Ontology-based Queries over Cancer Data

Alejandra González-Beltrán[1,2], Ben Tagger[1], and Anthony Finkelstein[1]

[1] Department of Computer Science
[2] Computational and Systems Medicine
University College London, London WC1E 6BT, United Kingdom

Abstract. The ever-increasing amount of data in biomedical research, and in cancer research in particular, needs to be managed to support efficient data access, exchange and integration. Existing software infrastructures, such *caGrid*, support access to distributed information annotated with a domain ontology. However, caGrid's current querying functionality depends on the structure of individual data resources without exploiting the semantic annotations. In this paper, we present the design and development of an ontology-based querying functionality that consists of: the generation of OWL2 ontologies from the underlying data resources metadata and a query rewriting and translation process based on reasoning, which converts a query at the domain ontology level into queries at the software infrastructure level. We present a detailed analysis of our approach as well as an extensive performance evaluation. While the implementation and evaluation was performed for the *caGrid* infrastructure, the approach could be applicable to other model and metadata-driven environments for data sharing.

Keywords: ontology, query, caGrid, UML, OWL2, pattern, module extraction

1 Introduction

In the biomedical sciences, the use, exchange and integration of the ever-increasing amount of data has become paramount to accelerate the discovery of new approaches for the detection, diagnosis, treatment and prevention of diseases. In particular, this applies to cancer, for which the US National Cancer Institute (NCI) and the UK National Cancer Research Institute (NCRI) have implemented Informatics Initiatives focusing on building and deploying software infrastructure to manage and analyse data generated from heterogenous data sources.

In this paper, we provide an analysis of the caGrid[1] software infrastructure developed within the NCI caBIG®[3] programme and extend it with richer querying capabilities. caGrid supports a collaborative information network for sharing cancer research data, and deals with syntactic and semantic interoperability of the data resources in a service-oriented model-driven architecture. Semantic interoperability is achieved by using a metadata registry, which maintains information models annotated with concepts from a domain ontology: the

[3] caBIG® stands for cancer Biomedical Informatics Grid® .

NCI thesaurus (NCIt)[2]. However, the query functionality provided in caGrid does not take into account the semantic annotations, but it only relies on each individual information model.

Our methodology is based on extending the caGrid service-oriented model-driven infrastructure with additional services to support ontology-based queries over the distributed data resources. In this way, the biomedical researchers, as the end-users of our system, will be able to query cancer data by building queries using their domain knowledge (expressed as concepts from the NCIt ontology) rather than having to know the underlying models. This also means that the queries are reusable across resources, which is not the case in the caGrid infrastructure. This functionality will be incorporated into the NCRI ONcology Information eXchange (ONIX[4]). Our approach involves a customised transformation from annotated information models to an ontological representation using the Web Ontology Language version 2 (OWL[5]). This representation supports annotations based on a primary concept and a list of qualifiers. Based on these ontological representations of the data resources, we have designed and developed a query rewriting and translation approach that converts concept-based queries into the query language supported by the caGrid infrastructure. This approach is general and could be used to support other target query languages, as the only step dependent on caGrid is the last one. This work presents significant improvements over our previous work[3], as we have significantly modified and improved the OWL representation and the design and implementation of the query rewriting and translation steps. We have developed a caGrid analytical service for the transformation from an annotated information model to OWL. Additionally, we present an analysis of the caGrid query language and information together with an extensive performance evaluation that justifies the applicability of our solution.

This paper is structured as follows. Section 2 introduces background material on the caGrid infrastructure. Section 3 presents an analysis of the caGrid query functionality and the type of queries supported by its query language. Then, we present in section 4.1 the OWL representation that is used for query rewriting and translation, which in turn is described in Section 4.2. The implementation details and performance evaluation results are given in Sections 4.3 and 4.4, respectively. The evaluation includes an analysis of the generated ontologies as well as several performance metrics for OWL generation and query rewriting, which justify the viability of our approach. After comparing our approach with related work in Section 5, we conclude the paper in Section 6, including considerations for future work.

[4] http://www.ncri-onix.org.uk/

[5] OWL is a recommendation from the World Wide Web Consortium (W3C) and the language overview for its second version can be found at http://www.w3.org/TR/owl2-overview/

2 Background

caBIG®[1] is an NCI programme whose aim is to create a virtual and federated informatics infrastructure for sharing data, tools and connect scientists and organisations in the cancer research community. The computing middleware in caBIG® is called caGrid, which is a Grid[4] extended to support data modelling and semantics[1]. caGrid has a number of core services and corresponding application programming interfaces (APIs), which we will introduce next, by analogy with the metadata hierarchy[5], as per Figure 1.

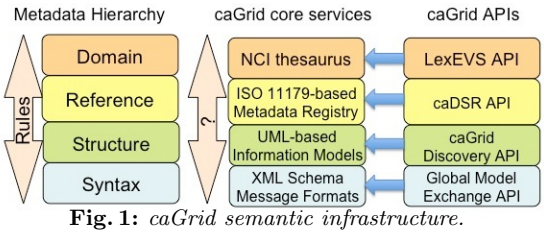

Fig. 1: *caGrid semantic infrastructure.*

The metadata hierarchy represents how the semantics of raw data (*instance data*) can be augmented by overlaying metadata of increasing descriptiveness [5]. The *syntactic metadata* refers to the language format and data types, and in caGrid is represented by XML schemas managed by the Global Model Exchange (GME)[1] service, which exposes them through the GME API[6]. The *structural metadata* gives form to the units of data. In caGrid, it is implemented as an object-oriented virtualisation of the underlying data resources[1] and it is represented as UML[7] models. These UML models can be accessed through the Discovery API[8]. The purpose of the *referent metadata* is to represent the linkages between the different data models. In caGrid, the linkages are provided by a metadata registry, called caDSR[9], based on the ISO/IEC 11179 standard[10]. caDSR manages common data elements (CDEs) and exposes them through the caDSR API. The *domain metadata* represents what the data is about. It is implemented by a domain conceptualisation, usually in the form of an ontology[5]. In the caGrid case, the NCIt ontology[2] is used, accessed via the LexEVS API[11]. Finally, the rules constitute an overarching layer that can be applied to all the aforementioned layers. Rules can be used to constrain and extend the semantics of metadata specifications at any of the abstraction levels[5]. In the current caGrid semantic infrastructure, there is no equivalent to the *rule metadata*.

A data resource owner can share the data by developing caGrid data services using common interfaces and metadata, as described above. In this way, a data service encapsulates the data, which is kept in native formats (including, for example, relational data or flat files), exposing an access interface based on the object-oriented (UML) model of the underlying resource. The common interface also exposes a query processor based on the Common Query Language (CQL) defined for caGrid. CQL is an object-oriented query language reflecting the underlying object model of the data resource while abstracting the physical representation of the data[1]. At the time of writing, there exist two versions of

[6] http://cagrid.org/display/gme/
[7] UML stands for the Unified Modeling Language, a specification of the Object Management Group®(OMG®)
[8] http://cagrid.org/display/metadata13/Discovery
[9] caDSR stands for cancer Data Standards Repository
[10] http://metadata-stds.org/11179/
[11] https://cabig.nci.nih.gov/concepts/EVS/

CQL and there is a pre-release version of the latest one[12]. More details on CQL are given in Section 3.

caGrid also supports basic distributed aggregations and joins of queries over multiple data services by means of the caGrid Federated Query Infrastructure[13], through a distributed extension of CQL called DCQL. Thus, caGrid relies on D/CQL – custom query languages based on the structural characteristics of the resources. In other words, caGrid builds a 'structural layer', where queries are expressed in terms of objects, attributes and associated objects, without allowing for semantic queries. D/CQL are evolving to provide richer structural queries as new requirements arise from different caBIG® projects. However, these query languages do not allow for data extraction based on semantic information. Thus, a shortcoming of caGrid is that does not currently exploit the *referent* and *domain metadata* maintained for its data services. Additionally, as already mentioned, it is not possible to specify *rules* about the models nor the domain semantics.

As stated in the introduction, this work advocates the extension of the caGrid infrastructure to exploit its rich metadata by building a semantic layer, using semantic web technologies to exploit caGrid's metadata. Additionally, this extension is capable of: *a*) accommodating other resources with different ways of dealing with metadata, and *b*) specifying rules at different levels of abstraction.

3 Analysis of the caGrid Query Language

A CQL query is defined by an XML document, which must comply to a specified XML schema[14]. The schema indicates that a CQL query must specify a ⟨Target⟩ element, which is the data type of the query result. Optionally, an ⟨Attribute⟩ element might indicate a predicate over an attribute of the object with ⟨Target⟩ type and an ⟨Association⟩ may specify a link with a related object. In Table 1 we show how a CQL query is built recursively presenting it as a context-free grammar, where ⟨CQLQuery⟩ is the start symbol, ϵ is the empty string and $\langle_{\text{xsd:string}}\rangle$ is the non-terminal variable representing the XSD:string data type.

So, CQL traverses the UML class diagram graph, where the ⟨Target⟩ is the initial class, the ⟨Association⟩ conditions allow for path navigation by traversing sequences of consecutive classes and ⟨Attribute⟩ conditions apply locally to individual classes. The terminal symbols ⟨Group⟩ and ⟨Group1⟩ represent the combination of two or more constraints over a particular node in the UML class graph.

4 Ontology-based queries over the caGrid infrastructure

As shown before, the caGrid queries rely on the structure of the underlying data resources, i.e. their UML models. Thus, a biomedical researcher interested in

[12] http://cagrid.org/display/dataservices/CQL+2
[13] http://cagrid.org/display/fqp/Home
[14] The CQL XML schema is available at: http://cagrid.org/display/dataservices/CQL+Schemas

⟨CQLQuery⟩ →⟨Target⟩ |
⟨Target⟩ ⟨QueryModifier⟩
⟨Target⟩ →⟨Name⟩ ⟨Attribute⟩ |
⟨Name⟩ ⟨Association⟩ |
⟨Name⟩ ⟨Group⟩
⟨Attribute⟩ →⟨Name⟩ ⟨Predicate⟩ ⟨Value⟩
⟨Association⟩ →⟨RoleName⟩
⟨Group⟩ →⟨LogicalOp⟩ ⟨Attribute⟩ ⟨Group1⟩ |
⟨LogicalOp⟩ ⟨Association⟩ ⟨Group1⟩ |
ϵ
⟨Group1⟩ →⟨Attribute⟩ ⟨Group⟩ |
⟨Association⟩ ⟨Group⟩ |
⟨Group⟩

⟨LogicalOp⟩ →**AND** |**OR**
⟨Predicate⟩ →**EQUAL_TO** |**NOT_EQUAL_TO** |
LIKE |**IS_NULL** |
IS_NOT_NULL |**LESS_THAN** |
LESS_THAN_EQUAL_TO |
GREATER_THAN |
GREATER_THAN_EQUAL_TO
⟨Name⟩ →⟨xsd:string⟩
⟨RoleName⟩ →⟨xsd:string⟩
⟨Value⟩ →⟨xsd:string⟩
⟨QueryModifier⟩ →⟨DistinctAttribute⟩ |
⟨DistinctAttribute⟩ ⟨AttributeNames⟩

Table 1: *CQL query context-free grammar*

retrieving data about, for example, a particular gene of interest will need to explore the UML model of each relevant data service and build a query considering the specific attributes and associations of the class maintaining the *Gene* objects. The queries can be built programmatically or also through the caGrid portal[15], which allows to explore the UML models and provides a query builder based on these models.

In this work, we propose a system that allows the user to concentrate on the concepts from the domain, as represented by the NCIt ontology on cancer, and build the ontology-based queries which are *high-level*[16] and *descriptive*[17]. Thus, the ontology-based queries can be applicable to any of the underlying data resources.

Apart from the cancer concepts found on NCIt, the queries combine elements from an ontology we built with metadata on UML models[18], namely the *UML model* ontology. This ontology contains OWL classes to represent UML classes and attributes (*UMLClass*, *UMLAttribute*), OWL object properties to represent UML associations and the relationship between a UML class and its attributes (*hasAssociation*, *hasAttribute*) and a data property to represent the values of attributes (*hasValue*).

Some simple example queries[19] are: *a*) *Specimen* to retrieve all the objects that are annotated with the Specimen concept; *b*) *Gene and hasAttribute some*

[15] http://cagrid-portal.nci.nih.gov
[16] By a high-level query, we mean a query that can be written without specific details about the structure of the target resource.
[17] By a descriptive query, we refer to queries that provide the criteria for the desired data rather than the procedure to find the data.
[18] We will see later, than the queries could also use elements from the list ontology[6].
[19] The example queries are given in Manchester OWL Syntax and are just intended to show queries retrieving objects from a UML class, a UML class with a condition over

(Gene_Symbol and hasValue value "BRCA%") to find all the genes whose symbol starts with the string BRCA; *c) Single_Nucleotide_Polymorphism and hasAssociation some (Gene and hasAttribute some (Gene_Symbol and hasValue value "TGFB1"))* to obtain all the *SNPs* associated with the *Gene Transforming Growth Factor Beta 1*[3]. In our system, these queries could be submitted to any data service, and they will be converted to the specific CQL query.

We note that the third query specifies SNPs that are associated with genes. This association may be present in different ways in two separate UML models. For example, the two corresponding classes may have a direct UML association, or the association may arise by traversing an association path from the first class to the second one. In order for our system to deal with those paths of associations, without the user requiring to know the specific underlying UML model, we define the *hasAssociation* property as transitive and use reasoning to determine the paths.

Next, we introduce our transformation from caGrid models to an OWL2 representation and the query rewriting/translation approach, which transforms ontology-based queries into CQL queries. The OWL2 ontologies provide an unified view of the UML models and their semantic annotations, which allows us to apply reasoning over them.

4.1 OWL Representation of caGrid Information Models

OWL model of UML class diagrams. First, we present our customised UML-to-OWL transformation. This transformation differs from previous approaches, as explained in Section 5. Next, we describe the transformation and use the portion of the caBIO 4.2 information model in Figure 2 to give examples. Every UML element is related to its counterpart in the *UML model* ontology: all UML classes and attributes are defined as subclasses of *UMLClass* and *UMLAttribute*, respectively (see equations Eq. 1 and Eq. 2 below[20]); all the UML associations are sub-properties of *hasAssociation* (Eq. 4), and the datatype property *hasValue* is used to specify the type of the attributes (Eq. 3) as an existential restriction. Contrary to other UML-to-OWL transformations, we represent UML attributes as OWL classes. This is required so that the ontology-based queries can include the concepts associated with attributes.

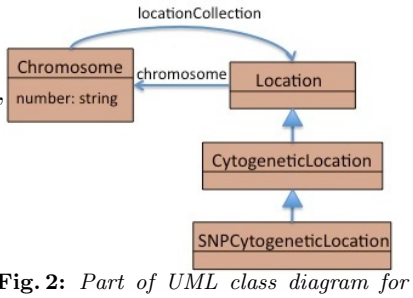

Fig. 2: *Part of UML class diagram for caBIO 4.2*

$$\text{c:Chromosome} \sqsubseteq \text{u:UMLClass} \quad (1)$$

$$\text{c:Chromosome\_number} \sqsubseteq \text{u:UMLAttribute} \quad (2)$$

$$\text{c:Chromosome\_number} \sqsubseteq \exists\, \text{u:hasValue.xsd:string} \quad (3)$$

$$\text{c:Chromosome\_locationCollection\_Location} \sqsubseteq \text{u:hasAssociation} \quad (4)$$

UML subclass and superclass relationships are represented with subsumption (Eq. 5). For each UML class, existential restrictions are added for its associa-

an attribute and two associated UML classes with a restriction over an attribute of one of the classes, respectively.

[20] The prefixes used in the equations are: *c:* for the caBIO 4.2 ontology, *u:* for the UML model ontology, *n:* for the NCIt ontology and *l:* for the list ontology. We note that the name of an OWL class corresponding to an attribute includes the class name to avoid duplications and for associations, it includes its domain and range.

tions (Eq. 6) and attributes (Eq. 7). While UML does not explicitly represent inherited associations, our OWL representation makes them explicit, modeling the semantics of UML. For example, as the UML class *Location* has an association *chromosome* with the class *Chromosome*, this association is inherited on the subclass *CytogeneticLocation* (Eq. 8).

$$\text{c:CytogeneticLocatoin} \sqsubseteq \text{c:Location} \tag{5}$$

$$\text{c:Chromosome} \sqsubseteq \exists\, \text{c:Chromosome\_locationCollection\_Location.c:Location} \tag{6}$$

$$\text{c:Chromosome} \sqsubseteq \exists\, \text{u:hasAttribute.u:Chromosome\_number} \tag{7}$$

$$\text{c:CytogeneticLocation} \sqsubseteq \exists\, \text{c:Location\_chromosome\_Chromosome.c:Chromosome} \tag{8}$$

We note that the generated OWL ontologies belong to OWL2EL[7], an OWL2 profile specifically designed to allow for efficient reasoning with large terminologies, which is polynomial in the size of the ontology. While OWL2EL disallows universal quantification on properties, it does allow the inclusion of transitive properties. Thus, it is suitable for our UML-to-OWL transformation customised for the rewriting approach as outlined before.

OWL Representation of the Semantic Annotations. Apart from representing the UML model, we also model its mapping to NCIt, as maintained in caDSR. Through the CDEs, UML elements are annotated with a primary concept, which indicates the meaning of the element. In turn, a list of qualifier concepts may be used to modify the primary concept, giving specific meaning. As OWL2 does not natively supports the representation of lists, we used Drummond *et al*'s design pattern for sequences[6] to model primary concepts and qualifier lists. The following equations give some examples on how the semantic annotations of UML classes (Eq. 9) and attributes (Eq. 10) with a single concept are modelled. Equation 12 models the class *c:SNPCytogeneticLocation* as being a *n:Location* qualified with *l:Chromosome_Band* and *n:Single_Nucleotide_Polymorphism*.

$$\text{c:Chromosome} \sqsubseteq \text{n:Chromosome} \tag{9}$$

$$\text{c:Chromosome\_numer} \sqsubseteq \text{n:Name} \tag{10}$$

$$\text{c:SNPCytogeneticLocation} \sqsubseteq \text{n:Location} \sqcap (\text{l:OWLList} \sqcap \exists \text{l:hasContents.n:Chromosome\_Band} \sqcap \exists \text{l:hasNext.}(\text{l:OWLList} \sqcap \exists \text{l:hasContents.n:Single\_Nucleotide\_Polymorphism})) \tag{11}$$

Module Extraction from NCI Thesaurus Ontology. The NCIt ontology is very large, as it provides a common vocabulary for the whole cancer domain[2]. Each caGrid data service is, in general, concerned with data pertaining to more specific domains than the whole NCIt ontology. Thus, for each caGrid data service referring to a subset Σ of the NCIt vocabulary, there is a subset of terms and relationships from NCIt that is *relevant*, called a *module* from the ontology[8]. The module $\mathcal{M}$ represents all knowledge about the terms of the *signature* Σ. One of the approaches to*relevance* is logic-based: the module $\mathcal{M}$ is relevant for the terms Σ if all the consequences of the ontology that can be expressed over Σ are also consequences of $\mathcal{M}$[8]. We follow that approach by

Sattler *et al* [8] and extract an NCIt module for each of the information models in caGrid. For succinctness and efficiency, this module is used, as opposed to the whole NCIt ontology, for the semantic annotations of UML models and subsequent reasoning. However, we observe that we removed the disjoint axioms from the NCIt modules, as we noted before[3,9] using subsumption to represent UML class to concept mapping may result in inconsistent ontologies as the annotations for a single class may come from two high-level branches in NCIt that are declared as disjoint.

4.2 Query Rewriting and Translation

This section describes how an ontology-based query is rewritten and then translated, first to an intermediate optimisation language and then to the target CQL language. While the overall approach is similar to our previous work[3], previously we relied completely on justifications[10] and now we have extensively improved the approach by dealing with each of the steps independently. We provide the output of each step for the third query from Section 4.

Parsing. First, the user query is syntactically parsed. The query uses concepts from the NCIt, the UML model (see Section 4.1) and the list ontologies[6].

UML Extraction. The NCIt concepts in the query are translated into specific UML classes, by reasoning over the generated ontologies. Each concept is the super-class of a UML class or UML attribute, depending on their position on the query. Often, a single NCIt concept will correspond to many UML classes (or attributes) and, in such cases, each UML class is returned to form an individual query. Therefore, the outcome of the UML extraction is a combination of possible queries given the extracted UML classes or attributes. The outcome for our example query is: *c:SNP and hasAssociation some (c:Gene and hasAttribute some (c:Gene_symbol and hasValue value "TGFB1"))* .

Data Values Extraction. As the generated ontologies do not contain instances, the semantic validation of the query, expressed as an OWL class expression, must ignore the data expressions. This step extracts the data expressions, which will be reinserted later on. This step results in *c:SNP and hasAssociation some (c:Gene and hasAttribute some (c:Gene_symbol))*.

Semantic Validation. We use a reasoner to check that the resulting query can be satisfied. If the query cannot be satisfied, subsequent rewriting of the query is halted.

Properties Path Finder. This step deals with the ontology corresponding to the UML model (the semantic annotations do not need to be considered any longer) and aims at finding the path of UML classes related through the transitive property *hasAssociation*[21]. The path finder rewrites the expression using non-transitive properties, corresponding to UML associations, by using an explanation generator[10] that retrieves the justification for two classes to be

[21] We note that the ontology is compliant with the OWL2 EL profile, as OWL2 EL supports the use of transitive object properties. For more information, see http://www.w3.org/TR/owl2-profiles/

connected via the transitive property, and thus allowing to find the intermediate classes. The path finder may find more than one path between a set of nodes and, in such cases, will return each path as a combination of possible queries for user selection. One path for our example query is: *c:SNP and hasAssociation some c:GeneRelativeLocation and hasAssociation some (c:Gene and hasAttribute some (c:Gene_symbol))*.

Data Values Addition. At this point, we can retrieve the data expressions removed earlier and re-insert them into the corresponding OWL classes, resulting in *c:SNP and hasAssociation some c:GeneRelativeLocation and hasAssociation some (c:Gene and hasAttribute some (c:Gene_symbol and hasValue value "TGFB1"))*.

OWL Expression to MCC Translation. No calculus or algebra has been defined for the object-oriented query language CQL. To provide a translation with CQL as target language, we use the monoid comprehension calculus (MCC), as it is a formal framework to support object queries optimizations[11]. Object queries involve collections (e.g. sets, lists, bags), whose semantics can be captured by monoid comprehensions (MC). In this paper, we only overview MCs and its use in our system[22]. Our approach is similar to the work by Peim *et al* [12], but while they use an expansion algorithm to rewrite an OWL expression based on a set of acyclic set of definitions, we follow the specific steps described above.

A MC takes the form $\oplus\{e \parallel \overline{q}\}$, where $\oplus$ is a monoid[23] operator called the *accumulator*, e is the *header* and $\overline{q} = q_1, \ldots, q_n, n \geq 0$ is a sequence of *qualifiers*. A qualifier can take the form of a *generator*, $v \leftarrow e'$ with v a range variable and e' an expression constructing a collection, or a *filter* predicate. The symbol $\uplus$ denotes the accumulator for bags[24]. For an OWL class expression from the previous step, an MCC expression is built such that: the header variable is determined by the first concept in the query and the qualifiers are built for each of the remaining expressions. The MCC expression for our example is: $\uplus$ { *s* $\parallel$ *s* $\leftarrow$ *SNP, r* $\leftarrow$ *s.relativeLocationCollection, r* $\leftarrow$ *GeneRelativeLocation, g* $\leftarrow$ *r.gene, g* $\leftarrow$ *Gene, g.symbol=TGFB1* }

MMC to CQL Translation. Translating the MCC expression into CQL amounts to: define as *Target* the type of the variable that appears in the header and then, including an *Association* per each pair of generators, one determining the name (the class to which they belong) and the other identifying the role name; include an *Attribute* restriction for each filter. As this last step is the only one involving CQL, only this last step requires to be modified to extend our methodology to other model-driven architectures with a different target language. The resulting CQL in the example is:

```
<ns1:CQLQuery xmlns:ns1="http://CQL.caBIG/1/gov.nih.nci.cagrid.CQLQuery">
<ns1:Target name="gov.nih.nci.cabio.domain.SNP">
```

[22] For more details, we refer the reader to [11] and [3]

[23] A monoid of type T is an algebraic structure defined by $(\oplus, Z_\oplus)$ where $\oplus : T \times T \rightarrow T$ is an associative funcion and $Z_\oplus$ is the left and right identity of $\oplus$. A collection monoid is a monoid for a collection type (e.g. lists or bags) and must also specify a unit function building a singleton collection.

[24] For example, $\uplus\{x \parallel x \leftarrow \{1,2\}\}$ is the monoid comprehension representing the bag $\{\{1,2\}\}$.

```
10
  <ns1:Association name="gov.nih.nci.cabio.domain.GeneRelativeLocation"
  roleName="relativeLocationCollection">
   <ns1:Association name="gov.nih.nci.cabio.domain.Gene" roleName="gene">
    <ns1:Attribute name="symbol" predicate="EQUAL_TO" value="TGFB1"/>
   </ns1:Association>
   </ns1:Association>
</ns1:Target>
 </ns1:CQLQuery>
```

4.3 Implementation

We have implemented two modules, with the functionalities: *a*) an OWL generator, which transforms a caGrid annotated UML model into an OWL ontology and includes the generation of a module from the NCIt containing the concepts relevant to the UML model; *b*) a query translation component, which takes as input a OWL class expression using concepts from the NCI thesaurus and transforms it into a CQL for a single data service.

For the first module, we also produced a caGrid analytical service called OWLGenService[25], which provides a simple API for the extraction of modules from NCIt and for the ontology generation, given a specific information model.

The implementation was done in Java and uses caGrid version 1.3[26], the OWLAPI version 3.1.0[27] (after upgrading from OWLAPI version 2), and relies on the reasoners Pellet 2.2.2[28] and HermiT 1.3.0[29].

4.4 Performance Evaluation

This section analyses the generated ontologies and presents two areas of performance evaluation that verify the viability of our approach. Since one important step in the query rewriting/translation process is the *property path finder* (see Section 4.2), we firstly introduce some metrics to assess the paths in the generated ontologies. These paths are sequences of concepts linked by object properties. Secondly, we present the generation times for the module extraction, the ontology generation and the inference of the ontologies using both the Pellet and HermiT reasoners. These results show that the generation of the ontologies that make possible our approach is done in a timely fashion. Thirdly, we evaluate the performance of the query rewriting process, showing a breakdown of the constituent parts of the rewriting algorithm. For this evaluation, we considered two sets of five queries each run over the caBIO data service[30], where each set consists of queries that involve paths of lengths one and two. The tests were run

[25] The OWLGenService is accessible through the caGrid portal at http://cagrid-portal.nci.nih.gov and available at http://stylus_157.stylusinternet.net:9600/wsrf/services/cagrid/OwlgenService

[26] http://wiki.cagrid.org/display/caGrid13/Home

[27] http://owlapi.sourceforge.net/

[28] http://clarkparsia.com/pellet/

[29] http://hermit-reasoner.com

[30] http://cabiogrid42.nci.nih.gov:80/wsrf/services/cagrid/CaBIO42GridSvc

on a Red Hat Enterprise Linux Server release 5.3 (Tikanga) with 64 bits and 48285 MB of RAM.

This section analyses the generated ontologies and presents two areas of performance evaluation that verify the viability of our approach. Since one important step in the query rewriting/translation process — from Section 4.2 — is the *property path finder*, we firstly introduce some metrics to assess the sequences of concepts linked by object properties (paths) in the generated ontologies. Secondly, we present the generation times for the module extraction, the ontology generation and the inference of the ontologies using both the Pellet and HermiT reasoners. These results show that the generation of the ontologies that make possible our approach take a short time. Thirdly, we evaluate the performance of the query rewriting process with a breakdown of the constituent parts of the algorithm. For this evaluation, we considered two sets of five queries each, where each set consists of queries that involve paths of lengths one and two. The results were obtained by running on a Red Hat Enterprise Linux Server release 5.3 (Tikanga) with 64 bits and 48285 MB of RAM.

Throughout this section, we have grouped caGrid projects into three distinct subsets: projects that are available from the *caDSR* service, all data services that are registered with the *caGrid* default index service[31], and *Information Models* (or *InfoModels*) (those models that are supported by a deployed service from the *caGrid* Index Service)[32]. We note that the groups *caGrid* and *InfoModels* are the more relevant for our system, as only against these projects it is possible to execute CQL queries. While *InfoModels* include a single project from caDSR for a set of deployed services corresponding to that project, *caGrid* may include the results for several services that correspond to a single model. Thus, the *caGrid* results will be skewed according to the relative weight of services as opposed to models.

Analysis of the OWL representation of the information models.

While ontology metrics have been defined in several tools [13], these have focused on basic metrics (e.g. number of classes) and semantic-based metrics (e.g. relationship richness) that allow for the comparison and quality evaluation of the ontologies. Here, we will focus on the presentation of some bespoke metrics we developed to measure the proliferation and complexity of paths within the ontologies, as these will ensure the viability of our approach.

As seen in Section 4.2, our rewriting process seeks to remove the upper-level and transitive object property *hasAssociation* and express the query using only non-transitive properties, which correspond to the UML associations in the models. In order to achieve this, we consider the paths between pairs of concepts

[31] http://cagrid-index.nci.nih.gov:8080/wsrf/services/DefaultIndexService

[32] It should be noted that not all caDSR projects are included in the metrics; some contained errors (their semantic metadata is not complete or refers to an older version of the NCI thesaurus) and some models are targeted for data modelling, rather than specifically holding data, making them not representative for our system. Out of the 136 projects in caDSR, 16 were excluded from the analysis for these reasons. However, none of the excluded projects had an associated service. Additionally, the *caGrid* subset has 63 services and *InfoModels* has 23 projects.

from the query connected through the *hasAssociation* property. The calculation of these paths is not trivial; there may be many intermediate nodes and there may be more than one path for a given pair of concepts. We define a *journey* as a traversal from one concept to another. A *journey* may have one or many paths, which represent the possible routes that the traversal can take. Thus, it is important to evaluate these aspects of the ontologies in order to assess the viability of a rewriting tool.

We propose the following metrics as a measure of complexity in this respect. The *Longest Path* is the maximum path length that may be computed within a given ontology. The longest path length provides an indication of the worse case for path calculation times. The *Average Paths per Journey* reflects the degree of path expansion within the rewriting algorithm, as each journey (e.g. from Node A to Node B) may have many different paths. The rewriting algorithm should return all possible paths as each path may refer to a different expression of the query. When we consider that a single query may include multiple independent journeys, the possible query rewritings can become very large. The *Average Nodes per Path* is the average number of nodes that must be visited in order to return a single path. The path length can affect the calculation time as well as the complexity of the resulting query.

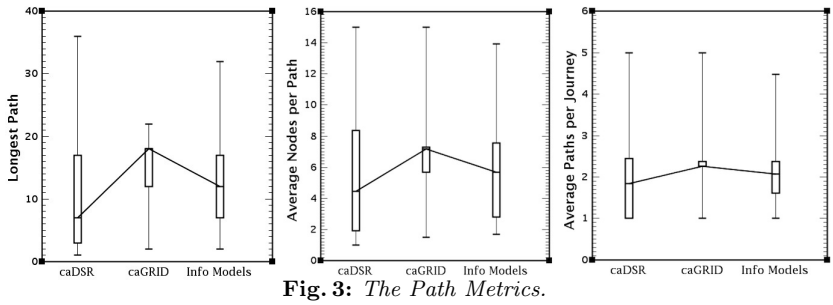

Fig. 3: *The Path Metrics.*

Figure 3 illustrates three box-and-whisker plots with the results of the path metrics for each project subset. We observe that while the longest path can have up to 36 nodes, for 75 % of the projects in each category their length is less than 17 or 18. The average path length varies between 4 and 7 nodes over the three subsets, and for 75 % of the *InfoModels* it is less than 8. There is an average of around 2 paths returned for each journey, and for 75 % of the projects in each category the average path per journey is less than 2.5, indicating that we will be returning a low number of path combinations as a result. These results, then, verify that the paths within the ontologies are manageable and appropriate for our rewriting tool. We also note that in all the metric diagrams, the caGrid subset is often very densely clustered around the mean. This is due to the fact that there are often many caGrid services for the same project that differ to one another very slightly or not at all, which can result in multiple similar or identical results.

Ontology Generation, Module Extraction and Classification. In order to isolate any overhead caused by variations in network performance, we extracted the XML corresponding to each project (or information model) in caDSR. This is a preliminary step to run the performance evaluator locally, and we do not include any data about the performance of this stage. We generate four ontologies for each project: the NCIt module ontology (incorporating the concepts from NCIt relevant to the project), the annotated UML ontology (including the classes describing the UML model) and inferred versions of the two ontologies[33]. We recorded the time for each generation and Figure 4 shows them for the four ontologies per project grouped by subset. The times are presented in a logarithmic scale. We can see that the vast majority (75%) of NCIt modules take less than 2 seconds to generate and even less time for ontology generation. The classification of the generated ontologies is also timely, with the average inference of the Pellet and HermiT reasoners never taking longer than 100 milliseconds.

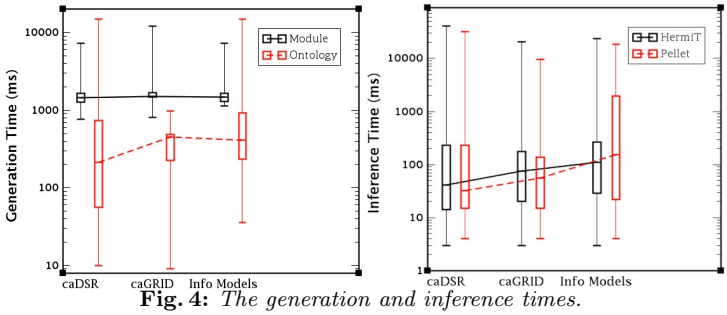

Fig. 4: *The generation and inference times.*

Query Rewriting Evaluation. We have developed a suite of queries of varying complexity in order to evaluate the query rewriting. The results are presented in figure 5, which shows the average time[34] taken at each stage of the query rewriting process[35] for five queries whose rewriting has path length one, five queries whose rewriting has path length two and the mean times for these two sets. Path length refers to the number of intermediate nodes in the query resulting from the rewriting. We can see from figure 5 that, while path length has a marked effect on the time taken at the path finding stage, the other stages of implementation remain largely unaffected. We therefore maintain that, given our analysis of paths within our target ontologies described above, we can provide query rewriting in a timely and efficient manner.

[33] We generate the inferred ontologies classifying the generated ontologies using both the HermiT and Pellet reasoners.

[34] Each query was ran 5 times and the average times calculated.

[35] These correspond to the stages of query rewriting; parsing, UML extraction, validation, path finding, MCC conversion and CQL conversion. For more information, refer to section 4.2.

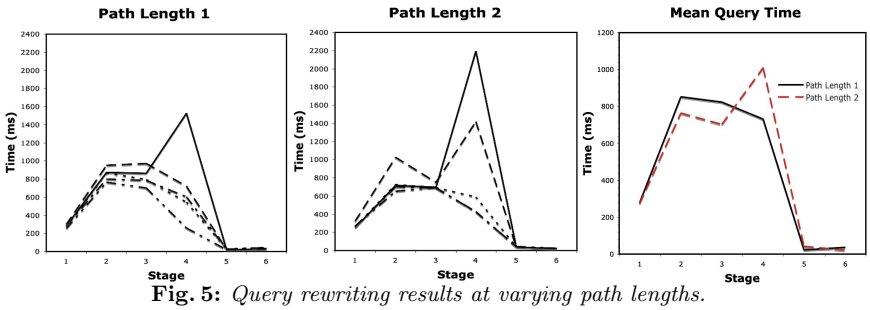

Fig. 5: *Query rewriting results at varying path lengths.*

5 Related Work

The UML-to-OWL transformation has been studied in different contexts and applications varying from the detection of inconsistencies in UML diagrams to use as interchangeable modeling artifacts[14,15]. We have also provided different alternative transformations before[3,9] and have improved it here so that the UML transformation conforms with OWL2EL profile, where the semantic annotations use subsumption and additionally, primary concepts and qualifiers are modelled as sequences.

The use of semantic web technologies to support semantic queries over distributed data environments in biomedicine have been implemented in systems such DartGrid[16] (for traditional chinese medicine), ACGT[17] and semCDI[18] (for cancer). To the best of our knowledge, the latest is the only work over the caGrid infrastructure. All these systems support SPARQL queries over the resources, while our system allows for high-level descriptive queries which do not need to be based on the structure of particular resources. Additionally, our approach using MCC as an intermediary language provides support for optimisations and generality, as a different target language could be used, even SPARQL.

6 Conclusions

This paper presented the design and implementation of an ontology-based querying functionality implemented over a service-oriented, model-driven infrastructure aimed at sharing cancer research data. In particular, the implementation was based on the caGrid infrastructure, but the approach could be used over similar model-driven software infrastructures. We presented: *a*) the generation of customised OWL2 ontologies from annotated UML models, based on the ISO11179 standard for metadata registries, which differs from traditional UML-to-OWL conversions and it is an improvement from[3], mainly as we now generate OWL2EL ontologies for the UML models and support annotations with primary concept and qualifiers; *b*) an analysis of the generated ontologies by determining several relevant ontology metrics, existing and new metrics that justify the viability of our rewriting technique; *c*) a caGrid analytical service implementing the OWL Generation facility; *d*) an analysis of the capabilities of

the caGrid query language, and the queries it supports; *e*) a significant revision and improvement of the query rewriting and translation steps to transform a domain ontology-based query into CQL; *f*) an extensive performance evaluation of the OWL generation and module extraction, plus an assessment of the querying rewriting and translation process and its viability. In future work, we will extend the query rewriting/translation evaluation providing a varied query set, explore the use of an OWL2EL reasoner to improve performance of the path finding process and support federated queries across data resources, where the selection of join conditions will be provided by a semantic analysis of the distributed resources.

Acknowledgements The authors are grateful to the National Cancer Research Institute Informatics Initiative for support for their research.

References

1. Saltz *et al* . caGrid: design and implementation of the core architecture of the cancer biomedical informatics grid. *Bioinformatics*, 22(15):1910–1916, 2006.
2. Hartel *et al* . Modeling a description logic vocabulary for cancer research. *Journal of Biomedical Informatics*, 2005.
3. González-Beltrán *et al* . Domain Concept-Based Queries for Cancer Research Data Sources. In *CBMS*, 2009.
4. Foster. The anatomy of the grid: Enabling scalable virtual organizations. In *CCGRID*, page 6, 2001.
5. Pollock *et al* . *Adaptive Information: Improving Business Through Semantic Interoperability, Grid Computing and Enterprise Integration*. Wiley-Interscience, 2004.
6. Drummond *et al*. Putting OWL in order: Patterns for sequences in OWL. In *OWLEd 2006*, 2006.
7. Cuenca-Grau *et al* . OWL 2: The next step for OWL. *J. of Web Semantics*, 6(4):309–322, November 2008.
8. Sattler *et al* . Which kind of module should I extract? In *DL*, 2009.
9. McCusker *et al* . Semantic web data warehousing for cagrid. *BMC Bioinformatics*, 10 Suppl 10, 2009.
10. Kalyanpur *et al* . Finding all justifications of owl dl entailments. In *ISWC/ASWC*, pages 267–280, 2007.
11. Fegaras *et al* . Optimizing object queries using an effective calculus. *ACM Trans. Database Syst.*, 25(4):457–516, 2000.
12. Peim *et al* . Query processing with description logic ontologies over object-wrapped databases. In *SSDBM*, pages 27–36, 2002.
13. Garcia *et al*. A Survey on Ontology Metrics. *CCIS*, 111:22–27, 2010.
14. Berardi *et al* . Reasoning on uml class diagrams. *Artificial Intelligence*, 168(1-2):70–118, 2005.
15. Gašević *et al* . MDA-based Automatic OWL Ontology Development. *JSTTT*, 9(2):103–117, 2007.
16. Chen *et al* . Dartgrid: a semantic infrastructure for building database grid apps. *Concurr. Comput. : Pract. Exper.*, 18(14):1811–1828, 2006.
17. Tsiknakis *et al*. A semantic grid infrastructure enabling integrated access and analysis of multilevel biomedical data in support of postgenomic clinical trials on cancer. *IEEE Trans on IT in Biomedicine*, 12(2):205–217, 2008.
18. Shironoshita *et al* . semCDI: Semantic Query Formulation for caBIG. *JAMIA*, 15(4):559–568, 2008.