CLIDE: Interactive Query Formulation for Service Oriented Architectures

Michalis Petropoulos
CSE Department
SUNY Buffalo
mpetropo@cse.buffalo.edu

Alin Deutsch
CSE Department
UC San Diego
deutsch@cs.ucsd.edu

Yannis Papakonstantinou
CSE Department
UC San Diego
yannis@cs.ucsd.edu

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval-*query formulation, selection process*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces-*interaction styles, theory and methods*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services-*web-based services, data sharing*

General Terms: Algorithms, Design, Performance, Theory

Keywords: Interactive Query Formulation, Web Services, Query Rewriting, Query Capabilities

1. INTRODUCTION

Integration systems typically support a restricted set of queries over the schema they export. The reason is that the participating information sources contribute limited content and limited access methods. In prior work, these limited access methods have often been specified using a set of parameterized views exported as Web services, with the understanding that the integration system accepts only queries which have an equivalent rewriting using the views [5, 7, 9, 10]. These queries are called *feasible*. Infeasible queries are rejected without an explanatory feedback.

To help a user, who is building an integration application, avoid a frustrating trial-and-error cycle, we demonstrate the CLIDE query formulation interface, which employes a coloring scheme to guide the user toward formulating feasible queries. CLIDE is based on Microsoft's Query Builder which is incorporated in MS SQL Server [1] and is, in turn, based on the Query-By-Example paradigm [11]. Any good interface that guides the user toward some action must be comprehensive and, at the same time, avoid overloading the user with information at every step [6]. CLIDE achieves both goals since it guarantees that the suggested query edit actions are *complete*, i.e., each feasible query can be built by following only suggestions, *rapidly convergent*, i.e., the suggestions lead to the closest feasible completions of the query, and suitably *summarized*, i.e., at each interaction step, only a minimal number of actions needed to preserve completeness are suggested.

CLIDE consists of a front-end and a back-end that enables the front-end's behavior. We demonstrate the use of CLIDE's front-end and the color-driven interaction using a scenario from service-oriented architectures. The algorithms implemented by the back-end are related to the problem of answering queries using views [4] and are discussed in detail in [8]. CLIDE's implementation is available on-line at `http://www.clide.info`.

SIGMOD'07, June 11–14, 2007, Beijing, China.

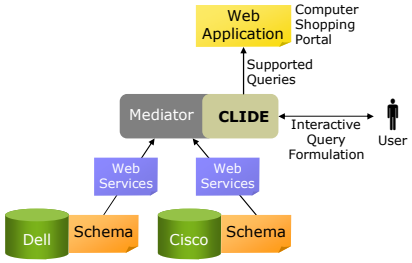

Figure 1: Service-Oriented Architecture

Demonstration Scenario Information systems offer limited access to their data by publishing views, Web services or APIs [2, 7]. Service-oriented architectures [3] aggregate a collection of such services in order to provide more sophisticated Web services and to support Web applications. Figure 1 shows a simple instance of an architecture where the mediator enables a computer shopping portal, such as CNET.com, to have integrated query access to two sources. We assume that Dell and Cisco export a set of Web services on their computer and router catalogs, respectively. Since we want to be able to issue (distributed) queries, we associate schemas with Dell and Cisco and model the Web services as parameterized views over those schemas. Figure 2 illustrates part of their respective schema and the signatures of four Web services they export.

The Dell schema describes computers characterized by their cid, CPU, RAM installed and price, and have a set of network cards installed. Each network card has the cid of the computer it is installed in, accommodates a data rate, implements a standard (e.g., 802.11g) and communicates with a computer via a particular interface (e.g., USB). The Web service *ComByCpu* returns the computers of a given *cpu*. The service *ComNetByCpuRate* provides computers of a given *cpu* that have installed network cards of a given data *rate*. The Cisco source describes routers that also accommodate a data rate, implement standards, have a price and are of a particular type. The *RoutersWired* and *RoutersWireless* services return routers that are of either wired or wireless type respectively.

In Figure 1, a user builds the computer shopping portal by formulating queries against the source schemas, and deploys a mediator in order to execute queries against the exported Web services during run-time. The mediator can answer the query "return all P4 computers with a 54Mbps network card and the compatible wireless routers" by combining the answers of Web service calls *CompNetByCpuRate* and *RoutersWired*. However, it cannot answer the query "return all computers with 1GB of RAM".

```
Computers(cid, cpu, ram, price)                          (Dell Schema)
NetCards(cid, rate, standard, interface)
ComByCpu(cpu) → (Computer)*                                      (V1)
SELECT DISTINCT Com.*
FROM Computers Com
WHERE Com.cpu=cpu
ComNetByCpuRate(cpu, rate) → (Computer, NetCard)*                (V2)
SELECT DISTINCT Com.*, Net.*
FROM Computers Com, NetCards Net
WHERE Com.cid=Net.cid AND Com.cpu=cpu
AND Net.rate=rate

Routers(rate, standard, price, type)                     (Cisco Schema)
RoutersWired() → (Router)*                                       (V3)
SELECT DISTINCT Rou.*
FROM Routers Rou
WHERE Rou.type=''Wired''
RoutersWireless() → (Routers)*                                   (V4)
SELECT DISTINCT Rou.*
FROM Routers Rou
WHERE Rou.type=''Wireless''

(S1.Computers.cid, S1.NetCards.cid)                  (Column Associations)
(S1.NetCards.rate, S2.Routers.rate)
(S1.NetCards.standard, S2.Routers.standard)
```

Figure 2: Source Schemas and Web Services

2. CLIDE'S FRONT-END

Figure 3 shows several snapshots of CLIDE's front-end, where the user graphically formulates a query over the schemas of Figure 2. The top pane of the front-end visualizes a conjunctive SQL query, which is displayed in the bottom pane. CLIDE displays a *table box* for each table alias in the FROM clause. Selections on columns in the WHERE clause are displayed in *selection boxes*. Columns are projected in the SELECT clause using check boxes, called *projection boxes*. Joins are displayed as solid lines connecting the respective column names. The list of tables defined in the input schemas is shown on the left. The user builds conjunctive queries by performing the following visual actions:

1. *Table action*: Drag a table name from the table list and drop it in the top pane. The interface draws a new table box with a fresh table alias, which is added to the FROM clause of the SQL query.
2. *Selection action*: Typing a constant in a selection box results to adding a selection to the WHERE clause.
3. *Join action*: Dragging a column name and dropping it on another one results to a join line connecting the two column names and a new join in the WHERE clause.
4. *Projection action*: Checking a projection box adds a projection to the SELECT clause.

When not all conjunctive queries against a database schema are feasible, CLIDE guides the user toward formulating feasible queries by coloring the possible next actions in a way that indicates what has to be done, what may and what cannot be done. Table actions are suggested by coloring the background of table names in the table list. Selections and projections are suggested by coloring the background of their boxes. Joins are suggested by coloring and dashing join lines. CLIDE employs the following coloring scheme:

- *Yellow* color indicates actions that are necessary for the formulation of a feasible query. For example, conditioning the cpu of Computers will be yellow since all queries that the mediator can answer and involve the Computers table require a cpu.
- *Blue* color indicates a set of actions where at least one of them is required in order to reach one of the next feasible queries. One can choose among many blue options. For example, after the cpu of Computers has been conditioned and a feasible query has been reached, one should condition either the ram or the price (among other choices) in order to reach the next feasible query.
- *White* color indicates selection conditions, tables and projections whose participation in the query is optional.
- *Red* color indicates actions that lead to unsupported queries, regardless of what is included next. For example, conditioning the type column of Routers with a constant other than "Wired" and "Wireless" leads to unsupported queries.

3. DEMONSTRATED INTERACTION

We illustrate the color scheme using the interaction session of Figure 3, which refers to the setting of Figure 2. The user wants to formulate a query that returns computers that meet various selection conditions about network cards and routers - as long as those conditions are supported. Figure 3 shows snapshots of the interaction session, where CLIDE's color scheme suggests, at each interaction step, which actions lead to a feasible query.

Required and Optional Actions Consider the query that the user has formulated in Snapshot 1. The interface indicates that this query is infeasible (see feasibility flag) and that every feasible query that extends it must have a selection on cpu. The latter indication is given by coloring *yellow* the cpu selection box. The rest of the selection boxes and projection boxes are *white* suggesting that these actions are *optional*, i.e., feasible queries can be formulated with or without these actions being performed.

So the user performs the yellow selection on cpu by typing a constant in the selection box. This leads to the feasible query of Snapshot 2. This query is feasible since the mediator can run view V_1 with the parameter instantiated to "P4" and then project out the cid and cpu columns.

Required Choice among Multiple Actions The user may terminate the interaction session and incorporate the query of Snapshot 2 in her application or may continue to extend the query. The interface indicates that, in order to reach a next feasible query, at least one of the NetCards, Routers or (an additional) Computers tables has to be included in the query, among other options. The indication is provided by coloring the corresponding names in the table list *blue*. Each given blue atom, say NetCards, does not appear in all feasible queries that extend the current query. If it did appear in all, then it would be yellow (i.e., required).

Non-Obvious Feasible Queries Snapshot 3 presents a complex case, where the interface's color scheme informs the user about non-obvious feasible queries. After the user introduces a NetCards table, the interface suggests that one of the following extensions to the query is required: The join line between the cid's of Computers and NetCards is suggested since it leads to the formulation of view V_2. It is blue since the user has more options: She can introduce a second copy of Computers, say Computers2, which will lead toward the feasible query that joins Networks with Computers2, selects on rate and takes a Cartesian product with Computers1. If Cartesian product queries are of no interest to the user, she can set an option to have CLIDE ignore them. In such case the cid join would be a required (yellow) extension. For the remainder of the interaction session, we assume that this option is set.

The user has another pair of options at Snapshot 3. She can perform the blue rate selection, which leads to the formulation of view V_2. Alternatively, she may introduce a Routers table and join the rate columns of NetCards and Routers, thus instantiating the rate parameter of V_2 with constants provided by another table.

Selection Options In Snapshot 4, the user has performed the suggested join and introduced a Routers table. Now the Routers.type column needs to be bounded and so the interface colors the corresponding selection box yellow (not shown), as in the case of Snapshot 1. Once the user clicks on the selection box, she is now presented with a drop-down list that explains which constants may be

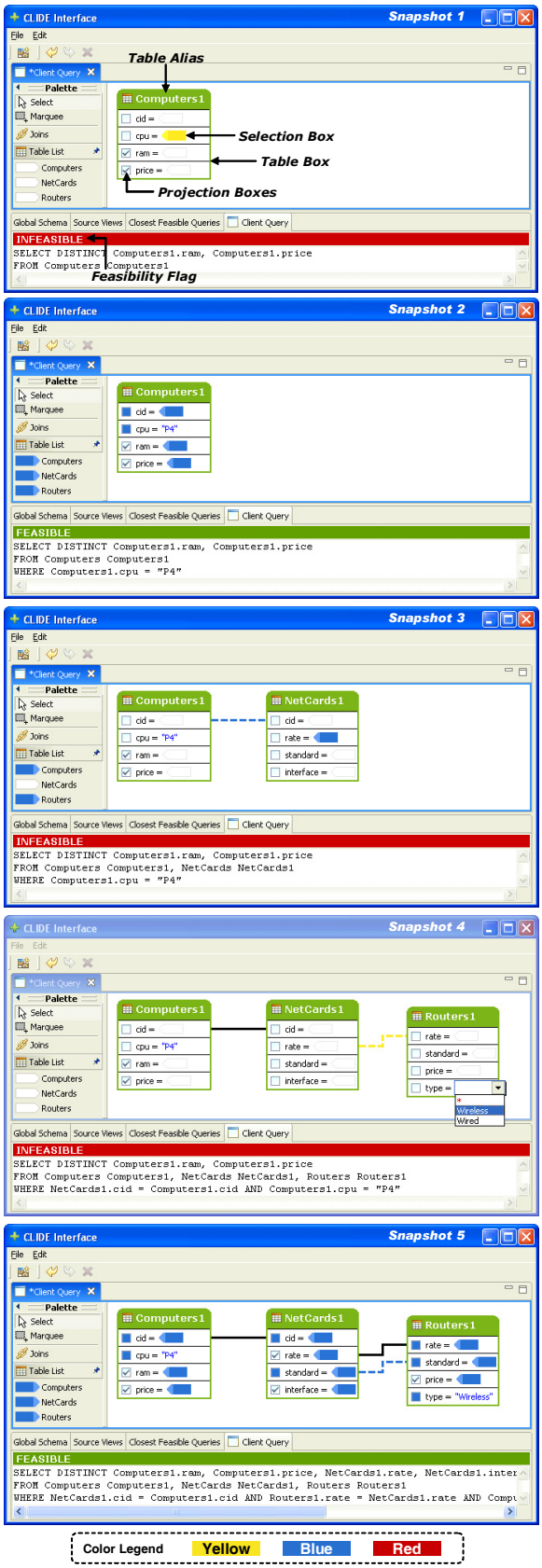

Figure 3: Demonstrated Interaction Session

chosen. She can either choose "Wired" or "Wireless". The symbol $*$ denotes any other constant and is colored *red* to indicate that no feasible query can be formulated if she chooses this option. In the special case where any constant can be chosen, then no drop-down list is shown, as in the case of the `cpu` selection box in Snapshot 1.

In the next steps, the user performs the suggested join, chooses the "Wireless" constant and checks several projection boxes. Snapshot 5 shows the resulting feasible query. The mediator plan that implements this query first accesses view V_4, then for each `rate` returned accesses view V_2 with its parameters instantiated to "P4" and the given `rate`, and finally performs the necessary projections.

The CLIDE front-end displays only yellow and blue join lines. Red and white join lines are typically too many and are not displayed. If the user wants to perform a join other than the ones suggested, she has to follow a trial-and-error procedure.

Note that unchecked projection boxes can be either blue, white or red. A projection box cannot be yellow, because if there is a feasible query that has the corresponding projection atom in the `SELECT` clause, then the query formulated by removing this atom is also feasible.

Finally, if the user performs a red action, then all boxes, lines and items in the table list are colored red, indicating that the user has reached a dead end. No feasible query can be formulated by performing more actions and she has to backtrack, i.e., undo actions.

4. APPLICABILITY

There are many scenarios, other than the demonstrated one, which would benefit from CLIDE's approach to query building. One example is the setting of [10], which is a special case of a service-oriented architecture with parameterized views restricted to identity views over individual tables. Their algorithm infers binding patterns for queries against these views, and could conceptually be used by the user to reach a feasible query by providing appropriate bindings. However, the user queries may be adorned with exponentially many binding patterns, turning the visual inspection by the user into a cumbersome process.

Another obvious CLIDE application is in data privacy enforcement. Recent work [9, 5] allows data owners to identify the non-sensitive data they are willing to export by means of parameterized, virtual views against the proprietary data. Data consumers formulate their queries against the proprietary database as well, but their queries are rejected [9] or return null values [5] if they are not feasible according to the virtual views. Again this leads to a frustrating trial-and-error development process.

5. REFERENCES

[1] Microsoft SQL Server. http://www.microsoft.com/sql/.

[2] Amazon E-Commerce Service. http://www.amazon.com/gp/aws/sdk/.

[3] Web Services and Service-Oriented Architectures. http://www.service-architecture.com/.

[4] A. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[5] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. J. DeWitt. Limiting disclosure in hippocratic databases. In *VLDB*, pages 108–119, 2004.

[6] J. Nielsen. *Designing Web Usability*. New Riders Publishing, 2000.

[7] M. Petropoulos, A. Deutsch, and Y. Papakonstantinou. Query set specification language (qssl). In *WebDB*, pages 99–104, 2003.

[8] M. Petropoulos, A. Deutsch, and Y. Papakonstantinou. Interactive query formulation over web service-accessed sources. In *SIGMOD*, 2006.

[9] S. Rizvi, A. O. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD*, pages 551–562, 2004.

[10] R. Yerneni, C. Li, H. Garcia-Molina, and J. D. Ullman. Computing capabilities of mediators. In *SIGMOD*, pages 443–454, 1999.

[11] M. Zloof. Query by example. *AFIPS NCC*, 44:431–438, 1975.