Goal-Oriented Conceptual Database Design

Lei Jiang † Thodoros Topaloglou ‡ Alex Borgida § John Mylopoulos †

†Dept. of Computer Science
University of Toronto
{leijiang,jm}@cs.toronto.edu

‡Dept. of Mech. & Ind. Eng.
University of Toronto
thodoros@mie.utoronto.ca

§Dept. of Computer Science
Rutgers University
borgida@cs.rutgers.edu

Abstract

We present details of a goal-oriented process for database requirements analysis. This process consists of a number of steps, spanning the spectrum from high-level stakeholder goal analysis to detailed conceptual schema design. The paper shows how goal modeling contributes to systematic scoping and analysis of the application domain, and subsequent formal specification of database requirements based on this domain analysis. Moreover, a goal-oriented design strategy is proposed to structure the transformation from the domain model to the conceptual schema, according to a set of user defined design issues, also modeled as goals. The proposed process is illustrated step-by-step using a running example from the design of a real-world, industrial biological database. We also report early progress towards building full tool support, by presenting a prototype that captures and stores design sessions in a queryable form. This facility makes it possible to answer questions that are hard, if not impossible, to answer using existing methodologies for database design.

1 Introduction

Standard database development [3, 1, 7] consists of steps for requirements analysis (resulting in a conceptual schema for the data to be stored, using a notation such as ER or UML Class Diagram), logical design (resulting in a relational database schema in SQL, for example), and physical design (resulting in access structures for optimizing certain queries). In contrast, goal-oriented approaches to software development start with an early requirements step that focuses on modeling stakeholders' goals, and deriving from these functional and non-functional requirements through a systematic process [24, 5, 19, 23]. Often-claimed benefits of Goal-Oriented Requirements Engineering (GORE) include systematic exploration of alternatives and traceability of rationale.

In an earlier paper [14], we argued for general benefits of a goal-oriented approach to database requirements analysis, by providing a rational reconstruction of several versions of the conceptual schema for a commercial biological database, and showing how they connected to *alternative* expansions of one stakeholder's goals. Such an approach would capture not only what data means, but also who wants them in the first place, and for what purposes; it therefore provides additional data semantics needed for data integration, machine interpretable web data, and scientific data management.

In this paper, we present *a goal-oriented process for database requirements analysis*. This process starts from multiple stakeholders, captures their goals and plans, analyzes them for alternative data requirements, and ends with a detailed conceptual schema of the data to be stored. The contributions of this paper include:

1. the consideration of multiple stakeholders, and formal capture of their goals, including *data quality goals* [2];
2. the distinction between a *domain model* of the application, and the *conceptual schema* for the database-to-be;
3. a *transformational approach* from the domain model to the conceptual schema, using a sequence of *design operations*, whose templates may be predefined;
4. a set of *design issues* (e.g., persistence, time, accuracy and governance) around which the templates of design operations are grouped, and a *goal-oriented design strategy* for structuring the transformation process, based on a particular prioritization of these issues;
5. a prototype implementation for capturing a particular sequence of design operations, which allows (a) queries to uncover information about the source of / motivation for conceptual schema elements (thus supporting traceability), and (b) roll-back on the design operations (thus supporting exploration of alternatives).

The rest of the paper is organized as follows. We first briefly review related work in Section 2, and provide an overview of the design process in Section 3. The design process is presented in detail with the running example in section 4. The prototype is summarized in Section 5, and

we conclude and point to our research plan in Section 6.

2 Related Work and Terminology

Our modeling of goals, operations and their relationships is influenced by the TROPOS agent-oriented software development methodology [4], which is evolved from the i* framework [24] for modeling and reasoning about organizational environments and their information system requirements. In TROPOS, *Goals* can be *hard* or *soft*: unlike hard goals, a softgoal has neither clear cut definition nor criteria to decide whether it is satisfied, and is usually used to model non-functional requirements. Three basic qualitative goal reasoning techniques are available: (i) *AND/OR-goal decomposition* for refining high-level abstract goals into low-level operational ones; (ii) *means-end analysis* for identifying operations (modeled as plans) to fulfill the refined goals; and (iii) *contribution analysis* for detecting lateral influence on goal fulfillment. The result of a goal analysis is a goal model [10], which is a forest of goal/plan AND-OR trees with contribution edges between nodes of different trees and means-end edges connecting goal and plan nodes. Thanks to the presence of OR-decomposition and means-end edges, there are subsets of leaves in the tree that define alternative ways, i.e., *design alternatives*, to fulfill the aggregate top-level goals.

Although, to our best knowledge, there is no existing GORE framework devoted specifically to database design, some early goal-oriented approaches do model both the dynamic and static aspects of the system. For example, the EKD methodology [5] proposed to model an enterprise in terms of several interconnected sub-models, including a "concepts model". Its main purpose is to define a vocabulary for things and phenomena that can be referred to and reasoned about precisely and consistently by other submodels. In the KAOS framework [8, 23], the goal-oriented approach to object modeling starts with the modeling of goals, and proceeds with the identification of concerned objects, using a simple rule of reference: an object is "concerned" by a goal if it is used in the formal specification of the goal. In most cases, the elements in the static models resemble closely to entities, relationships and attributes of an ER schema; the guidelines and techniques for deriving them from intentional elements are considered as a suitable starting point for our work. However, because of the lack of specificity for database design, many important issues pertinent to data storage, management and access are not addressed at all in these approaches.

Starting with the notion of goals in database design is not a new idea. See, for example, [7], where the first step is to define a mission statement, which describes the high-level purpose of the database. But, the mission statement is only used for scoping the problem space and has no direct influence on the rest of the design steps. In our approach, goals are not only identified, but also formally represented and fully analyzed. Moreover, goals play a central role in the analysis of initial data requirements and the specification of the requirements in terms of a conceptual schema.

Database researchers have proposed various *design strategies* to help structure database design process. For example, [7] defines an intuitive "construct-based" strategy, where the operations are defined and ordered according to the types of modeling constructs (e.g., entities, relationships and etc.). As another example, [1] offers a set of design strategies based on the direction of modeling (i.e., top-down, bottom-up, inside-out and mixed); for instance, in the top-down strategy, a conceptual schema is produced by a series of successive refinement operations, starting from a few highly abstract concepts. On a separate thread, researchers considering the problems of database schema integration (e.g., [17]) have seen the need for a transformation-based approach to database schema manipulation. Such approaches rely on precisely specified *design operations* that transforms an input schema to an output one.

3 Overview of the Design Process

An overview of the proposed design process is shown in Figure 1. It covers both the analysis of initial requirements and the specification of these requirements in terms of a conceptual schema. Goal-oriented requirements analysis starts with a list of stakeholders and their high-level goals, which are refined and interrelated to produce a goal model. The goal model captures not a single, but several alternative sets of data requirements, from which a particular one is chosen to generate the conceptual schema for the database-to-be.

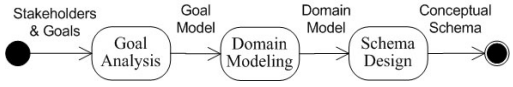

Figure 1: overview of the proposed process

Goal-oriented schema design is further divided into two stages: the modeling of the application domain and the detailed design of the conceptual schema. One of the important contributions of modern RE is the distinction between *environment* and *system-to-be* [11, 13]. In the context of our database design, the analogue is the distinction between the *domain model* and the *conceptual schema*. A domain model describes the necessary understanding of a part of the real world, and facilitates the communication of domain knowledge between developers, end-users and other stakeholders. In this sense, it plays a role similar to the "concepts model" in the EKD framework [5]. A conceptual schema, on the other hand, represents the semantics of the actual data in

the proposed database; its design focuses on issues that are specific to the conceptual content and organization of the data. This view is consistent with earlier work on applying general software development principles and techniques to database design [3], where our domain model and conceptual schema correspond to its analysis and design models respectively. Issues of interest for conceptual schema design mentioned in [3] include persistence, time and units. We are also interested in other issues, such as measurement provenance (i.e., metadata about the process that produces the actual data to be stored), data governance (i.e., issues concerning the usage of stored data such as security and privacy) and various data quality dimensions (e.g., accuracy), which can be (partially) addressed at the schema level and at the design time. One advantage of the separation of domain modeling and schema design is that one would not need to change the domain model when changes are made to governance, provenance and quality requirements.

For example, persistence asks what elements in the domain model need to be maintained by the database-to-be. In a library setting, the domain model is likely to have an entity for the library. But we may not model it explicitly in the conceptual schema, if there is only one library. In such a case, the ternary relationship *borrows(patron, book, library)* in the domain model would also be replaced by a binary relationship *borrows(patron, book)* in the conceptual schema. As another example, in a hospital setting, the domain model may include patients, nurses, medical measurements (e.g., blood pressure) of patients, and even equipment and methods for generating the measurements. But the latter may or may appear in the conceptual schema, depending on the provenance requirements.

Goals play a central role throughout the proposed process. In requirements analysis and domain modeling, hard goals circumscribe the subject matter to be modeled, i.e., to define the universe of discourse (UoD) for the database-to-be, while softgoals are used to select a subset of the leaf-level hard goals and plans as the chosen design alternative. In this sense, goal analysis helps to save domain modeling efforts. In conceptual schema design, softgoals drive the transformation from the domain model to the conceptual schema by guiding the selection and application of proper design operations. This leads to the goal-oriented design strategy that will be discussed in detail in the next section.

4 Detailed Design Process

The proposed design process is divided into eight steps, and illustrated step-by-step with a running example from the design of a real-world, industrial biological database, building on our previous case study [14]. In addition to the running example, each step is described by its input, output, the main tasks to be performed, and a detailed description. This includes available techniques that can be used to fulfill the main task(s), and if a design decision needs to be made, the criteria for making such decision.

In our previous case study [14], we demonstrated the benefits of a goal-oriented approach to database design through the *comparison* of concepts presented in several versions of the conceptual schema produced during its evolution with those derived from alternative expansions of stakeholder goals. In this paper, the focus is on the *derivation process* of the conceptual schema from stakeholder goals. The original biological database is large and only a portion of it is studied in the case study. In this paper, in addition to those already considered in the case study, the design process is also applied to additional data requirements, such as the acquisition of sample data, with special attention to data measurement, ownership and accuracy issues.

Step 1 Identify stakeholder goals, including their quality requirements.

Input: A list of stakeholders. ***Output***: A list of high-level goals of the stakeholders. ***Task(s)***: Goal identification.
Description: In our framework, we assume there are a number of key *roles* with respect to a database, such as data provider, owner, custodian, consumer and regulator. Furthermore, we assume that the list of stakeholders is given, instantiating these roles. Stakeholders express a variety of goals depending on their backgrounds, responsibilities and agendas. The objective of this step is to identify top-level (strategic) goals of each stakeholder, including both hard goals and softgoals. We further distinguish softgoals that concern the data quality (DQ) dimensions [2] from others.
Example: In our example, three main domain stakeholders are given: Genomics Information Sponsor, Chief Scientist, and Pharmaceutical Partner (hereafter called *Sponsor*, *Scientist* and *Partner* respectively). The *Sponsor* plays the role of data provider who is responsible for creating a research tool for subscription by making available a high quality gene expression (GX) reference dataset (*G1*). She is also the decision maker who has the financial responsibility for the project (i.e. controlled budget of the experiments, *S1*). The *Scientist* and *Partner* are application domain experts in molecular biology and drug discovery respectively, whose daily work depends on GX reference datasets. The ultimate goal of the *Scientist* is to obtain a comprehensive understanding the biology of genes (*G3*) and have publishable research results (*S4*), while for the *Partner*, the top priority is to discover and validate drug targets, focusing only on a deep understanding of the properties of disease-specific genes (*G2*). Consequently, the *Partner* wants the GX reference dataset that has deep coverage of diseases (*S2*) and has high biological relevance to drug discovery (*S3*). These goals are shown in Figure 2. Furthermore, the data quality

softgoals shared by all stakeholders include data accuracy (*QS1*), data security (*QS2*), flexible representation (*QS3*) and provenance of measurement data (*QS4*).

Step 2 Generate a goal model.

Input: A list of high-level goals produced in Step 1.
Output: A goal model. ***Task(s)***: Goal analysis.
Description: High level goals give the overall agenda promoted/pursued by stakeholders, but lack details. Goal (AND/OR) decomposition can be used to refine these goals. Systematic methods of decomposing goals have been proposed in the literature. For example, [16] introduced a variability-intensive approach to goal decomposition, based on variability concerns. A variability concern is a question associated with a goal, whose alternative answers lead to alternative refinements of the goal. For softgoals, the NFR framework [6] offers a catalogue of refinement methods, among others, to guide decomposing high-level softgoals into their offspring. Contribution analysis [4] can be used to interrelated goals by identifying positive/negative contributions to the fulfillment of both hard goals and softgoals. As we will see later, in the case of softgoals, "contribution edges" are often used to evaluate design alternatives.
Example: In our example, in order for the *Sponsor* to achieve its top goal *G1* of commercializing a GX reference dataset, she could *either* provide comprehensive GX data (*G1.1*) *or* provide disease-focused GX data (*G1.2*). On the contrary, the *Partner*'s top goal *G2* of drug discovery is AND-decomposed into *G2.1 - G2.3*, meaning that she has to achieve all these subgoals in order to fulfill the top goal. If we continue this process for each stakeholder, we produce a goal model, a portion of which is shown in Figure 2. Going down in the tree, as goals become more specific, we can start recognizing lateral influence between goals through contribution analysis. For example, Goal *G2.1.1* of the *Partner* to subscribe to a disease specific disease collection becomes feasible if the *Sponsor* pursues Goal *G1.2* which aims at providing disease specific GX data. Moreover, the contribution edge from Goal *G2.1.1* to Softgoal *S3* with label "+", indicates that subscription to a disease specific GX collection contributes positively to the fulfillment of *Partner*'s softgoal of drug discovery relevance; similarly, contribution edges labeled with "-" between *S1* (control of budget) and *S2* (coverage of diseases) models the fact that these two softgoals conflict with each other by nature.

Step 3 Select a design alternative.

Input: The goal model obtained in Step 2.
Output: A set of the leaf-level goals in the goal model.
Task(s): Goal evaluation.
Description: As we have discussed, a goal model captures not a single, but several design alternatives, thanks to goal

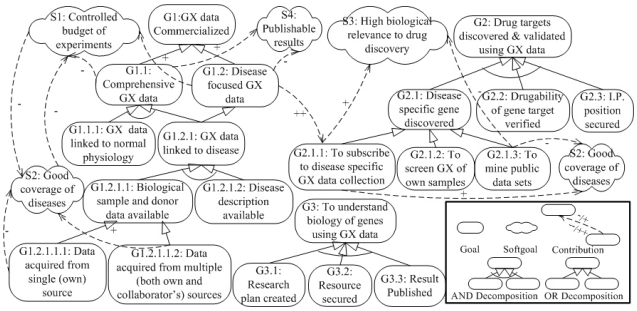

Figure 2: A portion of the goal model

OR-decomposition (i.e., goal level variability) and means-end analysis (i.e., process level variability). In this step, we resolve the first type of variability by selecting a set of leaf-level goals whose collective fulfillment achieves the aggregate top-level goals; we use softgoals as evaluation criteria. Formal goal reasoning techniques are available [10] to help the designer from carrying out this process manually. In particular, in backward reasoning [10], the goal model is analyzed to find out a particular design alternative that guarantees the achievement of the desired top-level goals and softgoals at the minimum cost. A complication in this step is that softgoals may conflict with each other, so that there is usually no uniformly "best solution". Conflicts among softgoals can be resolved by ranking softgoals into a partially ordered list using stakeholders' preferences.
Example: Given the goal model from Step 2, there are total $2 \times 2 \times 3 = 12$ alternative ways to fulfill the top-level goals $\{G1, G2, G3\}$, as shown in Table 1. Assuming a partial ordering of softgoals $\{\text{S2, S3}\} \succ \text{S1} \succ \text{S4}$, and the contribution edges as shown in Figure 2, the following choices can be made for the three variability points: "disease focused (*G1.2*)", "multiple, both (*G1.2.1.1.2*)", "subscription (*G2.1.1*)". This produces the design alternative $DA1_{(Step3)} = \{G1.2.1.1.2, G1.2.1.1, G1.2.1.2, G1.2.1, G1.2, G1\} \cup \{G2.1.1, G2.1, G2.2, G2.3, G2\} \cup \{G3.1, G3.2, G3.3, G3\} \cup \{S1, S2, S3, S4\} \cup \{QS1, QS2, QS3, QS4\}$[1].

Table 1: Design alternatives in the goal model

Variability Points	Number of Choices	Criteria
Coverage of the GX dataset: - Comprehensive (*G1.1*), - Disease focused (*G1.2*)	2	Budget (*S1*), Publishable (*S4*)
Choice of data sources: - Single, own (*G1.2.1.1.1*), - Multiple, both (*G1.2.1.1.2*)	2	Budget (*S1*) Coverage (*S2*)
Method of gene discovery: - Subscription (*G2.1.1*), - Inhouse screening (*G2.1.2*) - Public DB (*G2.1.3*)	3	Coverage (*S2*), Relevance (*S3*)

Step 4 Identify initial set of domain notions from goals.

Input: The goals in the chosen design alternative.
Output: A list of domain notions extracted from these goals.
Task(s): Domain knowledge extraction.

[1] We simply consider all softgoals belong to every design alternative.

Description: In this step, we extract initial data requirements from the goals in the chosen design alternative. The extracted terms are collectively called *domain notions*. The *rule of direct reference*, similar to that in KAOS [8], is used for this task: a term in a goal description is a domain notion if it describes a real-world concept, a relationship linking concepts or an attribute attached to a concept or relationship in the domain.

This rule always has high precision but possibly low recall with respect to the domain model. This is because some relevant domain notions may not appear in any goal description, and can only be discovered through other means (e.g., using domain ontologies). With respect to the final conceptual schema, it always has low precision. But this is not a drawback of our approach; identification of relevant domain notions for persistent storage is achieved partially by considering plans in Step 5 and partially by considering quality goals in Step 8.

Example: The set of domain notions *DNC1(Step4)*, corresponding to the design alternative *DA1(Step3)*, is shown in Table 2[2]. Of course, had a different design alternative been selected, this list would be different.

Table 2: Domain notions extracted from DA1(Step3)

Goals	Domain Notions
G1	gene, gene expression
G1.2	disease
G1.2.1	linked(gene expression, disease)
G1.2.1.1	biological sample, donor
G1.2.1.2	
G1.2.1.1.2	sample source, collaborator

As domain notions represent potential application data requirements, their identification helps to further refine some of the data quality softgoals. For example, the abstract data accuracy softgoal *QS1* can be refined into, among others, accuracy of disease information *QS1.1* (if it is indeed a requirement from the user) immediately after the extraction of the domain notion *disease*.

Step 5 Identify and select plans.

Input: The goals in the chosen design alternative.
Output: A set of plans that collectively fulfill these goals.
Task(s): Goal operationalization, plan evaluation.
Description: So far, the goals in the chosen design alternative are not actionable. The process of gradually analyzing goals to identify operational specifications is often referred to as goal operationalization [8, 19]. Both logic-based [9, 15] and heuristic-based [19, 18] approaches have been proposed for this task. The former is based on a set of pre-defined, formally proved refinement patterns, while the latter relies on a "round-trip" between goal identification and scenario authoring. Unlike other approaches, our emphasis is on identifying *alternative* plans to fulfill a goal, using the means-end analysis [4]. Plan selection are subject to the same evaluation process as the goals. In this step, there is no need to describe each plan in detail, nor formally, but only to a degree that is sufficient for their evaluation.
Example: In our example, at least two well-established procedures can be carried out to acquire biological sample and donor data needed for gene expression analysis: perform animal model study (*P1.1*) and perform human tissue survey (*P1.2*). The first choice studies animal models in a controlled environment, while the second relies on available tissue samples excised from human patients during a surgical operation. An animal model study is considered significantly more costly (*S1*) compared to a human tissue survey since it yields more samples and therefore requires more analysis resources. However, a disease model study gives better coverage of diseases (*S2*) than tissue surveys, because they are easier to manipulate in a controlled environment. With respect to higher biological relevance (*S3*), human data are obviously more valuable than animal data. This portion of the goal model is shown in Figure 3.

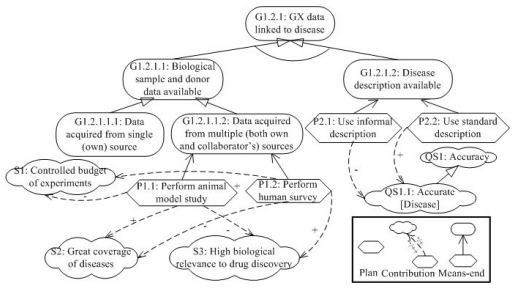

Figure 3: A portion of the goal model enriched with plans

Means-end analysis introduces additional design alternatives. The total number of alternatives we have identified so far is 12 $\times$ 2 (for the type of studies) $\times$ 2 (for choice of disease descriptions) = 48 (See Figure 3). Assuming the same partial ordering of softgoals as before, and the contribution edges shown in Figure 3, *P1.1* and *P2.2* are selected during the evaluation process. This extends the design alternative we obtained in Step 3: *DA1 = DA1(Step3)* $\cup$ $\{$*P1.1*, *P2.2*$\}$. Other design alternatives we will refer to later in the discussion are *DA2 = DA1(Step3)* $\cup$ $\{$*P1.2*, *P2.2*$\}$ and *DA3 = DA1(Step3)* $\cup$ $\{$*P1.1*, *P1.2*, *P2.2*$\}$.

Step 6 Expand the set of domain notions using plans.

Input: The plans in the chosen design alternative.
Output: A list of domain notions extracted from these plans.
Task(s): Process modeling, domain knowledge extraction.
Description: In this step, we expand the list of domain notions using the input plans. We produce for each leaf-level plan a detailed process description, which characterizes the operations in the plan in terms of pre-/post-conditions. Following [20, 8], we consider both inspecting operations (i.e., queries) and modifying operations (i.e., actions). This is

[2]To avoid unnecessary repetition and simplify the discussion, we now focus on the part of the goal model rooted at Goal *G1*.

particular important to database design, since high-level queries can be treated in the same way as actions in requirements analysis. There are two ways in which domain notions are identified: as the information required or produced by these operations, or as the information recorded as the evidence that these operations were carried out with proper attention to qualitative and/or quantitative detail.

Example: Animal model studies (*P1.1*) are normally carried out in the following steps before gene expression analysis: generating a study design (*P1.1.1*), measuring animal subjects (*P1.1.2*), excising (*P1.1.3*) and performing treatments (*P1.1.4*) on animal samples, obtaining planned sample measurements (*P1.1.5*), and performing quality and disease verification tests (*P1.1.6*)[3]. The domain notions that are derived from these sub-plans are listed in Table 3. The final result of this step is the expanded list of domain notions, corresponding to the design alternative *DA1* from Step 5: *DNC1 = DNC1(Step4)* ∪{domain notions extracted from *P1.1* and *P2.2*}. The same process would be carried out for *DA2* (or *DA3*), if it were selected instead of *DA1*.

Table 3: Domain notions extracted from DA1(Step3)

Plans	Domain Notions
P1.1.1	study, study purpose, study design, animal subject;
P1.1.2	animal measurements (e.g., strain, weight, gender)
P1.1.3	organ, organ type, tissue, cell culture
P1.1.4	sample treatment, treatment type, treatment description), disease description
P1.1.5	sample measurement (e.g., weight, dose per unit weight)
P1.1.6	test (test type, test result)

Step 7 Construct the domain model.

Input: The expanded list of domain notions.
Output: A domain model. ***Task(s)***: Domain analysis.
Description: In this step, the domain notions are analyzed and organized into a model of the application domain, using a diagrammatic notation such as UML diagrams (possibly with OCL assertions), or a formal language such as first-order logic. Well-known object modeling techniques can be applied here directly. In a design process, the designer has to add information when moving from earlier stages to later ones. Some of this information may come only from external sources, such as domain ontologies. Sugumaran and Storey have proposed a semi-automatic approach for creating and evaluating conceptual schemas using lightweight domain ontologies [21]. There are two essential tasks in their proposal: determining the relevance between user requirements and ontological terms, and specifying the stop conditions for the automatic propagation of relevance through relationships in the ontology. Explicit goal modeling provides assistance for both tasks.

It is also important to note that, following prior work on requirements engineering [11, 8], the domain model is taken to represent an all-encompassing ("God's eye") view of the *history* of the domain; therefore all domain notions are assumed to have an additional time dimension, which may or may not be explicit represented in the model, based on the modeling language used.

Example: Given the expanded list of domain notions, *DNC1* (from Step 6), a portion of the domain model is sketched in Figure 4, using a UML Class Diagram[4].

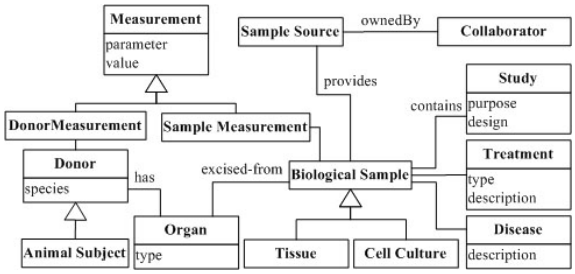

Figure 4: a portion of the domain model for DA1

Step 8 Construct the conceptual schema.

Input: The domain model from Step 7. ***Output***: A conceptual schema. ***Task(s)***: Schema transformation.
Description: In this step, we unify and expand the earlier mentioned use of schema design strategies (e.g., [1]) and schema transformations (e.g., [17]) into a transformational framework. We view schema design as a series of transformation steps; this is true even at the conceptual level, although the transformations are informal, in the sense that human expertise is often required. Part of the design knowledge can be crystallized as design operations and design strategies. A design operation is a template of a design action that transforms an input schema by adding or removing schema elements; a design strategy is defined by a set of available design operations and a partial order in which these operations can be applied.

Both *information-preserving* and *information-changing* (i.e., augmenting and reducing) operations are allowed. One novelty of our approach is that the design strategy itself is viewed as being *goal-oriented*, based on issues such as measurement provenance, data governance and data quality, which are relevant to the particular problem at hand. In particular, these issues are elicited and modeled as softgoals in earlier steps, and operationalized in this step as a set of *design properties*, attached to the elements of the schema. The value of a design property guides the designer in selecting and applying proper design operations to "resolve" the corresponding issue on the schema element.

The proposed goal-oriented design strategy is presented in Table 4 as pseudocode. It consists of three phases: *initialization* (step 1 - 3), *valuation* (step 4) and *resolution*

[3]For the sake of space, we omit the detailed descriptions for these plans here.

[4]For clarity, some of details are omitted in the diagram, such as association, role names and attributes, which are not relevant for the discussion.

(step 5 - 8). Although this process is intended for a database designer, tools can be built to provide guidance in the key tasks (shown in **bold**). For example, the selection and application of design operations can be facilitated by maintaining a catalogue of operations, together with their contributions to the typical valuations of the design properties. We show several examples of such design operations below.

Table 4: Goal-oriented design strategy

Input: a domain model, a set of design issues ***Output***: a conceptual schema where the design issues are "resolved" for every schema element
1. Copy the input domain model as the initial conceptual schema 2. **Prioritize** the input design issues 3. For each design issue, create a corresponding design property for every schema element, with the default "null" value 4. For each schema element, **assign** each of its design properties a non-null value (if possible) and mark it as "unresolved" 5. Find the design issue with highest priority and select a schema element *E* for which the corresponding design property *P* is marked as "unresolved". 6. *If* no such design issue, *then done*; *otherwise if* the value of *P* is "null", *then continue*; *otherwise* **apply** a design operation to resolve *P* on *E* 7. mark *P* as "resolved" for *E* 8. check *P* for other schema elements that are affected by the operation and mark them as "resolved", if so. *Return* to step 5

Example: According the quality softgoals identified in Step 1, there are four design issues to be addressed: data accuracy (*QS1*), data security (*QS2*), flexibility representation (*QS3*) and measurement provenance(*QS4*). We also consider two special issues, persistence (*QS5*) and temporal dimension(*QS6*), which are applicable to any schema design problem. In our example, these issues are prioritized in the following order: (*QS1*, *QS4*, *QS2*, *QS5*, *QS6*, *QS3*). Below we focus on the first five design issues and show examples of design operations that are used to transform the input domain model (from Step 7, Figure 4) into a conceptual schema. The goal is to generate a conceptual schema that is complete (i.e., all information that requires persistent storage is properly modeled in the schema) and pertinent (i.e., the schema only contains elements that are truly relevant to the achievement of goals in the chosen design alternative). Moreover, all the design issues need to be addressed in the schema, and the traceability from goals to schema elements need to be properly maintained.

Addressing accuracy issue. In the simplest case, the accuracy property of a schema element can be assigned a 'yes' or 'no' value, corresponding to an accuracy softgoal. For example, we could assign a 'yes' value to the attribute *disease description*, given Softgoal *QS1.1*. More useful assignment (in terms of providing guidance to the selection of design operations) can be made if the mechanisms to monitor, assess and improve the accuracy of the data is available. One such example appears in [2], where the syntactic accuracy (i.e., no invalid values) is distinguished from the semantic one (i.e., no incorrect values). One way to enhance the syntactic accuracy of an attribute is to refine it (e.g., *address*) into its components (e.g., *city*, *street* and *postcode*); one could argue that each component attribute could be more easily validated, possibly against a public-available dictionary or controlled vocabulary (e.g., a list of valid city names) [2]. In our example, the accuracy value for *disease description* can be refined as {'yes', 'syntactic', 'attribute decomposition'}. Figure 5 shows how a design operation decomposes the attribute *Disease.description* in the domain model into *Disease.name* (i.e., a standardized name for the disease), *Disease.code* (i.e., the corresponding standardized disease code), *Disease.source* (i.e., the standardization source) and *Disease.description* (i.e., an informal disease description). The entity *SNOMED_CT* represents a controlled vocabulary for disease names and codes, derived from the *SNOMED Clinical Terms*, one of the standardized health care terminologies[5].

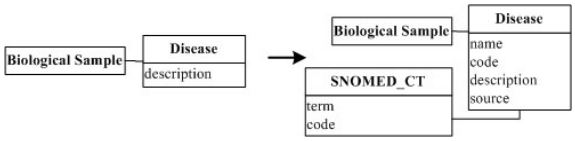

Figure 5: a sample operation addressing accuracy issue

Addressing provenance issue. In the context of e-science, measurement provenance [12] refers to the metadata that describes in-silico experiments, including the purpose, creator and design of the experiments, and the parameters used in the data generation processes. In our example, measurement provenance is an important issue to be addressed. For example, in order to monitor / control the quality of biological samples and subsequently generated gene expression data, the information about the experimental process needs to be recorded as well. This requirement assigns non-null values to the provenance properties to the entities *Tissue* and *Cell Culture*, among others. These values correspond to the process parameters that need to be stored: for *Tissue*, it includes "the amount of time it takes after the tissue sample is excised and before it is frozen"; for *Cell Culture*, it includes "the method used to harvest and isolate the cell". This issue can be resolved by an operation that adds the attributes *time-to-freeze* and *isolation-method* to *Tissue* and *Cell Culture* respectively (see Figure 6).

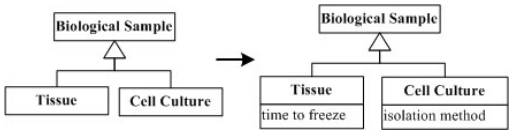

Figure 6: a sample operation addressing provenance issue

Addressing security issue. Data governance concerns additional data requirements that enforce the policies governing the use of application data. Being one type of data governance, the security issue concerns security assurance

[5]http://www.snomed.org

mechanisms which can be enforced at the schema level. In our example, the following schema elements are assigned a non-null value for their security properties: *Biological Sample*, *Tissue*, *Cell Culture*, *Measurement* and *Repeated Observation*. The valuation of these properties are based on the following requirements statement: "A biological sample may be derived from a sample source that is provided by a specific collaborator. In this case, this collaborator owns the biological sample data. The owner of biological sample always has unlimited access to all the data related to the sample, while other collaborator may also have certain access privileges based on mutual agreements." Figure 7 shows how the security issue is addressed for the entity *Biological Sample*, by a design operation that introduces the entity *User Group* and the relationship *access* with the attribute *privilege*.

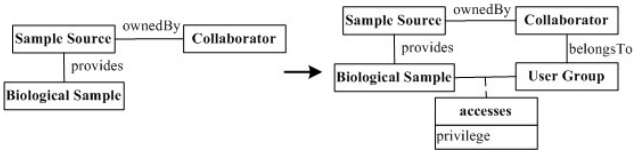

Figure 7: a sample operation addressing security issue

Addressing persistence issue. Persistence concerns whether certain elements could be removed from the schema, without affecting the fulfillment of goals in the chosen design alternative. Every schema element needs to have assigned a non-null ('yes' or 'no') value for its persistence property. It is obvious that all schema elements introduced by previous design operations, e.g., *SNOMED_CT* and *User Group*, are persistent elements. In our example, the list of non-persistent schema elements are: {*Donor Measurement*, *Sample Measurement*, *Donor*, *Organ*, *Study*, *Sample Source*}. The decisions are ultimately determined by the goals and plans in *DA1* (from Step 5). For example, *Donor* is non-persistent since *Animal Subject* is its only sub-entity justifiable in *DA1* (i.e., by *P1.1*). On the other hand, *Collaborator* is assigned a persistent value while *Sample Source* is not. This is because knowing the ownership of samples is sufficient to support the access control to the sample data (a subgoal of *QS2*), and no other goal fulfillment in *DA1* requires the availability of sample source information.

Removing a non-persistent attribute is straightforward. For a non-persistent entity involved in a generalization hierarchy, the well-known transformation rules to flatten the hierarchy can be applied. In other situations, the non-persistent entity is involved in regular relationships, some or all of which may be persistent. Figure 8 shows an design operation that removes the entity *Sample Source*, while promoting the indirect relationship between *Biological Sample* and *Collaborator* to a direct one.

Addressing temporal issue. For each persistent

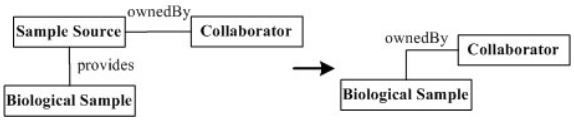

Figure 8: a sample operation addressing persistence issue

element, we need to decide at what time points data needs to be acquired and stored. The domain of the temporal property include following values: {'instantaneous-explicit', 'instantaneous-implicit', 'stable', 'single-explicit', 'single-implicit', 'multiple-periodic', 'multiple-ordered', 'multiple-predefined', 'multiple-user-specified'}. Like persistence, the valuation of the property is largely determined by the stakeholder goals and plans in the chosen design alternative. In our example, the relationship *owned-by* is assigned the temporal value 'single-implicit', reflecting the implicit assumption that ownership of biological samples will never change as far as the stakeholder goals are concerned (so recording the time when the ownership takes effect is unnecessary). On the other hand, the entity *Measurement* is assigned the temporal value 'multiple-periodic', meaning that some measurement data (e.g., survival status, dose per unit weight) need to be collected periodically *and* this historic information needs to be kept in the database.

Here we show one example of a design operation that can be applied to resolve the temporal issue for the entity *Measurement* (Figure 9). The original entity is divided into *Measurement* and *Repeated Observation*, where the former is used to record the general information about the measurement (including the *starting time* and *frequency*) and the latter to keep track of the actual values in a fixed interval (using the attribute *order* to distinguish them).

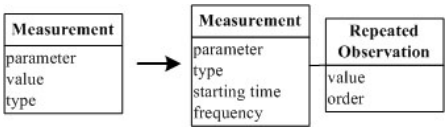

Figure 9: a sample operation addressing temporal issue

The final result of applying the above and other operations is shown in Figure 10[6].

5 Design Environment Prototype

A design process, especially a detailed and complex one, without proper tool support is not likely to be adopted in practice. The two most important benefits we seek in such a tool-supported design environment are (a) partial automation (i.e., to automate the routine part of the design process

[6]It is important to note that (i) the sample operations are used for demonstration purpose; they are by no means the only or optimal solutions, and (ii) the proposed design strategy applies to any design issue that the stakeholders feel relevant to the problem at hand, not just those we show in the example.

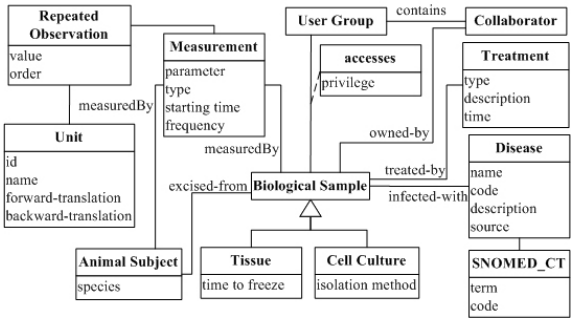

Figure 10: a portion of the conceptual schema for DA1

and provide guidance in certain steps of the process) and (b) book-keeping (i.e., to track traces of changes between steps). In this section, we report early progress towards building full tool support, in the form of a prototype that captures and stores design sessions in a queryable form in a MySQL database. This facility makes it possible to answer questions regarding the properties of the schema elements and design steps, which are hard, if not impossible, to answer using existing methodologies. In addition, roll-back features of the database can be used to explore alternative conceptual schema designs.

A key component of the prototype is a metamodel of the design process. Part of the metamodel is shown in Figure 11 (corresponding to the domain modeling and schema design steps in Figure 1). Based on this metamodel, the design session of the running example (as described in Section 4) is captured and stored in a relational database using MySQL. The questions that can be answered include the general ones, such as (a) what are the main purposes of the database and how does the schema design serve them, and (b) what other alternatives have been considered, and the specific ones, such as (c) how are the accuracy requirements addressed, (d) why is the entity *SNOMED-CT* included, and (e) what are the differences between the design alternatives DA1 and DA2 in terms of the schema elements involved.

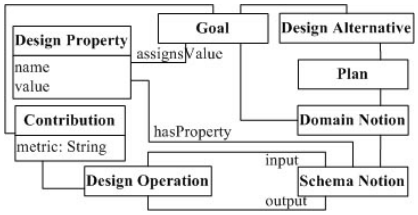

Figure 11: a portion of the metamodel for the prototype

Figure 12 is a screen dump showing how the last two questions above are answered using three SQL queries. The first query returns the softgoal "accuracy of disease information" (*QS1.1*) as the explanation of the presence of the entity *SNOMED-CT* in DA1; the second query lists the schema elements that are shared by design alternatives *DA1* and *DA2*, while the last one lists those that are specific to *DA1*. The ability to answer these and other questions helps to better understand the goal model and the resulting schema, and, in case of changing requirements (e.g., from animal studies (*P1.1*) to human studies (*P1.2*)), to make informed decisions during schema evolution.

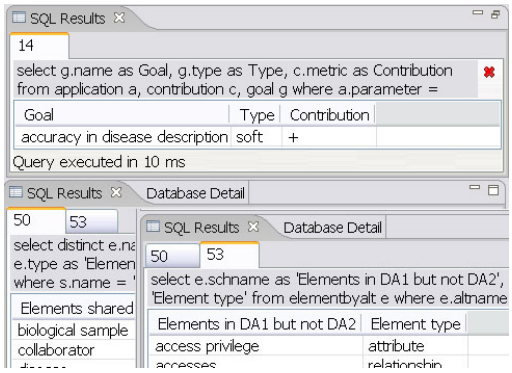

Figure 12: a screen shot of the design environment prototype

6 Conclusion

We have presented a goal-oriented process for database requirements analysis, drawing ideas from existing GORE frameworks. Our goal-oriented approach is advantageous over conventional ones for following reasons. First, it supports systematic exploration of design alternatives: goal models define alternative sets of concepts for fulfilling a goal. Conventional approaches start with a requirement statement that describes one way of solving the problem; therefore, they do not recognize alternatives, and as a result can lead to overweight designs (e.g., including all possible data that *might* be needed). Second, it distinguishes the domain model from the conceptual schema, bearing an analogy to the environment-system distinction, well accepted for software development. Last, it provides a goal-oriented design strategy to compliment the traditional strategies that are based on the types of modeling constructs or the direction of refinement. The proposed process has been applied to the design of a real-world biological database, and a prototype has been constructed as a proof-of-concept and first step towards building a stable design environment.

This work can be extended along several different threads. First, the present approach covers only structural aspects of the conceptual schema; identifying and specifying integrity constraints (e.g., identifiers, cardinalities) is another important part of schema design. Next, for a goal-oriented methodology to become practical, it has to be refined to accommodate both top-down and bottom-up fashions in the elicitation of different models. For GORE, this is consistent with the idea that goal-orientation does not imply a pure top-down strategy, i.e., goals can be identified by refinement using *how* questions and by abstraction using *why* questions [22]. For database design, this is consistent with

the view that the sample queries and data, and forms and reports from the existing system play an important role in requirements analysis [7], just like "scenarios" and "uses-cases" in software design.

Last but not least, in order to better appreciate the benefits, while minimizing the obvious overhead of the goal-oriented approach (e.g., modeling of goals), it is necessary to have a fully functioning design environment that provides support and guideline to the designer throughout the whole design process. As one of its key components, the catalogue of templates of design operations needs to be formally defined based on further experience and study, and at the same time, made extensible so that user-defined operations can be easily integrated.

References

[1] P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone. *Database Systems - Concepts, Languages and Architectures*. McGraw-Hill Book Company, 1999.

[2] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer, 1st edition, 2006.

[3] M. Blaha and W. Premerlani. *Object-oriented modeling and design for database applications*. Prentice-Hall, 1997.

[4] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[5] J. A. Bubenko, D. Brash, and J. Stirna. Ekd user guide. Technical report, Kista, Dept. of Computer and Systems Science, Royal Institute of Technology (KTH) and Stockholm University, Stockholm, Sweden, 1998.

[6] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.

[7] T. M. Connolly and C. E. Begg. *Database Solutions: A step by step guide to building databases*. Addison Wesley, 2003.

[8] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993.

[9] R. Darimont and A. van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering*, pages 179–190, 1996.

[10] P. Giorgini, J. Mylopoulos, and R. Sebastiani. Goal-oriented requirements analysis and reasoning in the tropos methodology. *In Eng. App. of Artifcial Intelligence*, 18/2, 2005.

[11] S. Greenspan, J. Mylopoulos, and A. Borgida. On formal requirements modeling languages: Rml revisited. In *Proceedings of the 16th international conference on Software engineering*, pages 135–147, Los Alamitos, CA, USA, 1994.

[12] M. Greenwood, C. Goble, R. Stevens, J. Zhao, M. Addis, D. Marvin, L. Moreau, and T. Oinn. Provenance of e-science experiments - experience from bioinformatics. In *The UK OST e-Science second All Hands Meeting 2003 (AHM'03), Nottingham, UK*, pages 223–226, 2003.

[13] M. Jackson. The meaning of requirements. *Ann. Softw. Eng.*, 3:5–21, 1997.

[14] L. Jiang, T. Topaloglou, A. Borgida, and J. Mylopoulos. Incorporating goal analysis in database design: A case study from biological data management. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 196–204. IEEE Computer Society, 2006.

[15] E. Letier and A. van Lamsweerde. Deriving operational software specifications from system goals. *SIGSOFT Softw. Eng. Notes*, 27(6):119–128, 2002.

[16] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, and J. Mylopoulos. On goal-based variability acquisition and analysis. In *Proceedings of the 14th IEEE International Conference on Requirements Engineering*, 2006.

[17] A. Poulovassilis and P. M. Brien. A general formal framework for schema transformation. *Data Knowl. Eng.*, 28(1):47–71, 1998.

[18] C. Rolland, G. Grosz, and R. Kla. Experience with goal-scenario coupling in requirements engineering. In *Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, page 74. IEEE Computer Society, 1999.

[19] C. Rolland, C. Souveyet, and C. B. Achour. Guiding goal modeling using scenarios. *IEEE Trans. Softw. Eng.*, 24(12):1055–1071, 1998.

[20] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[21] V. Sugumaran and V. C. Storey. Supporting database designers in entity-relationship modeling: An ontology- based approach. In *the International Conference on Information Systems, ICIS 2003, December 14-17, 2003, Seattle, Washington, USA*, pages 59–71, 2003.

[22] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, page 249. IEEE Computer Society, 2001.

[23] A. van Lamsweerde and E. Letier. From object orientation to goal orientation: A paradigm shift for requirements engineering. In *the Monterey'02 Workshop*, pages 4–8. Springer-Verlag, 2003.

[24] E. S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, page 226. IEEE Computer Society, 1997.