Generic Query Toolkit: A Query Interface Generator Integrating Data Mining

Lichun Zhu, University of Windsor, Windsor, Ontario, Canada N9B 3P4; E-mail: zhu19@uwindsor.ca
C. I. Ezeife, University of Windsor, Windsor, Ontario, Canada N9B 3P4; E-mail: cezeife@uwindsor.ca
R. D. Kent, University of Windsor, Windsor, Ontario, Canada N9B 3P4; E-mail: rkent@uwindsor.ca

ABSTRACT

Construction of flexible query interfaces constitutes an important part in the design of information systems. Either developers or end-users of information systems can build new queries. This paper presents progress towards a Generic Query Toolkit (GQT) that automates the query interface generation process. GQT consists of a parser and an interpreter for a newly defined Generic Query Script Language, a background query processing unit, a presentation layer service provider and the presentation layer component. Data mining querying features have been integrated into this query language.

Keywords: Business Intelligence, Query Automation, Data Mining

1. INTRODUCTION

The original idea for developing a Generic Query Toolkit (GQT) arose from projects for building data mart and report systems for business clients. In these projects, user requirements (business logic) are constantly changing. There is need to build prototypes quickly to speed up the communication cycle between developer and end users. To meet this requirement we developed a software solution to automate the query interface generation process and, thereby, make the prototyping process more efficient. In this solution, we defined an SQL-like query language called Generic Query Language (GQL). A language parser parses the GQL script and extracts elements for constructing a query interface, such as criteria input fields, display attributes and so on. These elements can be serialized into XML schema and stored in the database. GQT generates the query interface based on this schema and then binds end user inputs to generate sequences of target GQL statements that are processed by an interpreter in order to generate final query results. Lastly, a set of presentation tools renders results to the end user interactively.

The current version of GQL script supports not just SQL statements. We have added flow control, variable declaration and other statements to create a functional script language, and a set of language features to support XML based dataset manipulation and data mining functionalities (Han & Kamber, 2001).

The proposed GQT provides solutions for building queries for end-users quickly. Compared with other commercial solutions, our method is fairly lightweight and it can be widely integrated with software projects of various scales. Indeed, this toolkit has been integrated already with information systems ranging from small, personal desktop Management Information Systems to commercial distributed large data marketing and data warehouse systems, and from fat-client applications to web-based applications.

The rest of the paper is organized as follows. Section 2 summarizes relevant related work. Section 3 presents the design of GQT and GQL, its development and testing environment. Section 4 discusses the integration of data mining querying and algorithms into this toolkit. Section 5 presents conclusions and future work.

2. RELATED WORK

The ultimate goal of GQT is to provide more effective interface solutions to support Business Intelligence (BI). BI software typically includes data warehousing, data mining, analysis, reporting and planning capabilities (Golfarelli, Rizzi & Cella, 2004). Existing commercial BI solutions include BusinessObjects (Business Objects, 2006), Cognos (Cognos, 2006), and Oracle Business Intelligence Suite (Oracle Business Intelligence, 2006). Most current commercial BI tools are rather complicated heavyweight and expensive systems requiring significant time to learn for development and configuration purposes.

Some open source projects related to our work include: the Pentaho Business Intelligence Project (Dixon, 2005), Mondrian OLAP server (OLAP Server, 2006), JPivot project (JPivot Project, 2006), and the Weka Data Mining project (Witten & Frank, 2005).

3. THE DESIGN OF GENERIC QUERY TOOLKIT

3.1 Introduction of the GQL Script

This paper proposes GQL script language for customizing user querying processes to a backend database system. The GQL script language can be used to specify two broad tasks of:

1. User interface definition: Users can describe the data presentation patterns as *Field Attribute* and define the query criteria as *Condition Attribute* with this script.
2. Process or Service specification: Users can control business workflow and invoke various services (e.g., data mining services like Classification, Association Rule using WEKA or other mining algorithms and SQL statements) using this script.

The syntax to define data presentation pattern or display attribute list (*Field Attribute*) is a collection of semi-colon delimited fields enclosed in curly brackets "{}" as :

```
Field Attribute ::=   {Field Name;
                      Field Description;
                      Field Type;
                      Display Attribute [;
                      [Aggregate Attribute];
                      [Key Attribute]] }
```

where "[]" means optional attributes. **Field Name** is unique name of column to be displayed, **Field Description** is the display label of the column, **Field Type** is the SQL data type of the column, **Display Attribute** specifies the default display attribute of the column, which can be SHOW/HIDE, The optional **Aggregate Attribute** specifies the aggregation method like COUNT, MAX, MIN, SUM and AVG and **Key Attribute** indicates whether the column is used for grouping. For example, we can use "{*catalog; Category; STRING; SHOW; ; GROUP*}" to specify the display attributes for column *catalog*. The display label is *Category*. The data type is String. The column can be selected to appear in the SQL-GROUP clause.

To define the query criteria (*Condition Attribute*), we use a collection of semi-colon delimited fields enclosed in angular brackets "<>" as:

Copyright © 2007, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Condition Attribute ::= <Condition Expression;
Condition Description;
Condition Type [;
[Value Domain];
[Required Attribute];
[Default Attribute];
[Hint]] >

Condition Expression is unique name for query criteria, **Condition Description** specifies the display label of the criteria. We can specify "#sequence number" in this field to tell the parser that the current criteria share the same input value with the query criteria as the sequence number. **Condition Type** is the SQL data type of the column. **Value Domain** is specified as comma-delimited *"value|description"* pairs or *"#select value, description from tablename where-clause"* to generate a pick list for input field.

The query process written in GQL script contains a collection of semi-colon delimited GQL statements. Currently, the language consists of (a) eleven types of statements (SQL statement, Declare, Assignment, If-elif-else, While, Exit, Break, Continue, Call, Display and Mine statement), (b) two built-in functions (today, substring) and (c) one built-in object (DataSet). The detailed language specification can be found from the online user guide [http://kent1.galab.uwindsor.ca:8088/gqlview/pages/GQTUserGuide.html].

For example, to construct a GQL script for displaying the tuples of a database table called *t_dace*, would entail replacing columns in the regular select-list with Field Attribute placeholders, replacing criteria elements in the where-list with Condition Attribute placeholders to get the following:

```
select
{id;Item;INTEGER;SHOW;;GROUP},
{mark;Type;STRING;SHOW;;GROUP},
{catalog;Category;STRING;SHOW;;GROUP},
{cdate;Date;DATE;SHOW;;GROUP},
{sum(income) incom;Credit;MONEY;SHOW;SUM},
{sum(outcome) outcom;Debit;MONEY;SHOW;SUM},
{sum((income-outcome)) net;Net;MONEY;SHOW;SUM}
from t_dace
where
<id;Item;INTEGER;#select id,name from t item
        where id between 500 and 999 order by id> and
<note;Description;STRING> and
<mark;Type;STRING;#1> and
<catalog;Category;STRING;#3> and
<cdate;Date;DATE> and
<income*exrate;Credit;MONEY> and
<outcome*exrate;Debit;MONEY>
group by #1, #2, #3, #4
order by #1, #2, #3, #4;
```

Once the above script is submitted, the GQT system would generate a query interface shown as Figure 1, a type of input form that allows the user enter ranges of values for query input criteria of the SQL *where* clause.

After one inputs the query criteria (including conditions and summary groups, also shown in Figure 1) and submits the query, the parser generates the following target statement.

Figure 1. Generated query interface


```
select mark, catalog, sum(income) incom, sum(outcome) outcom,
sum((income-outcome)) net
from t_dace
where id between 501 and 512 and
mark = 'P' and cdate >= '01-01-2006'
group by mark, catalog
order by mark, catalog
```

Please note that those input criteria whose values are empty from the interface form the user filled out, will be removed from the where clause of the final GQL statement. This target query is evaluated to retrieve query results that are presented to the user through *CheckResult* form.

3.2 Architecture of the GQL Toolkit

The java-based architecture has five major components: Metadata repository, GQL Parser, GQL Daemon, GQL Server and GQL Viewer.

1. Meta data repository: Consists of Query Repository table for all pre-defined queries and the Task Queue Repository for submitted query tasks history.
2. GQL Parser: is a two-phase parser engine developed using java based lexical analyzer generator Jflex and java based LALR parser generator Java Cup (Java LARL Parser, 2006). The first-phase parse happens at the time that user generates a new query task. The second-phase parse happens at the time that GQL Daemon executes the submitted task in the background. It performs macro replacement to generate and interpret target GQL statements in sequential order, submits SQL statements to database server .
3. GQL Daemon: acts as the background query-processing unit. It awakes every few seconds to browse the task queue, looking for tasks in waiting status. When a waiting task is detected, the daemon program creates a thread to execute the task. It also performs the house-cleaning task, e.g., removes outdated query instances.
4. GQL Server: The GQL Server module provides service interfaces used by the presentation layer. Therefore, it can be called directly or via web service connection. Two major services currently provided are *Access Service* (for system related services such as user authentication) and *GQL Service* (for providing GQL related services like GQL script parsing, extracting query directory, query submission, query result extraction and query annotation).
5. GQL Viewer and Client Application: The GQL Viewer and Client Application represent the presentation layer of GQT system. There are seven major functions provided by the viewer: user authentication interface, presenting query directory to the user, generating query input form after user selects a query, binding user input into GQL XML schema and calling GQL Server to submit new queries, monitoring the task queue, annotation or task removal, displaying the query result to the screen, and finally reporting, data export and other interactive data analysis support.

Copyright © 2007, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

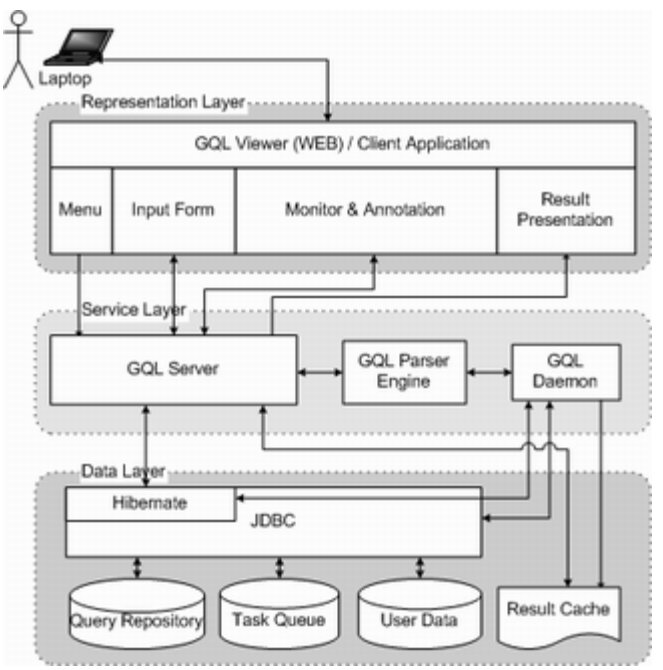
3.3 The GQT Querying Process

One main advantage of the GQL script is that it gives us the flexibility to use one generic set of pre-defined scripts to meet user's multiple requirements through the GQT query interface. The two types of users are: administrators who define and edit GQL scripts and edit existing queries; end users who may know nothing about SQL or GQL and simply use the interface to query the data. To use the system, administrators type in the GQL script and save it in the Query Repository table of the metadata database (in a column with TEXT data type) in advance if it is a new query. After they setup the new query in the database, if users log into GQT testbed system, they see the new query item in the left side menu of Figure 1. After they select the query item, the detailed information about the query is shown in the "View" form, including query name, description and GQL script. Clicking "Next" button switches the page to the "Input" form. The input query items or summary group list inside the Input form are dynamically generated based on the definition of GQL script.

The complete process for GQT querying processing is shown as Figure 2. The querying sequence contains four major steps.

1. Query Interface Generation:
   - User selects a query from menu that is generated from the Query Repository.
   - GQL Server extracts GQL script using GQL Parser Engine to generate the Query Schema, which contains input form elements (XML format).
   - The Input Form module generates input form webpage from the query schema.
2. Query Submission:
   - User inputs criteria from the Input Form, then clicks "Submit" to submit the query.
   - The GQL Viewer analyzes user's input and embeds the input value into the Query Schema, and then sends a *CheckCachedQuery* request to find matches in history queries.
   - If matches are found and user accepts cached result , GQL Server extracts cached result from Result Cache and returns it to Result Presentation module to display the query result. Otherwise, if matches are not found or user selects to ignore cached result, GQL Server creates a new query task in Task Queue using its *ApplyQuery* operator. User's requests are redirected to the Monitor & Annotation web module.
3. Background query processing:
   - GQL Daemon wakes up every five seconds (configurable) to check whether there are newly submitted tasks. If it finds new tasks, it spawns a thread to process the new query task. The status of the task will be turned to "Running".
   - The GQL Daemon thread sets the status of the task to "Success" or "Error".
4. Monitor and result presentation:
   - User refreshes the webpage to check the status of their submitted task.
   - If the status of the task turns to "Success", user can click "View" to view the query result, click "Delete" to remove the query instance, or submit "Change Notes" to append annotation to an existing query instance.

Figure 2. GQT querying process



3.4 The Development and Testing Environment

We have set up a GQT development and test bed environment on a PC environment (CPU: Pentium III, Memory: 256MB, OS: Fedora Core 4) that can be accessed via http://kent1.galab.uwindsor.ca:8088/gqlview. One can use the test user: zlc, password: 9999 to access this system. Because the whole application is developed using java, it can be deployed to various platforms. Currently, the backend database is Informix Dynamic Server 10.

4. INTEGRATING DATA MINING FEATURES INTO GQT

The GQT prototype in an embedded mining approach, integrates data classifier by applying classifier from WEKA data mining toolkit (Weka, 2006), which provides mining algorithms like Classifiers, Clusterers and Association rule miners.

The following example describes how to use WEKA classifier to first analyze the data in the GQT system. Then, the WEKA generated classifier model is exported as a serialized java object, which contains all the details of the algorithm and parameters. In the second stage, the generated model (*.model) file and training data (*.arff) are copied into the template directory of the GQT system. Then, in a GQL script, Mine statement is used to classify the data using pre-generated mining model file for classifying unlabeled data.

This example data is extracted from a cardholder's database that contains cardholder's demographic information and summarized purchase amounts. The original cardholder table contains 7 columns as (CID, SEX, AGE, MONTHIN, PURCHASE, BAD, VIP). Here, SEX, AGE, MONTHIN, PURCHASE and BAD are the predictor attributes while VIP is class attribute. First, we collect training records with VIP column already specified. Next, we load the training data into WEKA explorer, and analyze the data by comparing different classifier algorithms and different sets of parameters. At last, we export the analyzed result as a model file saved in the template directory on server, configurable as a system parameter. In this application, we select the J48 algorithm, an enhanced decision tree classifier that supports both nominal and numeric predictors and the exported model file is "j48.model".

Here, the given **Query Goal** is: Apply J48 classifier to all cardholders having age between 30 and 35 to classify based on their VIP. Next, we use the generated J48. model file to classify the data. The generic mining GQL script for this query is given below, all the non-SQL statements are marked with a prefix "$":

```
1. $declare ads Dataset;
2. select cid, sex, age, month in, purchase, vie, bad from cardholder where
       <cid;ClientId;integer> and
       <vip;VIP;integer;0|False,1|True> and
       <purchase;PurchaseAmt;Money>and
       <age;Age;Integer>
   into temp t1;
3. $set ads = Dataset.readtable('t1');
4. $mine ads classifier using sex, age, month in, purchase, bad class vip_j48
   model 'j48.model';
5. drop table t1;
6. $display ads using
       {cid;ClientId;Integer;SHOW},
       {sex;Gender;String;SHOW},
       {age;Age;Integer;SHOW},
       {monthin;MonthlyIncome;String;SHOW},
       {purchase;PurchaseAmt;Money;SHOW;SUM},
       {vip;VIP;integer;SHOW},
       {bad;Bad;integer;SHOW},
       {vip_j48;VIP_J48;String;SHOW};
```

Copyright © 2007, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

Figure 3. Query result for test classifier

Test Classifier

ClientId	Gender	Age	MonthlyIncome	PurchaseAmt	VIP	Bad	VIP_J48
36	M	30		2642.00			FALSE
37	M	30	3. 2000-4000	4518.51			FALSE
38	F	30		62219.27	TRUE	FALSE	TRUE
47	M	30	5. 8000-16000	1014.00			FALSE
51	M	30	6. 16000-32000	6043.78			FALSE
60	M	30	5. 8000-16000	4859.40			FALSE
73	F	30	4. 4000-8000	10671.81			FALSE
75	M	30	6. 16000-32000	11.70			FALSE
90	F	30		1306.00			FALSE
94	M	30	6. 16000-32000	4646.00			FALSE

After submitting and processing the query, the result is shown in Figure 3. The client with id 38 has a value of TRUE for VIP and FALSE for Bad. The computer generated classification results are shown as column "VIP_J48". The above script uses SQL statement in No.2. to extract the required data from database, then uses Dataset.readtable statement to load the intermediate data into the Dataset object, then uses Mine statement to classify the intermediate data using specified model j48.model (the result of classification algorithm will be stored in vip_j48 column as specified), at last displays the calculated data stored in the dataset to the user.

5. CONCLUSIONS

We present the Generic Query Toolkit as an economical solution for building reporting and data analysis focused applications. Data mining features have been integrated into GQT. We introduced the Generic Query Language to automate the query and display of results. We can integrate easily the user-defined business logic, together with back-end services and front-end presentation modules, to extend the system flexibility.

Future work will include providing OLAP features where user is able to display the data set as a cube, perform slice, drill down, roll up actions interactively is a future addition to the system. Further, we plan to support more data mining features at the query language level, including data clusterer statements and association rule mining statements. Other algorithms, such as those provided by WEKA, or self-designed algorithms like PLWAP tree (Ezeife & Lu, 2005), will be included in the system scope. Another important task is to expand the GQL Server web service, and integrate OGSA-DAI (OGSA-DAI, 2006) data service to support accessing distributed data sources.

REFERENCES

Business Objects Website. 2006. *http://www.businessobjects.com*.

Cognos Website. 2006. http://www.cognos.com.

Dixon, James. 2005. Pentaho Open Source Business Intelligence Platform Technical White Paper. http://sourceforge.net/project/showfiles.php?groupid=140317.

Ezeife, C.I. and Lu, Yi. 2005. Mining Web Log Sequential Patterns with Position Coded Pre-Order Linked WAP-Tree. The International Journal of Data Mining and Knowledge Discovery, Kluwer Academic Publishers, Vol.10, pp.5-38.

Golfarelli, M., Rizzi, S., and Cella, I. 2004. Beyond Data Warehousing: What's next in business intelligence. Proceedings of the 7th International Workshop on Data Warehousing and OLAP Washington DC.

Han, Jiawein and Kamber, Micheline. 2001. Data Mining Concepts and Techniques. Morgan Kaufmann.

Java Cup LALR Parser Generator. 2006. http://www2.cs.tum.edu/projects/cup/.

JPivot Project. 2006. http://jpivot.sourceforge.net/.

Mondrain OLAP Server. 2006. http://mondrian.sourceforge.net/.

Oracle Business Intelligence Suite. 2006. http://www.oracle.com/appserver/business-intelligence/index.html.

The OGSA-DAI Project. 2006. http://www.ogsadai.org.uk/.

W3C Simple Object Access Protocol (SOAP). 2006. http://www.w3.org/TR/soap.

WEKA Data Mining Project. 2006. http://www.cs.waikato.ac.nz/ml/WEKA/.

Witten, Ian H., and Frank, Eibe. 2005. Data Mining: Practical machine learning tools and techniques, 2nd Edition. Morgan Kaufmann, San Francisco.

Copyright © 2007, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

0 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage: www.igi-global.com/proceeding-paper/generic-query-toolkit/33270

Related Content

Critical Realism
Sven A. Carlsson (2009). *Handbook of Research on Contemporary Theoretical Models in Information Systems (pp. 57-76).*
www.irma-international.org/chapter/critical-realism/35824

The Evolution of the ISO/IEC 29110 Set of Standards and Guides
Rory V. O'Connorand Claude Y. Laporte (2017). *International Journal of Information Technologies and Systems Approach (pp. 1-21).*
www.irma-international.org/article/the-evolution-of-the-isoiec-29110-set-of-standards-and-guides/169765

The Role of Social Relationships and Social Networks in IT Project Teams
A.C. Leonardand D.H. Van Zyl (2015). *Encyclopedia of Information Science and Technology, Third Edition (pp. 5325-5333).*
www.irma-international.org/chapter/the-role-of-social-relationships-and-social-networks-in-it-project-teams/112981

Application of Methodology Evaluation System on Current IS Development Methodologies
Alena Buchalcevova (2018). *International Journal of Information Technologies and Systems Approach (pp. 71-87).*
www.irma-international.org/article/application-of-methodology-evaluation-system-on-current-is-development-methodologies/204604

An Approach to Distinguish Between the Severity of Bullying in Messages in Social Media
Geetika Sarnaand M.P.S. Bhatia (2016). *International Journal of Rough Sets and Data Analysis (pp. 1-20).*
www.irma-international.org/article/an-approach-to-distinguish-between-the-severity-of-bullying-in-messages-in-social-media/163100