of Internet Technology, Vol. 12 No. 6, 2011, P.1015-1024.

Biographies

Muhammad Saleem has completed his bachelor in Computer Software Engineering from N-W.F.P University of Engineering and Technology, in March 2007 and Master in Computer Science and Engineering from Hanyang University, South Korea in August 2010. His research interests include Semantic web, Web services, Database management, Information retrieval, and Social networks analysis. Currently, he is working as PhD candidate at Digital Enterprise Research Institute (DERI), National University of Ireland, Irealnd.

Iqbal Qasim received the BS degree in Computer Science from Allam Iqbal Open University, Islamabad, Pakistan, in 2005 and Master in Computer Science from National university of Science and Technology (NUST), Islamabad, Pakistan in 2009. His research interests include semantic web, web data mining, information retrieval, multimedia systems and social networks analysis. Currently, he is working as PhD candidate at Knowledge and Data Engineering (KDE) laboratory, Hanyang University, South Korea.

Ata ur Rehman has completed his bachelor in Telecommunication Engineering from N-W.F.P University of Engineering and Technology Pakistan, in March 2007 and Master in Electronics Engineering from Politecnico Di Torino Italy, in September 2010. His research interests include Wireless networks, Internet of things, RFIDs, and Social networks analysis. Currently, he is working as PhD candidate at Politecnico Di Torino Italy.