Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Biomedical Informatics 36 (2003) 501–517
Journal of Biomedical Informatics
www.elsevier.com/locate/yjbin

# OQAFMA Querying Agent for the Foundational Model of Anatomy: a prototype for providing flexible and efficient access to large semantic networks

Peter Mork,[a,b,*] James F. Brinkley,[a,b] and Cornelius Rosse[b]

[a] Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195, USA
[b] Structural Informatics Group, Departments of Biological Structure and Medical Education and Biomedical Informatics, University of Washington, Seattle, WA 98195, USA

Received 31 October 2003

**Abstract**

The development of large semantic networks, such as the UMLS, which are intended to support a variety of applications, requires a flexible and efficient query interface for the extraction of information. Using one of the source vocabularies of UMLS as a test bed, we have developed such a prototype query interface. We first identify common classes of queries needed by applications that access these semantic networks. Next, we survey StruQL, an existing query language that we adopted, which supports all of these classes of queries. We then describe the OQAFMA Querying Agent for the Foundational Model of Anatomy (OQAFMA), which provides an efficient implementation of a subset of StruQL by pre-computing a variety of indices. We describe how OQAFMA leverages database optimization by converting StruQL queries to SQL. We evaluate the flexibility and efficiency of our implementation using English queries written by anatomists. This evaluation verifies that OQAFMA provides flexible, efficient access to one such large semantic network, the Foundational Model of Anatomy, and suggests that OQAFMA could be an efficient query interface to other large biomedical knowledge bases, such as the Unified Medical Language System.
© 2003 Elsevier Inc. All rights reserved.

*PACS:* (L01.399); (L01.470, L01.700.508.280); (L01.700.568.810.280); (L01.700.568.810.780); (L01.700.508.300.221); (L01.453.245.945.800)

*Keywords:* Information management; Information storage and retrieval; Database management systems; Programming languages; Databases; Unified medical language system

## 1. Introduction

One of the key successes in artificial intelligence has been the development of expansive knowledge bases. The Unified Medical Language System (UMLS) [1] is one of the largest knowledge bases in existence, containing in its Metathesaurus more than 1.5 million English terms from over 60 source vocabularies [2]. Use of the UMLS is greatly facilitated by its Semantic Network [3], the nodes of which subsume the approximately 775,000 concepts to which the often disparate terms of the diverse source vocabularies refer. For this reason, the most extensive current applications of the UMLS are in clinical information systems for the reconciliation and standardization of terminology.

The Semantic Network (SN), together with the intrinsic hierarchies of the UMLS sources (which in aggregate can be considered the "extended SN") represent extensive knowledge [2,3]. To date, application developers have exploited this knowledge only minimally. The most exciting potential of the extended SN lies in the support it can provide for the development of next-generation, knowledge-based applications that call for machine-based reasoning or inference. There are at least two requirements for realizing this potential: (1) robust, scalable inference engines capable of interfacing with the extended SN and (2) flexible, efficient interfaces that support queries more complex than simple look-up.

* Corresponding author. Fax: 1-206-543-2969.
*E-mail address:* pmork@cs.washington.edu (P. Mork).

1532-0464/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.jbi.2003.11.004

The objective of this paper is to address the second of these requirements by creating a query agent that is both flexible and efficient.

We selected the Foundational Model of Anatomy (FMA) [4] as a test bed for developing a prototype query agent for the following reasons: (1) the FMA is an enhanced version of one of UMLS's largest source vocabularies, the Digital Anatomist; (2) the FMA enhancements include a large number of interrelationships between anatomical concepts that have not yet been incorporated in the Digital Anatomist vocabulary, making the FMA more complex than most other vocabularies in UMLS [4]; and (3) the FMA is implemented as a formal ontology in the Protégé-2000 frame-based knowledge representation system [5], which only supports manual traversal of paths through the knowledge base. Such paths are required for generating results to complex queries not explicitly represented in the ontology.

We believe that the OQAFMA[1] Querying Agent for the Foundational Model of Anatomy can serve as a prototype interface for retrieving answers to complex queries across the entire UMLS by traversing relationships in its extended semantic network. To improve and enhance current methods for querying the UMLS, OQAFMA has to meet three basic requirements: (1) support complex queries; (2) return results in a form readable by both humans and machines; and (3) operate efficiently.

The first requirement is not met by the UMLS's current Knowledge Source Server (KSS), through which most queries are submitted; its set of tools supports only keyword search. There are no mechanisms currently for submitting complex queries like "What are all of the parts of the heart?" or "Which organs are located in the thorax?" Based on an analysis of the classes of queries that need to be supported, we selected for the development of OQAFMA a declarative language called StruQL [6], which was developed at AT&T Labs for website management. StruQL queries provide a more flexible interface than KSS in two respects: first, arbitrary regular expressions can be constructed over the relationships in the network. For example, the parts of the heart can be found by following "part" edges to any depth. Second, multiple conditions can be expressed in a single query. Finding the organs located in the thorax, for example, involves two restrictions; namely "things contained in the thorax" and "things that are organs." Experimental results indicate that more than 80% of the English queries we considered could be expressed using the subset of StruQL we have implemented.

[1] OQAFMA is a recursive acronym in which the O stands for OQAFMA itself, i.e., OQAFMA stands for the OQAFMA Querying Agent for the Foundational Model of Anatomy.

We selected Extensible Markup Language (XML) [7] as an output format for query results to satisfy the second requirement that these results be readable by humans and machines. Our decision was influenced by the fact that XML has been widely adopted as a de facto standard for data exchange. Thus our intent is to provide XML answers to queries posed against a large semantic network, like the FMA, and ultimately the aggregate resources of UMLS.

In order to provide for the third requirement, namely speed of obtaining answers, we implemented two strategies for increasing the efficiency of processing StruQL queries. First, we preprocess the knowledge base and build a collection of indices: one index for each relationship type in the semantic network, and a second index for the transitive closure of each relationship type. The first index allows the system to quickly determine the direct children of a given node and the second provides for the rapid retrieval of all descendants. Second, we convert StruQL queries into SQL. This allows us to benefit from decades of research into query optimization in relational database systems. We implemented a subset of StruQL that can be expressed in SQL using the indices we built. All of the queries we tested completed in less than 1.5 s, including one query involving seven relationships: "What muscle is attached to the coracoid process and humerus?"

Our purpose with this communication is to describe OQAFMA in the context of its function as a server, which already supports complex applications such as a natural language interface [8] and a 3D scene generator [9]. In the next section, we present a classification of queries relevant to application developers. Section 3 provides background for our work through a brief synopsis of the Foundational Model of Anatomy in the context of semantic networks as they relate to regular expressions and StruQL. In Section 4 we describe the system architecture of OQAFMA with an emphasis on the techniques we use to provide efficient access, including index construction and the conversion of StruQL to SQL. Section 5 deals with an evaluation of OQAFMA and in Section 6 we discuss work related to this project, including a variety of XML query languages. In Section 7 we discuss the advantages of the query system we developed and highlight its relevance to the evolving FMA and, in a broader context, to UMLS. We present our conclusions in Section 8.

## 2. Query classification

To facilitate our choice of API for OQAFMA, we first consider the types of queries that need to be supported. Although in terms of their content it is not possible to anticipate the variety of queries submitted to the FMA or UMLS, we found that all queries can be

subsumed by four classes in terms of the processing required for generating the results. This conclusion is based on the fundamental implementation of these knowledge bases, which is a semantic network. As discussed in Section 3.1, a semantic network is a collection of concepts and relationships. In essence, the query classes are determined by the number and heterogeneity of edges that a path launched by the query traverses through the semantic net. We distinguish between three classes of queries: selection, projection, and path queries, and relate the latter to virtual relationships. We have designed OQAFMA to support the gamut of queries included in this classification.

### 2.1. Selection queries

The simplest interaction with a semantic network is to select some or all of the information pertaining to a specific concept. Any application that browses the knowledge base relies heavily on this style of querying. Because they are analogous to selection in a relational database, we refer to these queries as selection queries. A sample selection query might be, "What are the synonyms and direct parts of the heart."

Selection queries are characterized by selection on a single concept, and possibly multiple relationships. Sample applications that rely on this style of querying include:

1. An online browser that displays a given concept and its immediate neighbors;
2. A knowledge acquisition tool (like Protégé-2000 [5]);
3. A forward-/backward-chaining reasoning program (e.g., PROLOG).

### 2.2. Projection queries

The interaction calling for the next level of complexity is to select all of the information for a specific relationship, which corresponds to projecting a column in a relational database. Applications that support data export or transfer use this style of query frequently. One of the more common uses of a projection query is to select all of the children in some hierarchy; for example, "What are all of the parts of the heart (at any depth in the hierarchy)?"

Projection queries and selection queries differ in the depth at which potential answers are found. In the selection example, the only concepts in the result were at a 'distance' of one from the query concept ("heart"); distance being measured by the number of edges between two concepts. In the projection example, concepts in the result can be at any distance from the query concept, but only a single type of relationship can connect the query concept to the result set.

Applications that rely on this style of querying include:

1. An online browser that displays a concept and all of its subclasses;
2. An image retrieval system (e.g., display all images of parts of the heart);
3. Any application that supports data export.

Most current knowledge-base systems support only the two simple classes of selection and projection queries. More sophisticated applications, however, require more sophisticated query capabilities. Support for complex path queries is what distinguishes OQAFMA from alternative approaches.

### 2.3. Path queries

Before defining them, we first illustrate the need for path queries using anatomical examples: an online scene generator has been developed for interactively aggregating 3D graphics models of anatomical structures into larger body parts, simulating the reverse of dissection [9,10]. This application utilizes knowledge represented in the FMA. An exercise calling for aggregating the organs and organ parts that constitute the mediastinal part of the chest first requests from the FMA the names of these structures, which are then used to retrieve the graphics models indexed by these terms. Note that this request involves querying the FMA for concepts that are organs and organ parts involving the "is-a" relationship and the mediastinum using the "containment" and "part" relationships. A second application under development is Emily [11], which can use the FMA to extract the answers to textbook exam questions. A sample question is, "Which muscles form boundaries of the axilla?" This requires identifying the surfaces that constitute a boundary of the axilla (e.g., medial boundary of the axilla), identifying the appropriate set of all muscles, and finally identifying which muscles share a boundary with the axilla (e.g., the serratus anterior shares a boundary with the medial boundary of the axilla). Once again, multiple concepts are referenced ("axilla" and "muscle"), as are multiple relationships ("boundary" and "is-a").

These applications illustrate the nature of *path queries* [12]; they allow the query to reference any number of concepts and any number of relationships. Path queries are a common feature of query languages for both object databases and semi-structured data (examples include [13–15], among others). They provide for constructing complex relationships through concatenation, closure, and alternation. Thus, path queries allow for arbitrary regular expressions to be constructed over the relationships. In fact, selection and projection queries are special cases of this more general classification. Any application that expects the knowledge base to perform more than basic retrieval will require path queries, which, at a higher level, can be thought of as virtual relationships between nodes.

### 2.4. Virtual relationships

Our experience with the FMA illustrates that a semantic network becomes extended with new relationships as the knowledge base evolves. As a result, the complexity of a knowledge base tends to grow over time. Introduction of new relationships should not create problems for previously developed applications as long as relationships on which an application relies are not deleted. However, this assumption can prove to be wrong as illustrated by changes in the simple part hierarchy.

Part relationships are, as a rule, considered to be transitive; the FMA explicitly represents only direct parts (i.e., those connected by a single part relationship). The generic inverse relationships "has part" and "part of" subsume a number of more specific relationships [16]. Accordingly, in a recent version of the FMA, the generic "part" relationship has been further specified to distinguish, among other things, anatomical parts of an organ (e.g., head of the femur) from its arbitrary parts (e.g., proximal part or upper end of the femur) [17]. As a result, the "part" relationship is being split into multiple more specific relationships such as "anatomical part" and "arbitrary part." The "part" relationship as such will eventually disappear altogether from the FMA's implementation. When this happens, any application that retrieves information using the generic, unspecified relationship will break.

Ideally, it should be possible to insulate applications from both changes to the physical data representation (e.g., moving from text files to a relational database) as well as changes to the logical data representation (e.g., migrating "part" to multiple relationships). A well-defined API can be used to guarantee the former. Virtual relationships can be used to facilitate the latter.

A virtual relationship (like a database view) allows one to dynamically populate a relationship using a query. For example, one could define "part" as the union of "anatomical-part" and "arbitrary-part." The power of this approach is limited only by the expressiveness of the query language. This is a persuasive argument for choosing a query interface that accepts path queries; one gains the ability to express virtual relationships using any regular expression. Our aim with the design of OQAFMA was to satisfy this requirement.

One can use virtual relationships to abstract away granularity that is not necessary for a given application. For example, using Emily [11], one might ask, "Which organs are contained in the Thorax?" Since the FMA represents relationships at their most specific and granular level, it does not explicitly store this relationship. Instead, one must ask, "Which organs are directly contained in some part of the Thorax?" The colloquial interpretation of containment corresponds to a virtual relationship, namely the concatenation of "all parts" with "directly contains." Our intent with the development of OQAFMA is to anticipate the needs of diverse application developers and assure that this interface can handle virtual relationships and three major classes of queries.

## 3. Background

Before describing the system architecture of OQAFMA that enables processing of different query classes and virtual relationships, it is desirable to provide some background on regular expressions and StruQL's syntax and semantics. Since we use the Foundational Model of Anatomy as a test bed for developing OQAFMA, we begin by defining the FMA and relate its implementation to semantic networks, which provide the substrate for query processing not only by OQAFMA, but also more generally by any interface.

### 3.1. The Foundational Model of Anatomy

The Foundational Model of Anatomy is an evolving ontology for biomedical informatics; it is concerned with the representation of concepts and relationships necessary for the symbolic modeling of the structure of the human body in a computable form that is also understandable by humans [4]. Its development has been guided by a set of declared principles and a high-level representation scheme, which through their implementation jointly express a theory of anatomy. The model is regarded as *foundational* because (1) anatomy is fundamental to all biomedical domains and (2) the structural concepts and relationships encompassed by the FMA generalize to all these domains. The FMA is intended as a reference, rather than a domain ontology: its purpose is to provide anatomical information for the development of any application that calls for anatomical knowledge, rather than serve the needs of particular user groups.

The backbone of the FMA is an inheritance class subsumption hierarchy (Anatomy Taxonomy or AT) the concepts of which are interlinked by the "is-a" relationship. Currently the AT contains some 67,000 concepts (represented by over 110,000 terms), which refer to anatomical entities ranging in size and complexity from biological macromolecules to cells, tissues, organs and organ systems, and also include spaces and surfaces as well as conceptual entities. These concepts are further interlinked by 1.4 million additional relationships of 147 distinct types. As noted earlier, these concepts and relationships are implemented in the frame-based system of Protégé-2000 [5]. Although this representation system was selected for its expressivity, the underlying data structure is, in essence, a semantic network.

In the simplest terms, a semantic network is a graph-based data model in which the nodes correspond to concepts and the edges to named relationships among these concepts (for a comprehensive description see chapter 6 of [18]). Edges can also link concepts to values, like strings or integers (e.g., names and numerical identifiers associated with anatomical concepts). A semantic network is closely related to a frame-based data model, the precise definition of which varies according to different authors: in [18] frames have procedural attachments, whereas in [19] frames and semantic networks are indistinguishable. Because the FMA is authored using Protégé-2000 [5], which is based on the OKBC standard [20], we adopt the latter's definition.

Citing from the authors of OKBC [21], "Open Knowledge Base Connectivity (OKBC) is an application programming interface (API) for [knowledge representation systems]." The OKBC data model includes: (1) frames, which are named concepts; (2) slots, which are named (binary) relationships used to connect frames to either other frames (e.g., "part" and "boundary"), or to values; and (3) facets, which are tertiary constraints attached to frame/slot pairs. Frames and slots correspond directly to nodes and edges in a semantic network. Although the AT contains approximately 67,000 concepts, its implementation results in some 180,000 frames, which considerably augments the extent and complexity of the semantic network to be navigated by OQAFMA.

Facets have no counterpart in a semantic network. As a result, OQAFMA ignores facets, which are primarily used to guide data entry (for example, by restricting slot values to a particular class of frames, exemplified by the constraining the values for the "branch" slot to AT classes "hollow tree" and "neural tree"). In the FMA, this amounts to discarding 3% of the facts in the knowledge base. Thus regular expressions (defined below) can capture the vast majority of relationships, explicit and implicit, present in the FMA.

### *3.2. Regular expressions*

Although we recognize that regular expressions deserve thorough consideration (see, for example, chapter 2 of [22]) it serves our purpose to define them as concatenation, alternation, and closure operations over some alphabet. Path queries are defined as regular expressions over the edges (or slots) present in the semantic network. Adopting the notation in [22], we briefly describe these operators, displayed graphically in Fig. 1, because they are central to path queries, virtual relationships, and StruQL.

The *concatenation* of two paths (`P1.P2`) generates a new "longer" path, which can be interpreted as the first path, followed by the second path. Formally, this new relationship connects `X` and `Y` whenever there exists some node (`N`) such that `P1` connects `X` and `N` and `P2` connects `N` and `Y`. (In the frame literature this is known as a slot-chain.) For example, in a semantic network corresponding to a family tree, grandparents can be retrieved by concatenating "parent" with itself (i.e., "parent"."parent"). Paths of arbitrary (finite) length can be constructed by concatenating multiple edges together.

The *alternation* of two paths (`P1|P2`) generates a choice between the two paths. Formally, this new relationship connects `X` and `Y` whenever either `P1` connects `X` and `Y` or `P2` connects `X` and `Y`. This operation can also be thought of as disjunction or union. Continuing the family tree example, siblings can be retrieved by alternating "brother" and "sister" (i.e., "brother"|"sister"). Another version of alternation is the *optional* operator (?): `P?` corresponds to (`P` | $\varepsilon$), where $\varepsilon$ is the empty path of length 0.

*Closure* (`P+`) means following `P` an arbitrary number of times. Formally, this relationship connects `X` and `Y` whenever there exists a collection of intermediate nodes such that `P` connects each successive pair of nodes. This constraint is most naturally expressed (as in Fig. 1) as a recursive relationship. This operation is essential to the traversal of hierarchies (for example "part" or "is-a") since it allows traversal to an arbitrary depth. Thus, closure is crucial whenever a relationship exhibits transitivity. To complete the family tree example, ancestors can be retrieved by performing closure on the "parent" relationship (i.e., "parent"+). Because of the frequency with which the optional operator follows the closure operator, these two operators are combined by the star (*) operator. Thus, `P*` means follow `P` any number of times or not at all.

Of these operations, none are supported by Protégé-2000 and only concatenation is supported by OKBC. In a relational database, concatenation is implemented as a natural join and alternation as a union. Closure requires an expensive fixed-point operation that is not supported by many database engines. All of the operations are supported by StruQL.

### *3.3. StruQL*

OQAFMA is based on StruQL [6], a query language that is the ideal choice for querying complex semantic

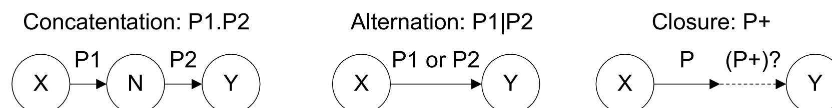


Fig. 1. Regular expression operators—concatenation (path composition), alternation (path union), and closure (recursive path traversal).

networks because it supports path queries and does not require explicit enumeration of join conditions. Moreover, StruQL uses an edge labeled graph as the underlying data structure, which corresponds exactly with a semantic network, and very closely to a frame-based data model. Further justification for choosing StruQL over other query languages is given in Section 6.

A StruQL query consists of several clauses, of which OQAFMA supports two: a WHERE clause, which binds variables to a subset of the semantic network, followed by a CREATE clause, which constructs the result. For example, the query in Fig. 2A selects the node named "Heart," identifies all of the synonyms of the heart and returns those values.

This simple example illustrates the basic constructs in the WHERE clause. First, one can express relationships between two nodes (illustrated in the example) or between a node and a value (e.g., by replacing `N` with the value "Heart") using the -> operator. Second, one can express binary conditions (usually equality: ==) between a variable and an atomic value. Finally, whenever a variable is referenced multiple times, it refers to the same node in all cases. For example, in Fig. 2A, the variable `H` is re-used and always represents a node whose "name" is "Heart." Readers familiar with SQL should note that re-using a variable name supports the equivalent of an equi-join operation.

These basic operations support all possible selection queries, such as those in Fig. 2. Fig. 2B demonstrates how additional constraints can be added using an escape to SQL (which supports pattern matching using LIKE), and Fig. 2C demonstrates how multiple variables can be retrieved using a single query. Note that neither 2B nor 2C can be answered directly using OKBC or Protégé-2000.

Moreover, StruQL allows either edge variables or regular expressions (paths) to be used in place of specific edge names. This allows one to express arbitrarily complex path queries using virtual relationships. One of the early motivations for choosing a language that supports virtual relationships was the way in which containment is modeled in the FMA.

An anatomical structure can only be contained in an anatomical space. Thus, it is valid to say that the left lung is contained in the thoracic cavity. However, it is not valid (in the FMA) to say that the left lung is contained in the thorax. A reasonable user query is "What are all of the organs contained in the thorax?" In the model, the answer is the empty set. If you asked anyone with a passing knowledge of anatomy this question, they would be able to list several organs. In terms of the FMA, the actual user query needs to be phrased as "What are all of the organs contained in the thorax or any of its parts?" This complex relationship can be written succinctly in StruQL as "part"*."contains" which can be read, "Starting from the thorax traverse 0 or more part relationships followed by a single containment relationship."

This example reveals one of the advantages of complex relationships. The underlying model can be arbitrarily precise, while the query interface can easily support the natural sorts of queries users want to ask. For example, it is technically true that only anatomical spaces can contain anatomical structures, but the query interface needs to support higher levels of abstraction. For example, the query interface could suggest replacing every appearance of "contains" with the less precise, but more intuitive, "part"*."contains" to the user. It is among our goals to construct a library of such intuitive conversions.

The indirection provided by paths also allows the underlying model to evolve while still exposing the same collection of virtual relationships. This logical independence is exactly analogous to virtual tables (i.e., views) in a relational database system. The intended meaning of a virtual relationship is defined by the semantics of StruQL, the presentation of which requires an elaboration of StruQL syntax.

#### *3.3.1. Syntax*

Fig. 3 presents a formal grammar describing the subset of StruQL we have implemented. The WHERE clause consists of variables, which are related to one another via path expressions: `X→P→Y`. Variables can also be equated with constants: `X=="Value"`. For convenience, the expression `X→P→Y, Y=="Value"` can be abbreviated: `X→P→"Value."` As further elaborated in Section 4.2.1, in StruQL, arbitrary paths can

**A**
```
WHERE
 H->"name"->N,
 H->"synonym"->S,
 N == "Heart"
CREATE
 Synonym(S)
```

**B**
```
WHERE
 H->"name"->N,
 H->"synonym"->S,
 N == "Heart",
 S{"LIKE 'C%'"}
CREATE
 Synonym(S)
```

**C**
```
WHERE
 H->"name"->N,
 H->"synonym"->S,
 H->"definition"->D,
 N == "Heart"
CREATE
 Synonym(S),
 Definition(D)
```

Fig. 2. Sample StruQL queries that retrieve information about the heart. The first query retrieves synonyms. The second retrieves synonyms that start with the letter C. The third retrieves synonyms and definitions.

```
Query :– WhereClause CreateClause
WhereClause :– WHERE [WhereExpression ,]* WhereExpression
WhereExpression :– Var -> Path -> Term
        | Var == String
        | Var { SQLString }
Term :– Var
        | String
Path :– Path . Path
        | Alternation
        | Var
Alternation :– Alternation | Alternation
        | Closure
Closure :– String Modifier
Modifer :– ?
        | *
        | +
        | ε
CreateClause :– CREATE [CreateExpression ,]* CreateExpression
CreateExpresion :– Var ( [Var ,]* )
```

Fig. 3. EBNF grammar for the currently implemented subset of StruQL.

be supported; at present we require a specific order of operations: closure, then alternation, then concatenation. The CREATE clause consists of node construction functions, parameterized using variables from the WHERE clause.

Precise semantics can be found below, but we will first consider an example. Until now we have been assuming that the (unique) identifier for every node in the semantic network is meaningful. This is not, in fact, the case. Nodes are uniquely identified using arbitrary numbers. However, because the FMA is constructed using Protégé-2000, every node (frame) has exactly one ":NAME" edge relating that node to its human-readable name. Fig. 4 displays a small portion of the FMA to illustrate.

Given this sample network, the query in Fig. 5A retrieves the names of all organs contained (using the colloquial definition) in the thorax. Variable X will be bound to the (single) node whose ":NAME" is "Thorax." From that node, the query explores every sub-part searching for a containment relationship. The nodes found in this manner are bound to the variable Y. This set of nodes will then be furthered constrained to only those that are subclasses of organs. Finally, the

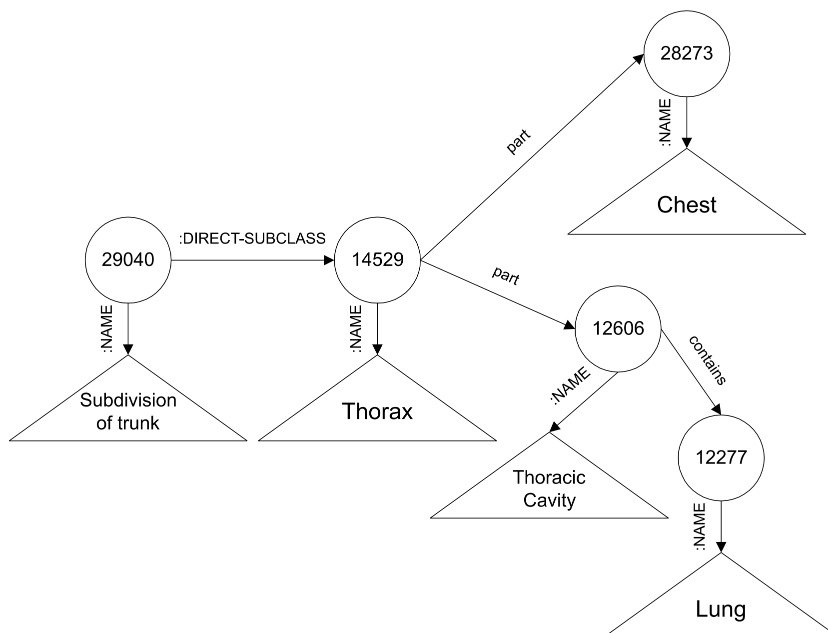


Fig. 4. Sample subset of the FMA semantic network.

**A**
```
WHERE
 X->":NAME"->"Thorax",
 X->"part"*."contains"->Y,
 Y->":DIRECT-SUPERCLASSES"+.":NAME"
  ->"Organ"
 Y->":NAME"->Contains,
CREATE
 TheThorax(Contains)
```

**B**
```
<results>
  <TheThorax>
    <Contains>Lung</Contains>
  </TheThorax>
  <TheThorax>
    <Contains>Thymus</Contains>
  </TheThorax>
  <TheThorax>
    <Contains>Right lung</Contains>
  </TheThorax>
  <TheThorax>
    <Contains>Left lung</Contains>
  </TheThorax>
</results>
```

Fig. 5. Sample StruQL query and results for the question, "What are the names of the organs contained in the Thorax?".

":NAME"'s of these nodes are bound to the variable `Contains`.

For every value identified by this query, a new node will be created and returned as indicated by the CREATE clause. The result of running this query can be found in Fig. 5B. In this example, 5 different values for `Contains` have been identified. For each such value an XML element is generated; one such element is:

```
<TheThorax>
  <Contains>Lung</Contains>
</TheThorax>
```

Each node is returned as a separate element whose sub-elements correspond to the variable names and values passed to the node constructor. This collection of 5 elements represents an XML forest. To ensure that the results correspond to a valid XML document (i.e., a tree), the results are aggregated inside a top-level `<results>` tag. This binding of variables and construction of elements is defined by the semantics of a StruQL query.

*3.3.2. Semantics*

The WHERE clause generates a series of variable bindings. Let **N** represent all of the nodes in the semantic network and let **E** represent all of the edges in the semantic network. Finally, let **Q** represent all of the variables in the WHERE clause. The semantics of the WHERE clause is the set of all assignments from **Q** into **N**∪**E** such that all of the conditions in the where clause are satisfied. That is, for every path constraint `X→P→Y`, `P` connects `X` and `Y`, as defined previously. `P` can itself be a variable, in which case some edge must connect `X` and `Y`. In addition, every binary condition (of the form `X==`"`String`" or `X` {"`String`"}) must also hold. The former holds if the value of `X` equals the string constant. The latter construct allows one to use any binary condition supported by SQL; the semantics of this comparison are defined by SQL (the string value is passed directly to the SQL engine). For example, Fig. 2B shows how to use a LIKE clause to constrain a variable.

For each path constraint `X→P→Y`, there are potentially an infinite number of paths connecting `X` and `Y`. However, the semantics of the WHERE clause restrict the output to a finite collection. Let **N**∪**E** contain k elements and let **Q** contain q variables. There are at most $k^q$ assignments from **Q** into **N**∪**E**. Thus, the WHERE clause must return a finite collection (polynomial in the size of the network).

The CREATE clause generates output based on the bindings returned by the WHERE clause. Each expression in the CREATE clause is an element constructor. (More formally, each expression corresponds to a Skolem function [19].) For each *distinct* binding of the variables listed, a new XML element is generated. The tag for this element is the name of the element constructor (e.g., `TheThorax`). The element contains one sub-element for each argument; the tags for these elements are the variable names (e.g., `Contains`). Finally, to guarantee that the resulting document is valid XML, the CREATE clause wraps all of the elements it creates in a `<results>` tag.

The query in Fig. 5A retrieves the names of the organs contained in the thorax, or one of its sub-parts. The only possible binding for `X` is the node whose name is "Thorax." There are several possible bindings for `Y`, one for each node reachable from `X` by the path "part"*."contains" such that a path along the ":DIRECT-SUPERCLASSES" link exists between `Y` and a node whose ":NAME" is "Organ." Finally, for every binding of `Y`, there is exactly one binding for `Contains` because each node has a unique ":NAME." Using a more extensive version of the network in Fig. 4, the results of this query are displayed in Fig. 5B. The OQAFMA server provides an efficient implementation of these operations.

## 4. OQAFMA system architecture

We designed OQAFMA as a server capable of receiving socket connections; it accepts StruQL queries and produces XML results. Fig. 6 presents an overview of the system architecture: Anatomical knowledge is entered in Protégé-2000 [5], which stores the data in a

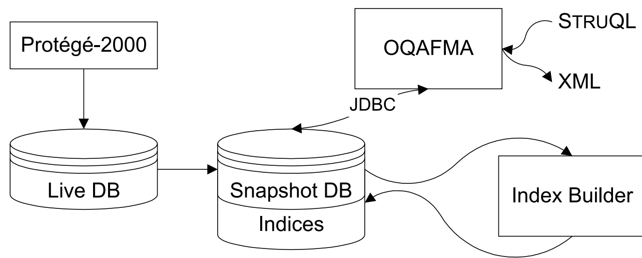


Fig. 6. System overview—the FMA is developed using the live DB. A snapshot is constructed in a read-only database, against which several indices are built. OQAFMA converts StruQL queries to SQL and returns the results as XML.

MySQL [23] database. A copy of the database is transferred into a PostgreSQL [24] database where several indices are constructed. OQAFMA reads StruQL queries from an incoming connection and converts those into appropriate SQL queries. The results are converted into XML and written to the socket.

This architecture incorporates several techniques for efficiently processing StruQL queries, which can broadly be classified as preprocessing and runtime techniques. The key runtime technique is conversion of StruQL to SQL, the advantage of which is that relational databases are highly optimized to perform joins. Preprocessing techniques involve the construction of space-intensive data structures against which the SQL queries are posed.

### 4.1. Preprocessing

Knowledge entered in Protégé-2000 can be stored in any database compliant with Java Database Connectivity (JDBC [25]). This version of the FMA is referred to as the 'live' version and is only accessible to the FMA authors. A 'snapshot' copy of the live database is made in a second read-only database (on another machine) for two reasons:

1. By transferring the database to a second machine, there is no contention between users attempting to query the system (generating heavy read traffic) and authors attempting to enter new knowledge (generating heavy write traffic).
2. Using a snapshot of the live version allows for the possibility of not publishing all of the information in the live version.

This latter advantage is significant. Determining the ideal representation for certain relationships is a matter of trial and error. The authors of the FMA do not necessarily want to expose this experimentation to the general public. When a snapshot of the database is made, the authors can select a subset of the FMA they wish to expose. Future implementations could even use database privileges to provide different subsets of the model to different users.

Creating a snapshot takes a few hours to run to completion—there are currently more than 1.1 million tuples to transfer. This process is slow, but adequate as long as snapshots are taken infrequently. If one desires more frequent snapshots (e.g., daily), then the live database could be transferred to the same database platform as the snapshot (although the databases can be stored on different machines). This architecture would eliminate the need to go through JDBC (or some other intermediary) when creating the snapshot.

Once a snapshot has been constructed, the database must be optimized for the retrievals requested in a StruQL query. For simplicity and flexibility, Protégé-2000 stores everything in a single *fact table* (whose basic schema, as shown in Fig. 7, is `FMA(Frame, Slot, Value)`). This schema is easy to maintain, but offers poor performance, especially when the goal is retrieval of a specific relationship (e.g., when executing a projection query). As a result, the next step is to create one new table for every edge. If there are E edge-types, then E new tables are constructed of the form `Slot_Id_Index(Head, Tail)`. Every row in `FMA` (of the form (`FMA(f,s,v)`) is replicated as `Slot_s_Index(f,v)`, i.e., there is an edge (labeled `s`) from `f` (the head) to `v` (the tail).

This schema facilitates single edge retrieval, but StruQL also includes closure operators (+ and *). Most often, a closure is computed over a single edge-type. As a result, the transitive closure of every edge-type is pre-computed and stored in a table of the form `Slot_Id_Plus(Head, Tail)`, provided that the edge-type connects two nodes (as opposed to connecting a node and a value). The transitive closure is computed using Tarjan's algorithm [26].

Finally, it is necessary to support the optional operator (?). This is another operation that only makes sense if the edge-type connects two nodes. For each of these edge-types, two views are added to the database: `Slot_Id_Index_Opt` and `Slot_Id_Plus_Opt`. These views are the union of `Slot_Id_Index` (or `Slot_Id_Plus`) and a special `NoOp` table. The `NoOp` table contains one entry for every node in the database, connecting it to itself via the null (or ε) edge. As a result, when a specific node is retrieved from `Slot_Id_In-`

```
FMA(Frame, Slot, Value)
NoOp(Head, Tail)
Slot_Index(ID, Name)
```
*For each distinct edge in Facts:*
```
Slot_Id_Index(Head, Tail)
```
*For each distinct edge in Facts connecting two nodes:*
```
Slot_Id_Plus(Head, Tail)
Slot_Id_Index_Opt(Head, Tail)
Slot_Id_Plus_Opt(Head, Tail)
```

Fig. 7. Snapshot database schema.

dex_Opt, the result includes the values reachable from that node via the indicated edge as well as the node itself. This is exactly the definition of the optional operator.

Finally, Protégé-2000 stores edges using a unique identifier, but StruQL queries use edge names. As a result, a look-up table (Slot_Index) is constructed that maps edge names to the corresponding identifiers. This index is used for look-up when converting from StruQL to SQL.

Taking a snapshot and building the indices are performed by a single script, which takes roughly 15 h to complete. We are considering re-implementing this step entirely within the database engine (to improve performance). Since the time between snapshots is large (roughly 2 weeks), the time needed to construct indices is currently not an issue. Of greater concern is to minimize the time required for runtime processing.

### 4.2. Runtime processing

The subset of StruQL that is supported by OQAFMA was chosen based on the extent to which the pre-computed indices can be leveraged. For example, the closure of a concatenation (like (a.b)*) does not correspond to any index, nor can it be easily built from the indices. Each of the components of WHERE statement supported in OQAFMA relates to an SQL operation over the pre-computed indices; these operations are, in turn, optimized by the relational database. We distinguish the processing of WHERE and CREATE statements and conclude this section by illustrating the querying of OQAFMA.

#### 4.2.1. Processing WHERE statements

There are two basic types of statements that can be made in a StruQL WHERE clause. The first is a unary assertion restricting the range of a variable (either using equality or the SQL escape sequence). These restrictions are passed directly to the database as part of the SQL WHERE clause.

It is also possible to express binary (or ternary) relationships using paths (and edge variables): X→Path→Y or X→L→Y. Each edge mentioned in a path expression corresponds to a specific index table in the database. Use of an edge variable corresponds to the fact table.

Given these correspondences, the conversion from StruQL to SQL is relatively straightforward. In each path expression, every edge is modified by a closure operator (* or +), an optional operator (?) or nothing. An edge/modifier pair uniquely determines which index contains the relevant data. (Because Protégé-2000 uses numeric edges and StruQL uses the corresponding names, the EdgeIndex is kept in memory and used to perform this translation.)

The StruQL grammar supports the alternation of edge/modifier pairs. Each use of alternation corresponds to a SQL UNION. For example, (a|b) corresponds to (aIndex UNION bIndex). Finally, it is possible to concatenate multiple alternations (or edge/modifier pairs, which are trivial alternations). Each concatenation (a.b) corresponds to a JOIN in which a's value is equal to b's node. Thus, every path expression constructs a view that consists of a collection of joins across a group of unions. The 'first' node column and 'last' value column are bound to X and Y, respectively.

Edge variables are easier to convert to SQL, but require using the fact table, which can be quite large. Each ternary expression (X→L→Y) binds FMA(node, edge, value) to X, L, and Y, respectively.

Finally, whenever a variable is shared across clauses, an additional constraint is added to the SQL WHERE clause to enforce this similarity. Fig. 8 shows the SQL that results from a specific StruQL query.

In this example, the first clause (X->":NAME" ->"Thorax") results in X being assigned the value 14529. The system knows that :NAME is a unique at-

**A**

```
WHERE
 X->":NAME"->"Thorax",
 X->"part"*
   |"general part"*
   ."contains"->Y,
 Y->":NAME"->Contains
CREATE
 TheThorax(Contains)
```

**B**

```
SELECT T3.tail AS Contains FROM
((SELECT * FROM slot_63832_plus_opt)
           UNION
 (SELECT * FROM slot_163208_plus_opt))
AS T1,
((SELECT * FROM slot_155777_index))
AS T2,
((SELECT * FROM slot_2002_index))
AS T3
WHERE
  14529 = T1.head AND
  T1.tail = T2.head AND
  T2.tail = T3.head;
```

Note that 14529 is the node identifier associated with Thorax; this lookup eliminates a join. The edge names (part, contains, etc.) have been converted to node ids (63832, 155777, etc.).

Fig. 8. Sample StruQL query and the resulting SQL for a query similar to the one presented in Fig. 5.

tribute and therefore immediately converts the name to a node identifier to eliminate a join.

The next clause (`X->"part"*|"general part"*."contains"->Y`) generates two temporary tables (`T1` and `T2`). The alternation generates table `T1`, which is the union of the tables corresponding to "part"* (using index `slot_63832_plus_opt`) and "general part"* (using index `slot_163208_plus_opt`). This result is joined with `T2` (the index `slot_155777_index` corresponds to "contains") by equating the tail of `T1` with the head of `T2`. From this result, `X` will be bound to `T1.head` and `Y` to `T2.tail`.

The final WHERE clause (`Y->":NAME"->Contains`) introduces a new relationship, `T3`, which corresponds to ":NAME" using the index `slot_2002_index`. The shared use of the variable `Y` joins `T3` to `T2`. The new variable `Contains` is bound to `T3.tail`.

The only variable mentioned in the CREATE clause is `Contains`. As a result, `T3.tail` is the only column returned by this query. This resultset is returned to the server and turned into XML based on the CREATE statement.

#### 4.2.2. Processing CREATE statements

The expressions in the CREATE clause are implemented as hash tables. A cursor iterates over the relation returned by the database. As each tuple is encountered, the values in the columns corresponding to each element constructor's parameters are compared against the contents of the hash table for that constructor. If these values are not present in the hash table, a new XML fragment is created:

```
<fn>
  <var1>value1</var1>
  <var2>value2</var2>
  ...
</fn>
```

Each `<fn>` tag contains the name of a constructor. Each `<var>` tag contains the name of a variable. Within each `<var>` element is the current value of that variable. Finally, the values are added to the hash table (so that they will not be output a second time).

This approach requires memory proportional to the size of the result relation times the number of constructors. Some of the techniques presented in [27] may be applicable to reduce the performance penalty of this crude approach.

#### 4.2.3. Querying OQAFMA

OQAFMA is running as a server, listening for socket connections on a devoted port (4242). A client interested in using OQAFMA opens a socket connection (TCP/IP) to the machine hosting the server (quad.biostr.washington.edu). Once a connection has been established, one StruQL query can be sent to OQAFMA (one query per connection corresponds to the HTTP/1.0 protocol). The end of that query is indicated using a semi-colon (;). Once a StruQL query has been written to the socket, the XML results can be read from the socket. Fig. 9 illustrates sample Java code.

```
Socket s =
     new Socket("quad.biostr.washington.edu", 4242);
String struql =
     "WHERE Hd->\"contains\"->T1 CREATE N(Hd, T1);";
s.getOutputStream().write(struql);
String result = s.getInputStream().read();
```

Fig. 9. Sample Java code for connecting directly to OQAFMA (note that a buffered reader/writer is actually needed to receive/send Strings, but this has been removed for clarity and brevity).

## 5. Evaluation

Our purpose with developing OQAFMA was to provide flexible and efficient access to the FMA for application developers competent in database queries. We have completed an evaluation of the performance of OQAFMA in terms of its flexibility and efficiency, which we will follow up with a full-scale evaluation once application developers begin to use OQAFMA for accessing the FMA and other UMLS resources. The flexibility of our interface can be measured by determining what proportion of queries of interest can be expressed in StruQL. The efficiency of our implementation can be measured directly in terms of speed of query evaluation. The first step was to establish a corpus of queries of interest.

### 5.1. Methods

We established queries of interest in consultation with anatomists. They provided for us a collection of 50 queries they believed could be answered using the FMA irrespective of the potential difficulty in extracting the answers to these queries. When possible, these English queries were converted to StruQL. We then executed each StruQL query 10 times (at different times of the day).

The queries we obtained from the anatomists ran the gamut from trivial (e.g., "What kind of cell is a sperm?") to complex (e.g., "What structures are posterior to and to the right of the T8 part of the esophagus?"). Once converted to StruQL, the number of relationships (excluding ":NAME") used ranged from 1 (e.g., "What kind of synapses are there?") to 7 ("What muscle is attached to the coracoid process and humerus?"). On average, 2.25 relationships were used per query.

### 5.2. Results

Of the 50 queries, 7 of them could not be converted to the subset of StruQL supported by OQAFMA. There were three factors that prevented these queries from being converted.

Table 1
Time to first result by query class

| Query class | N | Minimum time (ms) | Median time | Maximum time (ms) | Mean time (ms) |
|---|---|---|---|---|---|
| All | 31 | 481 | 551 ms | 1314 | 677 |
| Synonyms | 2 | 1280 | n/a | 1314 | 1297 |
| Intersections | 5 | 1018 | 1094 ms | 1096 | 1075 |
| All others | 24 | 481 | 542 ms | 644 | 543 |

First, OQAFMA does not support negation. There were 5 queries involving negation including "What parts of the aorta are not in the superior mediastinum?" and "What is the difference between cytoplasm and protoplasm?" (The latter question asks what is true of one concept, but not true of the other.)

Second, we have not implemented nested queries, which would allow us to answer, "What is calmodulin?" This query asks for the definition (if any) and superclass of calmodulin. Because no definition has been entered for calmodulin, the query fails—every variable must be bound to a node, or none will be.

Third, we do not support arbitrary closure operations. Hence, we cannot answer the query, "Does the T5 segment of the spinal cord contribute to the greater splanchnic nerve?" The continuity relationship has been modeled using a reified relationship. To identify the structures immediately continuous with the greater splanchnic nerve requires traversing the path "continuous with"."related part" and finding all such structures requires a closure operation on this concatenation.

Of the remaining 43 queries, 12 could be converted to StruQL, but could not be answered using the current contents of the FMA, which is a work in progress. We do not consider these queries further.

The amount of time required to answer the 31 queries that could be answered (correctly in all cases) ranged from 481 to 1314 ms. (Note that all timings are from the moment the query is sent to the moment the first result is received; the size of the result is not a factor.) The median response time was 551 ms and the mean response time was 677 ms.

The large difference between median and mean response times (and a visual analysis of the results) is consistent with a bimodal distribution. This led us to investigate what feature or features were shared by the slower queries. We were able to identify two factors, each of which only occurred in slow queries. (Every slow query exhibited at least one of these factors as well.)

First, two of the queries used a synonym instead of the preferred name. The preferred name of a concept can be retrieved directly using ":NAME." Retrieval using preferred names or synonyms requires the more cumbersome "Preferred name"|"Synonym"."name" construct. This involves a union of two large relations.

Second, five of the queries asked for the intersection of two large hierarchies (at least one of which was the "part" hierarchy in four of the queries). As shown in Table 1, the average response time for these intersection queries is slightly less than twice the response time for simpler queries. If this intersection were performed outside of OQAFMA (i.e., in the application using OQAFMA), the time required would be at least double (since two queries would be needed), not including the time to perform the intersection.

In conclusion, the number of queries that could not be converted into StruQL seems, at first, disappointing. Note, however, that whereas we could answer 43 queries directly, the Protégé-2000 API only supported 10 of the queries (with a single function call—any of the queries can be answered with a custom program). Of the seven incompatible queries, five involved negation. To support negation, we would need to make a closed-world assumption. Since the FMA is a work in progress, incorrect results would be returned for many queries involving negation (since under closed-world semantics, missing information implies negation). To solve this problem, we will need to address nesting and arbitrary closure.

The efficiency results are very promising. The time required to answer a query did not depend on the number of relationships mentioned in the query, but instead on the topology of the query. This justifies relying on the database engine to optimize the query. Moreover, synonym queries can be improved, as we discuss in Section 7.1.

## 6. Related work

Since in the development of OQAFMA we have extensively relied on StruQL (an existing query language for semi-structured data), Protégé-2000 (a frame-based knowledge representation system), and XML (a data exchange standard), we provide a rationale for our design choices. We also discuss how OQAFMA relates to other projects, including systems for publishing data in XML and alternative query languages.

### 6.1. Semi-structured query languages

OQAFMA can only return results to queries if the underlying representation scheme for the information to be navigated is a semantic network, or a labeled graph. Interest in labeled graphs as data structures has surged in

recent years, largely because of the ease with which one can represent semi-structured data as a labeled graph. One advantage of a semi-structured data model, as presented by Buneman et al., [28] is that data and schema values are stored together, "blurring the distinction between schema and instance." Popular semi-structured formats include OEM [29], ASN.1 [30], and XML [7].

As noted earlier, XML has become the de facto data standard and XQuery [14], developed by the World-Wide Web Consortium (W3C) its standard interface. XQuery borrowed from a number of earlier query languages, and therefore shares certain features with StruQL. Both languages allow one to bind variables to nodes in a graph; in XQuery these paths are expressed using XPath [31] (another W3C standard). The XPath language supports both closure operations and concatenation.

From the perspective of the FMA, the limitations of XQuery are profound. XPath does not support alternation or the optional operator, which makes it difficult to look-up concepts by name because the query term may be a preferred name or synonym (recall that a disjunction requires alternation).

The XQuery data model is that of a tree. Graphs are supported by wiring elements together using unique identifiers and references (IDREFs) to those identifiers. As a result, closure in XPath requires that elements be nested; it is not possible to express a closure operation across references.

Finally, the ability to bind a variable to metadata is limited. XQuery does allow one to bind a variable to the name of a tag, but tags are strings. Thus, it is not possible to determine the properties of the relationship indicated by the tag. In OQAFMA, one can bind a variable to an edge label, which can in turn participate in other relationships. For example, one can retrieve cardinality or type constraints that pertain to the edge.

The first language for semi-structured data to include the ability to bind variables to edges was Lorel [15], developed for the Lore database management system. Lorel (like StruQL) was designed to query the OEM data model. Paths in Lorel can be constructed using all of the operations listed for StruQL (including concatenation, alternation, closure, and the optional operator).

Lorel supports edge variables as well as two novel constructs: wildcards and path variables. Wildcards allow one to partially describe the name of an edge (e.g., zip% matches both zip and zipcode). According to [15], "the value of a path variable is a data path in the OEM graph." A StruQL edge variable is bound to a single edge; a path variable is bound to a series of edges. In the presence of a cycle, a path variable could (in theory) be bound to an infinite number of values.

Despite the power of Lorel, it was rejected as the basis for OQAFMA for three reasons. First, the language is very wordy. It was derived from OQL [32], which was in turn derived from SQL. The succinct syntax in StruQL simplifies parsing. Second, Lorel is a strongly typed language. This is ideal for object-oriented databases, but unnecessary for the applications supported by OQAFMA. Finally, the object creation methods in Lorel are complex and not as powerful as the simple constructors in StruQL.

There are a number of other query languages for tree-based structures (like XML without IDREFs). An early language was UnQL [33], which introduced the notion of structural recursion. XSLT [34] is used to reorganize or display (in HTML) an existing XML document. Quilt [35] was a forerunner to XQuery.

From the perspective of querying the FMA, these languages all assume that the data can naturally be represented as a tree. The FMA contains far too many interconnections for this to be the case. Therefore, Protégé was found to be a more suitable representation for the FMA, although the richness and complexity of relationships in the FMA push the envelope even of this expressive knowledge representation environment.

### 6.2. Protégé-2000 API

OKBC was developed as a general protocol for accessing frame-based systems. In contrast to the plethora of semi-structured languages, it seems to be the only widely used protocol for frame-based systems. This protocol was instrumental in guiding the development of the Protégé-2000 knowledge-modeling environment.

A Protégé knowledge base is represented as a Java class. One can query the knowledge base object for the frame (or frames) with a given name (or pattern). Since slots are also frames, this same mechanism allows one to retrieve a slot by name. Given a frame object and a slot object, one can query the frame for the values of that slot. This corresponds to the most basic selection query. By iterating over the collection of all slots, one can retrieve all of the frame-slot-value triples for a given frame.

The drawbacks of using this API for a server are twofold. First, all queries are performed by invoking methods on Java objects. Thus, clients are constrained to the Java language. Second, projection and path queries are not supported. These features served as the chief motivators for developing OQAFMA. In addition, we wanted to export large portions of the FMA as XML, a task not directly supported by the Protégé-2000 API.

### 6.3. XML publishing

The basic goal of OQAFMA is to export portions of the FMA as XML. The task of publishing relational data as XML has been tackled by a number of projects. Before describing the specific projects, it is worth noting that our task was to export a semantic network as XML.

Microsoft's SQL Server 2000 supports directly publishing relational data as XML. The simplest approach

is to write an SQL query. Each tuple becomes an element in which the attribute names are column names and data values are attribute values. A more sophisticated approach provides a default nesting of elements based on the order in which columns appear in the SELECT clause. The most complicated approach requires writing an SQL query that produces a result that corresponds to the "Universal Table" format, the details of which can be found in [36].

SQL Server 2000 supports a second approach in which one or more views (written using XDR) are defined. These views describe how the tables will be nested (using foreign keys). The server then accepts XPath expressions to select a subset of a given view.

This approach is very similar to the one proposed in the SilkRoute project [37]. SilkRoute composes a user query (written in XQuery) with a global query (also written in XQuery). Because XQuery allows the user to reformat an XML document, users are not required to retrieve results as they are structured in the global query.

In both cases, the database designer describes how a number of relational tables nest within one another. This is not appropriate for the FMA, which stores all of the data in a single table. Moreover, this approach does not allow one to encode an arbitrary hierarchy. As a result, it is not possible to perform a closure operation because XPath does not support closures across references.

Finally, work at IBM [27] focused on the best approach for structuring and tagging XML generated by a relational database engine. They identified three basic strategies based on which component was responsible for structuring the results, and which for tagging. (Note that XML cannot be tagged until it is structured.) These functions can be implemented in the database engine itself, or by middleware. The experiments suggest that the database should be responsible for structuring. Tagging should be performed by middleware when the result set is small enough to fit in main memory, and in the database engine otherwise. OQAFMA currently structures and tags in middleware, but we are exploring using the relational engine.

This analysis provides the rationale for the FMA's implementation in Protégé-2000 and for OQAFMA's current architecture, which uses StruQL as its input query language and XML to output results. Our ongoing work with OQAFMA and its relationship to other projects has also led to a number of qualitative observations, which we discuss in the next section.

## 7. Discussion

The motivation for the work we describe in this paper was provided by the need to develop query mechanisms that could assist in the development of knowledge-based applications by facilitating the retrieval of complex information embedded in existing and evolving knowledge bases. The paucity of such applications contrasts sharply with the expanding number of biomedical ontologies and the numerous sources embraced by UMLS. Anatomy, the domain we selected as a substrate for developing the query agent OQAFMA, illustrates the absence of computable knowledge in web-accessible educational applications [38]. While interactive images of varying type and quality are widely used, all knowledge is presented in the form of text; not a single program could be found that makes use of machine-based inference.

As in the case of the Foundational Model of Anatomy itself, the motivation for developing OQAFMA initially derived from the need perceived in anatomy education. However, during the development process of both the FMA and OQAFMA, we recognized that the problems we encountered and the solutions we devised targeting anatomy, generalize to the much broader fields of bioinformatics and medical informatics. It is in this spirit that we present our report and contend that the significance of OQAFMA lies in its applicability to any concept domain that is represented in a system reliant on a semantic network. This potential of OQAFMA for generalizing to diverse knowledge sources is largely the consequence of the query language StruQL integrated in OQAFMA's architecture. Therefore we focus this discussion on the advantages of the design choices we made for OQAFMA's architecture, before presenting our views on the relevance of OQAFMA to the Foundational Model of Anatomy in particular, and UMLS in general.

### 7.1. Design choices for OQAFMA

We developed OQAFMA as a server, which implements a subset of StruQL [6]. As we illustrate in Section 3.3, this language is capable of supporting a wide variety of queries ranging from the trivial to the complicated. Although in Section 5 we present evidence for OQAFMA's effectiveness for processing all classes of queries, the role we envisage for the FMA in biomedical informatics [4] motivates us to consider upgrading OQAFMA's design, and we allude to these plans in the following discussion.

Since implementing OQAFMA for the FMA two years ago, we have had considerable opportunity to evaluate our design choices. The design process and subsequent use have led us to some interesting observations regarding some schema issues, choice of a query language and optimizing the processing of virtual relationships.

#### 7.1.1. Schema issues

Our current schema constructs one index for each slot. This schema is not intuitive because a frame-based system is organized around the nodes in the network, not the slots. As a result, the information pertaining to a given concept becomes scattered throughout the

database. We intend to address this problem based on the work of Shanmugasundaram et al. [39], which suggests a more elegant database design. The design of our indices could benefit from collapsing several indices based on cardinality constraints. Whenever two indices are collapsed, the conversion from StruQL to SQL may be able to eliminate a join operation. As demonstrated in [39] (using the XML data model), the performance benefits can be significant. We would like to replicate these results using our data model.

#### 7.1.2. Choice of query language

We originally chose StruQL for its simplicity and elegance. This has, for the most part, been a good decision for at least three reasons. First, one of the advantages of StruQL for querying the FMA is that its underlying edge-labeled data structure corresponds to the frame-based data model of the FMA, which is essentially a graph. Second, program developers are able to learn the language quickly and begin using OQAFMA with little assistance from the developers. Third, the conversion of StruQL to SQL is fairly straightforward. One drawback of StruQL is that there is not (to the best of our knowledge) much work being done on its optimizations or extensions. More widely adopted query languages, such as XQuery [14] or XPath [31], may present an advantage when we begin to align or integrate the FMA with other data sources, such as the Gene Ontology [40]. The underlying data structure of these languages, however, is a tree, rather than a graph. Confining our queries to the FMA allowed us to postpone the adoption of methods for navigating trees, but the alignment of other knowledge sources with the FMA may force us to confront this issue, which may also be of relevance to querying the UMLS. Patel-Schneider and Siméon [41] have proposed a way to simultaneously accommodate an XML tree and an RDF graph, which may hold promise for addressing this requirement in OQAFMA.

#### 7.1.3. Virtual relationships

An advantage offered by OQAFMA is the navigation of virtual relationships; their processing, however, could be further optimized. We have observed that certain virtual relationships are commonly used. One example is the virtual relationship "has-term," which is ("Preferred name"|"Synonyms").("name"|"Latin name (TA)"). This relationship is used extensively by the natural language interface that converts English questions to StruQL queries [8]. Because the program cannot guarantee that the user knows the preferred name for a given concept, it must retrieve all synonyms (including Latin terms). In SQL, this virtual relationship requires joining the results of two UNION queries (over fairly large tables). As a result, these queries are, by comparison to others, slow. We are considering techniques that will allow us to define and pre-compute commonly used virtual relationships.

The enhancements we currently envision will further improve the functionality and performance of OQAFMA by the time application developers outside our own group adopt this query interface. However, even in its present state, OQAFMA has proven to be useful in developing applications that query the FMA. Since OQAFMA is independent of the FMA, the methods presented in this paper are useful for querying any knowledge base that can be expressed as a semantic network. In particular, they are immediately applicable for querying the large number of knowledge bases that are being developed using the Protégé toolkit and can easily be adapted to query across the UMLS source vocabularies.

### 7.2. Relevance to the FMA

The immediate relevance of OQAFMA to the FMA relates to its role in the evaluation of this evolving knowledge base. There is no gold standard with which the FMA can be compared and it is too large and complex for domain experts to evaluate its semantic structure by browsing the model. Querying the FMA is proving to be an indispensable requirement for assessing the presence or absence particular pieces of information. While OQAFMA can be used directly for this purpose, it is also critical for supporting a prototype natural language interface [8], which is currently intended for evaluators of the FMA. A different interface designed to constraining queries to concepts and relationships currently represented in the FMA links directly to Protégé [11], is being redesigned to make use of OQAFMA.

In the longer term, our intent is to convert the widely used Digital Anatomist interactive web atlases [42] into knowledge-based tutorials by enhancing them with knowledge represented in the FMA. OQAFMA will be critical for developing these applications and also for providing the foundation for self-evaluation tools designed for various types of users. Since we regard anatomy as foundational to other fields of biology and medicine, we hope that these applications, as well as OQAFMA itself, will promote the development of next-generation "smart" programs not only for education but for research and clinical medicine as well. In particular, we regard the FMA as a reference ontology for biomedical informatics [4], and once aligned with other ontologies in this domain, we envisage evolving versions of OQAFMA as playing a critical role in extracting knowledge from these interrelated resources.

### 7.3. Relevance to UMLS

As mentioned in the introduction, UMLS accomplishes interrelation between its constituent vocabularies through its Semantic Network; therefore it lends itself

well for querying by OQAFMA. With the addition of the Gene Ontology [40] and other evolving bioinformatics ontologies to UMLS, a foundation is being established for supporting inference about normal and perturbed biological structure and function, provided queries can cross unimpeded from one ontology to the other. We envisage OQAFMA as the prototype tool for retrieving information from UMLS's extended SN in response to queries about biological structure and function through applications that target education, research, and health care. Our experience with querying the FMA suggests that even in its current state, OQAFMA can free programmers from concerns about how to retrieve information, allowing them to write simple, declarative queries that could extract knowledge from across a range of UMLS sources.

One of our intents with this publication is to provide a motivation for planning an experiment that would test such a hypothesis. Designing and implementing such an experiment could promote not only improvements and enhancements in OQAFMA, but also some revisions in the UMLS Semantic Network and in the implementation of some of its source vocabularies, in order to facilitate querying the extended SN at conceptual levels higher than those afforded by keyword search.

## 8. Conclusion

We describe the development and operation of OQAFMA, an interface we designed for processing different classes of queries submitted to large knowledge bases. Based on the Digital Anatomist Foundational Model of Anatomy as a test bed, we propose a classification of queries that, regardless of content, generalize to any concept domain, and can be submitted to any knowledge source, provided a semantic network underlies its implementation. OQAFMA meets the three basic requirements we set for a query interface: (1) it supports all classes of queries, including those to which answers are not explicitly represented in the knowledge base; (2) it returns results in XML, a de facto standard for data exchange, in a form also intelligible to humans; (3) it operates with efficiency on the FMA, which is one of the largest and most complex knowledge bases in biomedical informatics, returning results to queries in the response time range of 481–1314 ms.

These performance characteristics of OQAFMA are assured by several factors: (1) it leverages the power and efficiency of a relational database engine; (2) it makes use of a subset of StruQL, an expressive and flexible query language for semi-structured data; (3) it converts each StruQL query into an SQL query, whose results are serialized as XML; (4) it pre-computes transitive closures over single relationships; (5) it supports paths that concatenate encoded relationships into so-called virtual relationships, which are absent from, and are more complex than those explicitly represented in, the knowledge base. These characteristics provide for OQAFMA's ability to answer complex path queries reasonably quickly without sacrificing performance on simple queries.

We propose that OQAFMA be used initially for the evaluation of the FMA and subsequently for the support of knowledge-based applications that call for anatomical information. Moreover, OQAFMA can serve as a prototype for querying the extended Semantic Network of UMLS at conceptually higher levels than current query mechanisms support. We base this hypothesis on conclusions we reach in this communication about OQAFMA's performance on the Foundational Model of Anatomy.

## Acknowledgments

Funding for this work was provided by NLM training Grant T15LM07442, research Grants LM06316 and LM06822 and the Human Brain Project Grant DC02310. Thanks to Rachel Pottinger and the anonymous reviewers for feedback concerning the clarity of the paper. We would like to thank Kevin Hinshaw, Emily Chung, Vishrut Srivastava, and Greg Distelhorst for their development of other interfaces to the FMA; Dr. Hinshaw also developed the website for OQAFMA (http://sig.biostr.washington.edu/projects/oqafma/). Drs. José L.V. Mejino and Augusto V. Agoncillo helped us identify the queries used for evaluation. Finally, Jiang-Jiang Cheng implemented the feature that allows OQAFMA to remain online during the construction of indices.

## References

[1] National Library of Medicine (NLM). Unified Medical Language System (UMLS). Available from: http://www.nlm.nih.gov/research/umls/.

[2] Bodenreider O, Mitchell J, McCray AT. Evaluation of the UMLS as a terminology and knowledge source for biomedical informatics. In: Proceedings of the American Medical Informatics Association (AMIA) Annual Symposium; 2002 November 9–13. San Antonio, TX: AMIA; 2002. p. 61–5.

[3] McCray AT. Representing biomedical knowledge in the UMLS semantic network. In: Broering NC, editor. High performance medical libraries: advances in information management for the virtual era. Westport, CT: Meckler; 1993. p. 45–55.

[4] Rosse C, Mejino JL. A reference ontology for biomedical informatics: the foundational model of anatomy. J Biomed Inform 2003;36:478–500.

[5] Musen M, Crubézy M, Fergerson R, Noy NF, Tu S, Vendetti J. Protégé-2000. Stanford, CA: Stanford Medical Informatics. Available from: http://protege.stanford.edu/.

[6] Fernandez MF, Florescu D, Levy AY, Suciu D. A query language for a web-site management system. SIGMOD Rec 1997;26(3):4–11.

[7] Bray T, Paoli J, Sperberg CM, Maler E. Extensible markup language (XML) 1.0. 2nd ed. World Wide Web Consortium (W3C®). Available from: http://www.w3.org/TR/REC-xml.

[8] Distelhorst G, Srivastava V, Rosse C, Brinkley JF. A Prototype Natural Language Interface to a Large Complex Knowledge Base, the Foundational Model of Anatomy. In: Proceedings of the American Medical Informatics Association (AMIA) Annual Symposium; 2003 November 8–12; Washington, DC, USA; 2003. p. 200–204.

[9] Wong BA, Rosse C, Brinkley JF. Semi-automatic scene generation using the digital anatomist foundational model. In: Proceedings of the American Medical Informatics Association (AMIA) Annual Symposium; 1999 November 6–10. Washington, DC: AMIA; 1999. p. 637–41.

[10] Wong BA, Albright E, Hinshaw KP, Rosse C, Brinkley JF. The dynamic scene generator. Structural Informatics Group. Seattle, WA: University of Washington. Available from: http://sig.biostr.washington.edu/projects/dsg/.

[11] Shapiro LG, Chung E, Mejino JL, Detwiler LT, Brinkley JF. A query interface for evaluating relationships in a large biomedical knowledge base, the foundational model of anatomy. J Am Med Inform Assoc 2003 (Submitted).

[12] Shapiro SC. Path-based and node-based inference in semantic networks. In: Waltz D, editor. TINLAP-2: theoretical issues in natural languages processing. New York: ACM; 1978. p. 219–25.

[13] Bretl R, Maier D, Otis A, et al. The gemstone data management system. In: Kim W, Lochovsky FH, editors. Object-oriented concepts, databases, and applications. New York, NY: ACM Press; 1989. p. 283–308.

[14] Boag S, Chamberlin D, Fernandez MF, Florescu D, Robie J, Siméon J. XQuery 1.0: An XML Query Language. World Wide Web Consortium (W3C®). Available from: http://www.w3.org/TR/xquery/.

[15] Abiteboul S, Quass D, McHugh J, Widom J, Wiener JL. The Lorel query language for semistructured data. Int J Digital Libraries 1997;1(1):68–88.

[16] Winston ME, Chaffin R, Douglas H. A taxonomy of part-whole relations. Cogn Sci 1987;11(4):417–44.

[17] Mejino JL, Agoncillo A, Rickard K, Rosse C. Representing Complexity in Part-Whole Relationships within the Foundational Model of Anatomy. In: Proceedings of the American Medical Informatics Association (AMIA) Annual Symposium; 2003 November 8–12. Washington, DC: AMIA; 2003. p. 450–454.

[18] Luger GF. Artificial intelligence structures and strategies for complex problem solving. Harlow, England: Pearson Education Limited; 2002.

[19] Russell S, Norvig P. Artifical intelligence a modern approach. Upper Saddle River, NJ: Prentice Hall; 1995.

[20] Chaudhri VK, Farquhar A, Fikes R, Karp PD, Rice JP. Open knowledge base connectivity 2.0.3. Stanford, CA: Stanford University; 1998.

[21] Chaudhri VK, Farquhar A, Fikes R, Karp PD, Rice JP. OKBC: a programmatic foundation for knowledge base interoperability. In: Fifteenth National Conference on Artificial Intelligence; 1998 July 26–30. Madison, WI: AAAI Press/The MIT Press; 1998. p. 600–7.

[22] Hopcraft JE, Ullman JD. Introduction to automata theory, languages, and computation. Menlo Park, CA: Addison-Wesley; 1979.

[23] MySQL®. Available from: http://www.mysql.com/.

[24] PostgreSQL. Available from: http://www.postgresql.org/.

[25] JDBC™ Data Access API. Sun Microsystems. Available from: http://java.sun.com/products/jdbc/.

[26] Nuutila E, Soisalon-Soininen. A Single-Pass Algorithm for Transitive Closure. Technical Report. Helsinki, Finland: Helsinki University of Technology, Laboratory of Information Processing Science; 1993. Report No.: TKO-B95.

[27] Shanmugasundaram J, Shekita E, Barr R, et al. Efficiently publishing relational data as XML documents. In: Abbadi AE, Brodie ML, Chakravarthy S, Dayal U, Kamel N, Schlageter G, et al., editors. Proceedings of 26th International Conference on Very Large Data Bases; 2000 September 10–14. Cairo, Egypt: Morgan Kaufmann; 2000. p. 133–54.

[28] Buneman P, Davidson S, Fernandez MF, Suciu D. Adding structure to unstructured data. In: Afrati FN, Kolaitis P, editors. 6th International Conference on Database Theory; 1997 January 8–10. Delphi, Greece: Springer; 1997. p. 336–50.

[29] Papakonstantinou Y, Garcia-Molina H, Widom J. Object exchange across heterogeneous information sources. In: Yu PS, Chen ALP, editors. Proceedings of the Eleventh International Conference on Data Engineering. Taipei, Taiwan: IEEE Computer Society; 1995. p. 251–60.

[30] International Telecommunication Union (ITU). ASN.1. Available from: http://www.itu.int/ITU-T/asn1/.

[31] Clark J, DeRose S. XML Path Language (XPath) Version 1.0. World Wide Web Consortium (W3C®). Available from: http://www.w3.org/TR/xpath.

[32] Cattell RGG, Barry DK. The object data standard: ODMG 3.0. Burlington, MA: Morgan Kaufmann; 2000.

[33] Buneman P, Fernandez MF, Suciu D. UnQL: a query language and algebra for semistructured data based on structural recursion. VDLB J 2000;9(1):76–110.

[34] Clark J. XSL Transformations (XSLT) Version 1.0. World Wide Web Consortium (W3C®). Available from: http://www.w3.org/TR/xslt.

[35] Robie J, Chamberlin D, Florescu D. Quilt: an XML Query Language. Available from: http://www.almaden.ibm.com/cs/people/chamberlin/quilt_euro.html.

[36] Rys M. Bringing the internet to your database: using SQL server 2000 and XML to build loosely-coupled systems. In: Proceedings of the 17th International Conference on Data Engineering; 2001 April 2–6. Heidelberg, Germany: IEEE Computer Society; 2000. p. 465–72.

[37] Fernandez MF, Tan W-C, Suciu D. SilkRoute: trading between relations and XML. In: Ninth International World Wide Web Conference; 2000 May 15–19. Amsterdam, The Netherlands; 2000.

[38] Kim S, Brinkley JF, Rosse C. A profile of on-line anatomy information resources: design and instructional implications. Clin Anat 2003;16(1):55–71.

[39] Shanmugasundaram J, Tufte K, Zhang C, He G, DeWitt DJ, Naughton JF. Relational databases for querying XML documents: limitations and opportunities. In: Atkinson MP, Orlowska ME, Valduriez P, Zdonik SB, Brodie ML, editors. Proceedings of 25th International Conference on Very Large Data Bases; 1999 September 7–10. Edinburgh, Scotland, UK: Morgan Kaufmann; 1999. p. 302–14.

[40] Gene Ontology™ Consortium (GO). Gene Ontology (GO). Available from: http://www.geneontology.org/.

[41] Patel-Schneider P, Siméon J. The Yin/Yang web: XML syntax and RDF semantics. In: Eleventh International World Wide Web Conference; 2002 May 7–11. Honolulu, Hawaii: ACM; 2002. p. 443–53.

[42] Bradley SW, Eno K, Prothero J, Brinkley JF. Interactive Atlas Software. Seattle, WA: University of Washington. Available from: http://www9.biostr.washington.edu/da.html.