Quarry: Digging Up the Gems of Your Data Treasury

Petar Jovanovic
Universitat Politècnica de Catalunya, BarcelonaTech
Barcelona, Spain
petar@essi.upc.edu

Oscar Romero
Universitat Politècnica de Catalunya, BarcelonaTech
Barcelona, Spain
oromero@essi.upc.edu

Alkis Simitsis
HP Labs
Palo Alto, CA, USA
alkis@hp.com

Alberto Abelló
Universitat Politècnica de Catalunya, BarcelonaTech
Barcelona, Spain
aabello@essi.upc.edu

Héctor Candón
Universitat Politècnica de Catalunya, BarcelonaTech
Barcelona, Spain
hector.candon@est.fib.upc.edu

Sergi Nadal
Universitat Politècnica de Catalunya, BarcelonaTech
Barcelona, Spain
snadal@essi.upc.edu

ABSTRACT

The design lifecycle of a data warehousing (DW) system is primarily led by requirements of its end-users and the complexity of underlying data sources. The process of designing a multidimensional (MD) schema and back-end extract-transform-load (ETL) processes, is a long-term and mostly manual task. As enterprises shift to more real-time and 'on-the-fly' decision making, business intelligence (BI) systems require automated means for efficiently adapting a physical DW design to frequent changes of business needs. To address this problem, we present *Quarry*, an end-to-end system for assisting users of various technical skills in managing the incremental design and deployment of MD schemata and ETL processes. *Quarry* automates the physical design of a DW system from high-level information requirements. Moreover, *Quarry* provides tools for efficiently accommodating MD schema and ETL process designs to new or changed information needs of its end-users. Finally, Quarry facilitates the deployment of the generated DW design over an extensible list of execution engines. On-site, we will use a variety of examples to show how *Quarry* facilitates the complexity of the DW design lifecycle.

1. INTRODUCTION

Traditionally, the process of designing a multidimensional (MD) schema and back-end extract-transform-load (ETL) flows, is a long-term and mostly manual task. It usually includes several rounds of collecting requirements from end-users, reconciliation, and redesigning until the business needs are finally satisfied. Moreover, in today's BI systems, deployed DW systems, satisfying the current set of requirements is subject to frequent changes as the business evolves. MD schema and ETL process, as other software artifacts, do not lend themselves nicely to evolution events and in general, maintaining them manually is hard. First, for each new, changed, or removed requirement, an updated DW design must go through a series of validation processes to guarantee the *satisfaction* of the current set of requirements, and the *soundness* of the updated design solutions (i.e., meeting MD integrity constraints [9]). Moreover, the proposed design solutions should be further optimized to meet different quality objectives (e.g., performance, fault tolerance, structural complexity). Lastly, complex BI systems may usually involve a plethora of execution platforms, each one specialized for efficiently performing a specific analytical processing. Thus the efficient deployment over different execution systems is an additional challenge.

Translating information requirements into MD schema and ETL process designs has been already studied, and various works propose either manual (e.g., [8]), guided (e.g., [1]) or automated [2, 10, 11] approaches for the design of a DW system. In addition, in [4] a tool (a.k.a. *Clio*) is proposed to automatically generate correspondences (i.e., schema mappings) among different existing schemas, while another tool (a.k.a. *Orchid*) [3] further provides interoperability between *Clio* and procedural ETL tools. However, *Clio* and *Orchid* do not tackle the problem of creating a target schema. Moreover, none of these approaches have dealt with automating the adaptation of a DW design to new information needs of its end-users, or the complete lifecyle of a DW design.

To address these problems, we built *Quarry*, an end-to-end system for assisting users in managing the complexity of the DW design lifecycle.

Quarry starts from high-level information requirements expressed in terms of analytical queries that follow the well-known MD model. That is, having a subject of analysis and its analysis dimensions (e.g., *Analyze the revenue from the last year's sales, per products that are ordered from Spain.*). *Quarry* provides a graphical assistance tool for guiding non-expert users in defining such requirements using a domain-specific vocabulary. Moreover, *Quarry* automates the process of validating each requirement with regard to the MD integrity constraints and its translation into MD schema and ETL process designs (i.e., *partial designs*).

Independently of the way end-users translate their information requirements into the corresponding *partial designs*, *Quarry* provides automated means for integrating these MD schema and ETL process designs into a *unified* DW design satisfying all requirements met so far.

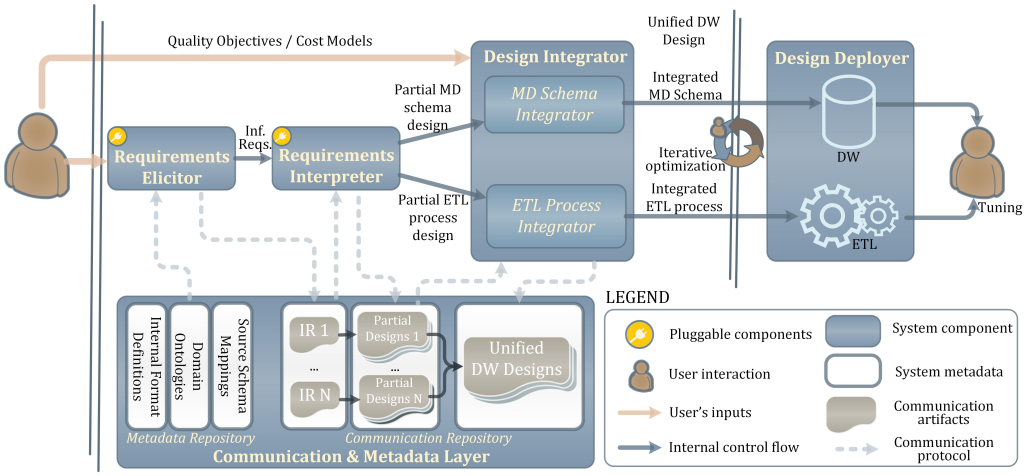

Figure 1: Quarry: system overview

Quarry automates the complex and time-consuming task of the incremental DW design. Moreover, while integrating *partial designs*, *Quarry* provides an automatic validation, both regarding the *soundness* (e.g., meeting MD integrity constraints) and the *satisfiability* of the current business needs. Finally, for leading the automatic integration of MD schema and ETL process designs, and creating an optimal DW design solution, *Quarry* accounts for user-specified quality factors (e.g., *structural design complexity* of an MD schema, *overall execution time* of an ETL process).

Since *Quarry* assists both MD schema and ETL process designs, it also efficiently supports the additional iterative optimization steps of the complete DW design. For example, more complex ETL flows may be required to reduce the complexity of an MD schema and improve the performance of OLAP queries by pre-aggregating and joining source data.

Besides efficiently supporting the traditional DW design, the automation that *Quarry* provides, largely suits the needs of modern BI systems requiring rapid accommodation of a design to satisfy frequent changes.

Outline. We first provide an overview of *Quarry* and then, we present its core features to be demonstrated. Lastly, we outline our on-site presentation.

2. DEMONSTRABLE FEATURES

Quarry presents an end-to-end system for managing the DW design lifecycle. Thus, it comprises four main components (see Figure 1): *Requirements Elicitor*, *Requirements Interpreter*, *Design Integrator*, and *Design Deployer*.

For supporting non-expert users in providing their information requirements at input, *Quarry* provides a graphical component, namely *Requirements Elicitor* (see Figure 2). *Requirements Elicitor* then connects to a component (i.e., *Requirements Interpreter*), which for each information requirement at input semi-automatically generates validated MD schema and ETL process designs (i.e., *partial designs*). *Quarry* further offers a component (i.e., *Design Integrator*) comprising two modules for integrating *partial* MD schema and ETL process designs processed so far, and generating unified design solutions satisfying a complete set of requirements. At each step, after integrating *partial designs* of a new requirement, *Quarry* guarantees the *soundness* of the *unified* design solutions and the *satisfiability* of all requirements processed so far. The produced DW design solutions are further sent to the *Design Deployer* component for the initial deployment of a DW schema and an ETL process that populates it. The deployed design solutions are then available for further user-preferred tunings and use.

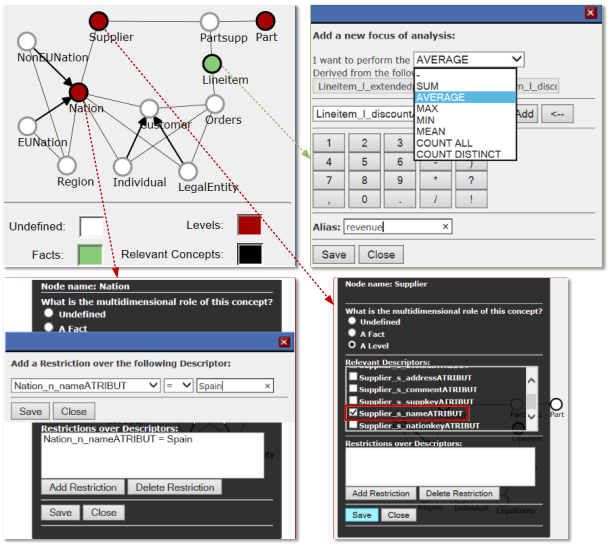

Figure 2: Requirements Elicitor

To support intra and cross-platform communication, *Quarry* uses the *communication & metadata* layer (see Figure 1).

2.1 Requirements Elicitor

Requirements Elicitor uses a graphical representation of a domain ontology capturing the underlying data sources. A domain ontology can be additionally enriched with the business level vocabulary, to enable non-expert users to express their analytical needs. Notice for example a graphical representation of an ontology capturing the TPC-H[1] data sources in top-left part of Figure 2. Apart from manually defining requirements from scratch, *Requirements Elicitor* also offers assistance to end-users' data exploration tasks by analyzing the relationships in the domain ontology, and automatically suggesting potentially interesting analytical perspectives. For example, a user may choose the focus of an anal-

[1] `http://www.tpc.org/tpch/`

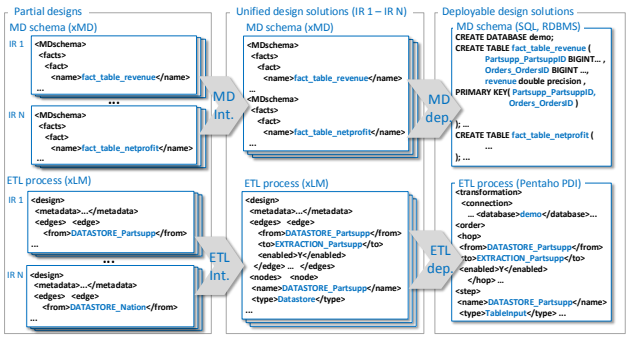

Figure 3: Design integration & deployment example

ysis (e.g., `Lineitem`), while the system then automatically suggests useful dimensions (e.g., `Supplier`, `Nation`, `Part`). The user can further accept or discard the suggestions and supply her information requirement.

2.2 Requirements Interpreter

Each information requirement defined by a user, is then translated by the *Requirements Interpreter* to a *partial* DW design. In particular, *Requirements Interpreter* maps an input information requirement to underlying data sources (i.e., by means of a domain ontology that captures them and corresponding source schema mappings; see Section 2.5), and semi-automatically generates MD schema and ETL process designs that satisfy such requirement. For more details and a discussion on correctness we refer the reader to [11].

In addition, *Quarry* allows plugging in other external design tools, with the assumption that the provided *partial designs* are sound (i.e., meet MD integrity constraints) and that they satisfy an end-user requirement. To enable such cross-platform interoperability, *Quarry* provides logical, platform-independent representations (see Section 2.5). Generated designs are stored to the *Communication & Metadata layer* using corresponding formats and related to the information requirements they satisfy.

2.3 Design Integrator

Starting from each information requirement, translated to corresponding *partial* MD schema and ETL process designs, *Quarry* takes care of incrementally consolidating these designs and generating *unified design solutions* satisfying all current requirements (see Figure 3).

MD Schema Integrator. This module semi-automatically integrates *partial* MD schemas. *MD Schema Integrator*, comprises four stages, namely *matching facts*, *matching dimensions*, *complementing the MD schema design*, and *integration*. The first three stages gradually match different MD concepts and explore new DW design alternatives. The last stage considers these matchings and end-user's feedback to generate the final MD schema that accommodates new information requirements. To boost the integration of new information requirements spanning diverse data sources into the final MD schema design, we capture the semantics (e.g., concepts, properties) of the available data sources in terms of a domain ontology and corresponding source schema mappings (see Section 2.5). *MD Schema Integrator* automatically guarantees MD-compliant results and produces the optimal solution by applying cost models that capture different quality factors (e.g., structural design complexity).

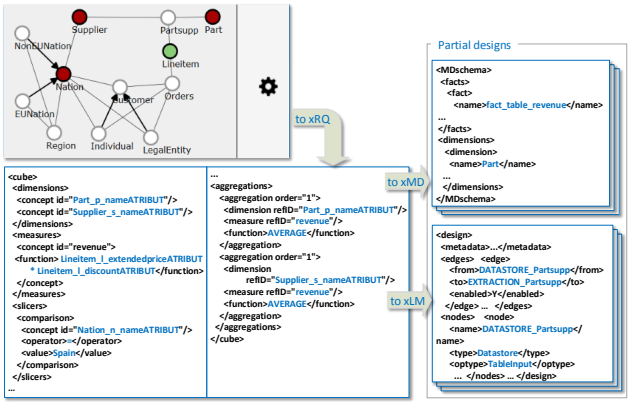

Figure 4: Example design process

ETL Process Integrator. This module processes *partial* ETL designs and incrementally consolidates them into a *unified* ETL design. *ETL Process Integrator*, for each new requirement maximizes the reuse by looking for the largest overlapping of data and operations in the existing ETL process. To boost the reuse of the existing data flow elements when answering new information requirements, *ETL Process Integrator* aligns the order of ETL operations by applying generic equivalence rules. *ETL Process Integrator* also accounts for the cost of produced ETL flows when integrating information requirements, by applying configurable cost models that may consider different quality factors of an ETL process (e.g., overall execution time).

More details, as well as the underlying algorithms of *MD Schema Integrator* can be found in [6] and of *ETL Process Integrator* in [5].

2.4 Design Deployer

Finally, *Quarry* supports the deployment of the unified design solutions over the supported storage repositories and execution platforms (see example in Figure 3). By using platform-independent representations of a DW design (see Section 2.5), *Quarry* is extensible in that it can link to a variety of execution platforms. At the same time, the validated DW designs are available for additional tunings by an expert user (e.g., indexes, materialization level).

2.5 Communication & Metadata Layer

To enable communication inside *Quarry*, the *Communication & Metadata layer* uses logical (XML-based) formats for representing elements that are exchanged among the components. Information requirements are represented in the form of analytical queries using a format called *xRQ*[2] (see bottom-left snippet in Figure 4). An MD schema is represented using the *xMD* format[3] (see top-right snippet in Figure 4), and an ETL process design using the *xLM* format [12] (see bottom-right snippet in Figure 4). Moreover, the *Communication & Metadata layer* offers plug-in capabilities for adding import and export parsers, for supporting various external notations (e.g., SQL, Apache PigLatin, ETL Metadata; see more details in [7]).

Besides providing the communication among different components of the system, the *Communication & Metadata layer*

[2] *xRQ*'s DTD at: `www.essi.upc.edu/~petar/xrq.html`
[3] *xMD*'s DTD at: `www.essi.upc.edu/~petar/xmd.html`

also serves as a repository for the metadata that are produced and used during the DW design lifecycle. The metadata used to boost the semantic-aware integration of DW designs inside the *Quarry* platform, are *domain ontologies* capturing the semantics of underlying data sources, and *source schema mappings* that define the mappings of the ontological concepts in terms of underlying data sources.

2.6 Implementation details

Quarry has been developed at UPC, BarcelonaTech in the last three years, using a service-oriented architecture.

On the client side, *Quarry* provides a web-based component for assisting end-users during the DW lifecycle (i.e., *Requirements Elicitor*). This component is implemented in *JavaScript*, using the specialized *D3* library for visualizing domain ontologies in form of graphs. The rest of modules (i.e., *Requirements Interpreter*, *MD Schema Integrator*, and *ETL Process Integrator*) are deployed on *Apache Tomcat 7.0.34*, with their functionalities offered via HTTP-based RESTful APIs. Such architecture provides the extensibility to *Quarry* for easily plugging and offering new components in the future (e.g., design self-tuning). Currently, all module components are implemented in *Java 1.7*, whilst new modules can internally use different technologies. For generating internal XML formats (i.e., *xRQ*, *xMD*, *xLM*) we created a set of *Apache Velocity 1.7* templates, while for their parsing we rely on the *Java SAX* parser. For representing domain ontology inside *Quarry*, we used *Web Ontology Language* (OWL), and for internally handling the ontology objects inside Java, we used the *Apache Jena* libraries. Lastly, the *Communication & Metadata layer*, which implements communication protocols among different components in *Quarry*, uses a *MongoDB* instance as a storage repository, and a generic XML-JSON-XML parser for reading from and writing to the repository.

3. DEMONSTRATION

In the on-site demonstration, we will present the functionality of *Quarry*, using our end-to-end system for assisting users in managing the DW design lifecycle (see Figure 1). We will use different examples of synthetic and real-world domains, covering a variety of underlying data sources, and a set of representative information requirements from these domains depicting typical scenarios of the DW design lifecycle. Demo participants will be especially encouraged to provide example analytical needs using *Requirements Elicitor*, and play the role of *Quarry*'s end-users. The following scenarios will be covered by our on-site demonstration.

DW design. Business users are not expected to have deep knowledge of the underlying data sources, thus they may choose to pose their information requirements using the domain vocabulary. To this end, business users may use the graphical component of *Quarry* (i.e., *Requirements Elicitor*), and its graphical representation of a domain ontology. This scenario shows how *Quarry* supports non-expert users in the early phases of the DW design lifecycle, to express their analytical needs (i.e., through assisted data exploration of *Requirements Elicitor*), and to easily obtain the initial DW design solutions.

Accommodating a DW design to changes. Due to possible changes in a business environment, a new information requirement could be posed or existing requirements might be changed or even removed from the analysis. Designers thus must reconsider the complete DW design to take into account the incurred changes. This scenario demonstrates how *Quarry* efficiently accommodates these changes and integrate them by producing an optimal DW design solution. We will consider *structural design complexity* as an example quality factor for output MD schemata, and *overall execution time* for ETL processes. The participants will see the benefits of integrated DW design solutions (e.g., reduced overall execution time for integrated ETL processes, executed in Pentaho PDI).

Design deployment. Finally, after the involved parties agree upon the provided solution, the chosen design is deployed on the available execution platforms. In this scenario, we will show how *Quarry* facilitates this part of the design lifecycle and generates corresponding executables for the chosen platforms. We use PostgreSQL for deploying our MD schema solutions, while for running the corresponding ETL flows, we use Pentaho PDI.

4. REFERENCES

[1] Z. E. Akkaoui, E. Zimányi, J.-N. Mazón, and J. Trujillo. A BPMN-Based Design and Maintenance Framework for ETL Processes. *IJDWM*, 9(3):46–72, 2013.

[2] L. Bellatreche, S. Khouri, and N. Berkani. Semantic Data Warehouse Design: From ETL to Deployment à la Carte. In *DASFAA (2)*, pages 64–83, 2013.

[3] S. Dessloch, M. A. Hernández, R. Wisnesky, A. Radwan, and J. Zhou. Orchid: Integrating Schema Mapping and ETL. In *ICDE*, pages 1307–1316, 2008.

[4] R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis. Clio: Schema Mapping Creation and Data Exchange. In *Conceptual Modeling: Foundations and Applications*, pages 198–236. Springer, 2009.

[5] P. Jovanovic, O. Romero, A. Simitsis, and A. Abelló. Integrating ETL processes from information requirements. In *DaWaK*, pages 65–80, 2012.

[6] P. Jovanovic, O. Romero, A. Simitsis, A. Abelló, and D. Mayorova. A requirement-driven approach to the design and evolution of data warehouses. *Inf. Syst.*, 44:94–119, 2014.

[7] P. Jovanovic, A. Simitsis, and K. Wilkinson. Engine independence for logical analytic flows. In *ICDE*, pages 1060–1071, 2014.

[8] R. Kimball, L. Reeves, W. Thornthwaite, and M. Ross. *The Data Warehouse Lifecycle Toolkit*. J. Wiley & Sons, 1998.

[9] J.-N. Mazón, J. Lechtenbörger, and J. Trujillo. A survey on summarizability issues in multidimensional modeling. *Data Knowl. Eng.*, 68(12):1452–1469, 2009.

[10] C. Phipps and K. C. Davis. Automating data warehouse conceptual schema design and evaluation. In *DMDW*, volume 58 of *CEUR Workshop Proceedings*, pages 23–32, 2002.

[11] O. Romero, A. Simitsis, and A. Abelló. GEM: Requirement-Driven Generation of ETL and Multidimensional Conceptual Designs. In *DaWaK*, pages 80–95, 2011.

[12] A. Simitsis and K. Wilkinson. The specification for xLM: an encoding for analytic flows, HP Technical Report, 2015.