Toward cross-granular querying over modularized ontologies

C. Maria Keet

Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
keet@inf.unibz.it

Abstract. To address the problems of both structured coordination of linked and modularised ontologies and to query a large dynamic ontology system, we propose a basic granularity framework and a set of functions to query such a granulated system. The granularity framework enforces a constrained and structured modularization. This facilitates automation of both dividing a large body of represented information as well as relinking the pieces. The functions enable basic cross-granular querying in a transparent and scalable way, as they rely on the unambiguous management layer provided by the granularity framework, and are reusable for ontologies represented and stored in different formats.

1 Introduction

Modularization of ontologies and modularization of the subject domain to have different ontologies require, to some extent, different methodologies and technologies to address successfully. Tried and tested approaches are, for instance, manual modularization of conceptual data models as is customary in software development with UML Packages [1], and the inverse where one has a large ontology that has to be split-up somehow in smaller chunks [2, 3]. The former tends to be used to make the 'cognitive overload' manageable, whereas the latter has more to do with a system's performance and scalability. Moreover, modularization of the subject domain can avoid the problem of having to modularize a large ontology, but it passes on problems to, among others, 1) how to modularize in an optimal way, 2) which way is optimal, 3) how to keep the modules coordinated to avoid subject domain overlap and corresponding range ontology matching issues, 4) how to repeatedly (re-)connect those ontology modules statically or dynamically on-demand when a user poses a query across the modules, and 5) if one could keep such querying locally within the module by good design of the modularization. The most active domain of ontology development, bio-ontologies, faces both issues: having to deal with very large ontologies that can benefit from modularization, as well as having to manage multiple smaller ontologies that need to be connected. One direction bio-ontologists have taken is coordination of the 'modules' along levels of *granularity*, i.e., a specific *constrained modularization* that has flavours of subject domain-motivated modularization, yet also the desire to computationally manage this. A formal approach to such endeavours is

lacking, however. Research into computational implementations of granularity is not new, and is applied mainly in GIS, data warehousing, and time granularity, but its use to coordinate ontologies and application in knowledge representation in general, is. Here it will be demonstrated that with a formally defined granularity framework as additional knowledge management layer, the modularization and connectivity is not just any mapping, but can be semantically enriched with *why* and *how* the linking between the modules has been performed. This also greatly simplifies posing granular queries over such a modularized system. The connected ontologies remain separable, yet can be linked transparently into one coherent system within such a domain granularity framework.

In the remainder of the paper, we first introduce a motivating example in section 2, after which we introduce a basic granularity framework in section 3. In section 4 we introduce the set of functions to perform granular queries over the to be loosely linked and coordinated ontologies at different levels of granularity. We close with conclusions and future work in section 5.

2 Motivating example

Early attempts to group and coordinate bio-ontologies and related medical information systems focussed primarily on identifying which perspectives one can take on the chosen subject domain, which levels of granularity one can identify, and how many of them would be practically useful [4–6]. The largest and most ambitious effort to date is OBO Foundry's approach to coordinated evolution of ontologies [7] and LAV integration of biological databases. OBO Foundry's overview of *both* linking *and* modularizing bio-ontologies at different levels of granularity is depicted in Fig.1-A, which lists 10 of the 16 Foundry ontologies out of the 64 that are indexed [8] (as of 3-3-2008). For instance, at the "Organ and Organism"-level, the NCBI taxonomy may be linked to both the Foundational Model of Anatomy [9], which covers human anatomy only but into great detail for supra-cellular levels, and CARO [10] that also considers anatomy of other types of organisms but contains only generic universals for anatomy. How all this coordination among the ontologies and between sections of ontologies is to be implemented is not yet clear—and far from trivial. The ontologies are developed in a distributed fashion and owned by different organizations, represented in different representation languages, of widely varying size, and are intended to be *complimentary* in coverage. The latter has as major advantage that the linking of ontologies is not expected to face serious difficulties like with traditional ontology matching [11, 12]. Instead, connect points between the ontologies and sections thereof have to be clearly defined and re-computed each time after one of the participating ontologies is updated (which is a frequent event). A first step is enabling computational management of those different perspectives and levels, which includes both a formal representation of granularity as well as (cross-)granular querying for basic information retrieval from such a large domain that no person comprehends in full. The second step is cross-granular querying, for instance:

A

Granularity	Continuant				Occurrent
	Independent		Dependent		
Organ and organism	Organism (NCBI taxonomy or similar)	Anatomical entity (FMA, CARO)	Organ function (Physiology ontology, to be determined)	Phenotypic quality (PATO)	Organism-level process (GO)
Cell and cellular component	Cell (CL, FMA)	Cellular component (FMA,GO)	Cellular function (GO)		Cellular process (GO)
Molecule	Molecule (ChEBI, SO, RnaO, PRO)		Molecular function (GO)		Molecular process (GO)

B

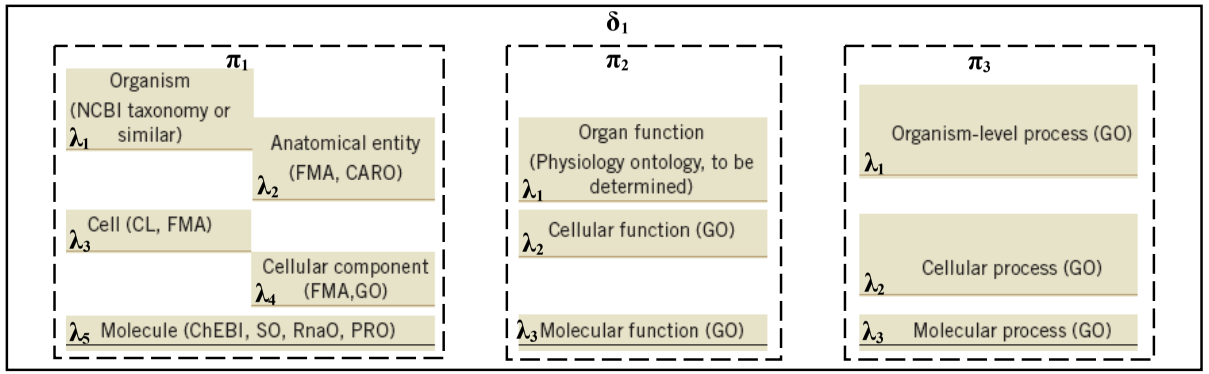

Fig. 1. A: OBO Foundry table of ontologies and current coverage of ontologies at different levels of granularity, divided by level of granularity (rows), perspective (columns), and levels' contents that are the ontologies listed between the braces after each level's name, such as the GO and Protein Ontology [7]; B: Clearer, finer-grained distinctions of the levels (shaded boxes with λ_i) that are contained in perspectives (indicated with dashed rectangles and π_i) through the $R_E(\lambda_i, \pi_i)$ relation, that, in turn, are contained in the domain granularity framework δ_1^f (solid rectangle) with $R_E(\pi_i, \delta_i^f)$. Relations between levels, such as $R_L(\lambda_5, \lambda_4)$ in π_1, and between perspectives (e.g., $R_P(\pi_1, \pi_2)$) are not drawn. Abbreviations: CARO = Common Anatomy Reference Ontology, ChEBI = Chemical Entities of Biological Interest, CL = Cell Type, FMA = Foundational Model of Anatomy, GO = Gene Ontology, NCBI = US National Centre for Biotechnology Information, PRO = Protein Ontology, RnaO = RNA ontology, SO = Sequence ontology.

Q1: "What are the cell components of blood?", which can be decomposed into smaller tasks where Blood resides in another level as the cells it has as parts
Q2: "which organs have macrophages?", i.e. for each macrophage (and its subclasses in the Cell-level), which organ (or its subclasses in the Organ-level) are they part of?
Q3: "Which hormones are located in the kidney, and where in the kidney?" This query uses three different perspectives, being a structural one for Kidney (an organ) with its parts, one for location (a spatial perspective), and a functional one for Hormone (a molecule with a specific function).

That is, we need ways to select and retrieve entities, levels, and perspectives, and combine such queries into more complext ones.

3 Basic granularity framework for relating the ontologies

To enhance coordination of the modules that contain the ontologies or sections of an ontology at different levels of granularity, we introduce a *simplified*, yet effective, granularity framework. A comprehensive theory of granularity with definitions, constraints, and proofs—i.e., with model-theoretic semantics—is presented in [13], but abridged here due to space limitations. The main components are given in the following definition. Note that, given that it is to be applied to (type-level) ontologies, we would be in second order, but they could be used in a first order language or description logic language with nominalization, added outside the ontology itself in the application layer, or added to an ontology that is stored in a database.

Definition 1 (Granularity framework $\mathcal{G}$) *A granularity framework is a tuple* $\Sigma = \{\Delta, \Pi, \Lambda, \Upsilon, \Theta, \Gamma, R_E, R_L, R_C, R_G, uses_\gamma\}$ *where*

- Δ *is the domain, that can be divided up into a particular subject domain* δ^s *and the encompassing granularity frame* δ^f *that contains the other elements of the granularity framework;*
- Π *denotes granular perspective (granulation hierarchy), where its instances are denoted with* $\pi_1, \ldots, \pi_n$*;*
- Λ *denotes granular level, where its instances are denoted with* $\lambda_1, \ldots, \lambda_n$*;*
- Υ *is the granulation criterion (a combination of at least two properties) by which one granulates a particular perspective, where its instances are denoted with* $\upsilon_1, \ldots, \upsilon_m$*;*
- Θ *is a type of granularity from the taxonomy of types of granularity [14];*
- Γ *is a granulation relation between entities residing in adjacent levels;*
- R_E *is a binary parthood relation constrained to relating two framework components, being either a level and a perspective or a perspective and a domain;*
- R_L *is a binary parthood relation constrained to relating two adjacent fine and coarser-grained levels that reside in the same perspective;*
- R_C *is a binary relation associating a granulation criterion to a perspective;*
- R_G *is a binary relation relating a perspective or level to the type of granularity it adheres to;*
- $uses_\gamma$ *is a binary relation between* Θ *and* Γ*.*

Salient constraints are that each perspective must have at least two levels, a level must be contained in exactly one perspective ($\forall x(\Lambda(x) \rightarrow \exists! y R_E(x,y))$), that the multiplicity (cardinality) for R_L is 1:1, and a perspective can be identified by the combination of the criterion and type of granularity it adheres to ($\forall x(\Pi(x) \rightarrow \exists! y, \phi(R_C(x,y) \wedge R_G(x,\phi)))$ where ϕ is a shorthand for any of the eight types of granularity). The components of $\mathcal{G}$ enable one to represent explicitly the distinction between, e.g., an is-a taxonomy of structural body parts versus a partonomy of body parts ('is-a vs. part-of' is dealt with by Θ and Γ and the 'human structural anatomy' by Υ and Π), and to manage various distinct perspectives within one domain. There are three perspective depicted in Fig.1-B, each with at least three levels. As levels for the partonomic granular perspective for humans, we have in Fig.1-B, e.g., $\lambda_2 = Organ$ in π_1, where its *contents* are provided by the FMA by means of selecting the Organ entity type as root and recursively querying for the taxonomic subtypes of organ (about 3000 entity types). The molecule levels in the three perspectives, on the other hand, are not well covered by the FMA, therefore this gap is filled by other ontologies, such as PRO and ChEBI in λ_5 in Fig.1-B. That is, the contents at each level are intended to be (or assumed to be) complimentary to one another, where the $\mathcal{G}$ provides the machinery for a structured coordination and linking; hence, *one knows what is linked where and how*, which then is, at least in theory, easy to generate again when an ontology is developed independently and at certain time intervals updated and re-connected. Likewise, adding a new level filled with another ontology can be done transparently and non-disruptive for the other perspectives and levels. Thus, we have four principal components for a granulated information system: the types of granularity [14] that links to the granularity framework as theory (Definition 1) through Θ, an instantiation (model) of this theory for a specific subject domain, such as the λ_i, π_i etc. shown in Fig.1-B, and the data sources that are granulated (the GO, FMA, etc.).

4 Granular queries

The next step is cross-ontology reasoning over such ontologies linked by granularity, which already has been noted as a requirement [7, 15]. A major advantage of having a granularity framework for coordinating the different (sections of) ontologies, is that one can reuse the underlying idea of querying conceptual models [16], but where in this case it is not the conceptual model but the granularity framework $\mathcal{G}$ (illustrated in Fig.1-B) that is used to structure and simplify the granular queries. They share the notion that the goal of the query can remain the same, but their implementation differ, be it different SQL versions [16] or a Semantic Web ontology language. Given that ontologies can be represented and saved in different languages and software, we define the types of granular queries in a *purpose-oriented* way to ensure portability. The first group deals with querying for perspectives and levels, the second with retrieving level's contents, and the third group with conditional cross-granular queries and other auxiliary

queries to retrieve additional information. The functions will be discussed in the remainder of this section.

Table 1. List of function arguments and how they relate to each other.

Symbol	Selected section
$\mathcal{P}$	Set of perspectives in a domain granularity framework i.e., $\pi_1, \ldots, \pi_n \in \mathcal{P}$
$\mathcal{DL}$	Set of levels in a domain granularity framework, i.e., $\lambda_1, \ldots, \lambda_m \in \mathcal{DL}$
$\mathcal{L}$	Set of levels $\lambda_i, \ldots, \lambda_k$ $(i, k \leq m,\ i \neq k)$ of a particular π_i
l_i	Selected level λ_i, output of the function *selectL*
ls_i	Set of selected levels, output of the function *selectLs*
lss_i	Set of selected levels, output of the function *selectDL*
$\mathcal{E}$	Collection of universals residing in a particular level λ_i
$\mathcal{I}$	Intersection of the contents of two levels λ_i and λ_j (with $i \neq j$)

Selection of one level. Selecting a level within a perspective is a four-step process: retrieve the desired perspective, select a perspective, retrieve the levels in the selected perspective, and then select the desired level; subscripts denote what is selected.

F1. ***Goal:*** *retrieve all granular perspectives $\pi_1, \ldots, \pi_n$ contained in the domain granularity framework d^f. Input is a d^f and output is an unordered set of perspectives of that domain, P.* ***Specification:*** *getP* $: \delta^f \mapsto \mathcal{P}$ *(F.1)*

F2. ***Goal:*** *select a granular perspective π_i from the perspectives retrieved with getP. Input is the set of perspectives of the domain, $\mathcal{P}$, output is a set with one perspective $\pi_i \in \mathcal{P}$.* ***Specification:*** *select*$P_{\Pi=\pi_i} : \mathcal{P} \mapsto \mathcal{P}$ *(F.2)*

F3. ***Goal:*** *retrieve all granular levels $\lambda_1, \ldots, \lambda_n$ contained in the selected perspective π_i. Input is a $\pi_i \in \mathcal{P}$ and output is an ordered set of levels of that perspective, $\mathcal{L}$.* ***Specification:*** *getL* $: \mathcal{P} \mapsto \mathcal{L}$ *(F.3)*

F4. ***Goal:*** *select a particular granular level λ_i from the levels retrieved with the getL function. Input is the set of levels of the perspective, $\mathcal{L}$, output is a set, l_i, with a single element from $\mathcal{L}$.* ***Specification:*** *select*$L_{\Lambda=\lambda_i} : \mathcal{L} \mapsto \mathcal{L}$ *(F.4)*

Although one could choose to add a function to select a level from $\mathcal{DL}$ (all specified levels) only, this is prone to user-mistakes, the above four functions can more easily be reused for other selection and retrieval functions, and they can be abstracted into one compound user-interface operation anyway.

To select the Cell-level as declared in Fig.1-B for query Q2, one uses *getP* to retrieve $\{\pi_1, \pi_2, \pi_3\}$, *select*$P_{\Pi=\pi_1}$ to obtain the human structural anatomy perspective, and with *getL* the levels of that perspective ($\{\lambda_1, \ldots, \lambda_5\}$), and, last, *select*$L_{\Lambda=\lambda_3}$ for the Cell-level. The same procedure can be used for selecting the Organ-level, but with the last step being *select*$L_{\Lambda=\lambda_2}$.

Selection of multiple levels. Two options to select multiple levels are distinguished: 1) selecting several levels to subsequently retrieve and combine its contents for further processing, and 2) conditional selection that considers the contents as well. Option 1 can be subdivided into two similar operations: selecting more than one level from one perspective and selecting levels from different perspectives. Both can be accomplished with a sequence of sub-functions. In addition to those introduced in the previous section, we have:

F5. ***Goal:*** *select* ≥ 1 *granular levels contained in one granular perspective. Input is set of levels* $\mathcal{L}$ *retrieved with getL, output is a set of selected levels* ls_i *from* $\mathcal{L}$ *such that* $ls_i \subseteq \mathcal{L}$ *holds.* ***Specification:*** $selectLs_{\bigwedge \lambda_i} : \mathcal{L} \mapsto \mathcal{L}$ *(F.5)*

F6. ***Goal:*** *perform a selection of* ≥ 1 *levels from* ≥ 1 *perspectives. Input is the set of perspectives,* $\mathcal{P}$*, and for each perspective the set of levels,* $\mathcal{L}$*, from which one selects, and output is a set of levels contained in* d^f*,* $\mathcal{DL}$*, denoted with* lss_i*, where* $lss_i \subseteq \mathcal{DL}$*.* ***Specification:*** $selectDL_{\bigwedge \pi_i \bigwedge \lambda_i} : \mathcal{P} \times \mathcal{L} \mapsto \mathcal{DL}$ *(F.6)*

Thus, *selectL* is extended for multi-level selection within one perspective, using the binary operator $\wedge$ to select multiple levels, which has been written in shorthand notation, $\bigwedge$, such that $\bigwedge \lambda_i = \lambda_1 \wedge ... \wedge \lambda_k$ where π_i contains j levels, $k \leq j$, and $R_E(\lambda_i, \pi_i)$. Note that ls_i is not a proper subset of $\mathcal{L}$ because it is possible that a user wants to select all levels in the chosen perspective. The 1-level selection is a special case of the multi-level one, but labelled differently to avoid overloading terms.

For selection of more than one level from more than one perspective, *selectLs* cannot be extended to selecting levels from multiple perspectives, because one has to nest selection of at least one other perspective and have some way to distinguish levels belonging to different perspectives. A mapping of (F.6) to a formal notation and implementation algorithm can be achieved with a loop in two near-equivalent ways: either to select all desired perspectives and subsequently one or more levels for each selected perspective, or repeat the two-step process of selecting a perspective & retrieve levels and then selecting levels.

Retrieving the contents of a level. Retrieving the contents of a particular granular level is conceptually straightforward with a *getC* function, but this hides many details, in particular the need to use the structure of the level's contents given the different types of granularity. This is elaborated on afterward.

F7. ***Goal:*** *retrieve the contents, i.e., entity types and their relations, of a selected granular level* λ_i*. Input is the selected level, where* $\lambda_i \in \mathcal{L}$ *and output is a set of predicates,* $E \in \mathcal{E}$*, that takes into account the structure of the contents in the level.* ***Specification:*** $getC : \mathcal{L} \mapsto \mathcal{E}$ *(F.7)*

F8. ***Goal:*** *intersect the retrieved contents of selected granular levels* λ_i, λ_j *(with* $R_E(\lambda_i, \pi_i)$*,* $R_E(\lambda_j, \pi_j)$*, and* $i \neq j$*). Input are the contents of the selected levels*

E_i and E_j (obtained with getC for each level), and output is the intersection, set $I \in \mathcal{I}$, where $\mathcal{I} \subseteq \mathcal{E}$. ***Specification:*** *intersect* : $\mathcal{L} \times \mathcal{L} \mapsto \mathcal{I}$ *(F.8)*

getC takes a particular granular level as argument and returns the contents of that granular level, *irrespective of how the contents themselves may be structured*. Without having represented explicitly which type of granularity is used for the perspective and levels, one cannot know this other than manually hardcoding this information, which is laborious and error-prone. However, if we have, for instance, two granular perspectives that both devise the levels where entity types in finer-grained levels are always part of those residing in coarser levels and for both hierarchies the contents is a taxonomy, such as types of organs and types of molecular processes, then the mechanism of granulation is the same (just applied to different subject domain information). One can exploit this sameness in approach of granulation by identifying each principal granulation mechanism and relate the mechanism to the levels and perspectives, so that one has to define only *once* how the contents of a level should be retrieved. For the types of organs and types of molecular processes, this amounts to a straight-forward recursive query. To achieve this efficiency, we can reuse the taxonomy of types of granularity [14], i.e. a set of granulation mechanisms, and relate that to the perspectives and levels. This is achieved by reusing R_G and transforming it into a function, tgL, which is an abbreviation of *t*ype of *g*ranularity that the *l*evel adheres to, thus $tgL : \mathcal{L} \mapsto \Theta$ iff $R_G(x, \phi)$, where R_G has been typed already such that $\Lambda(x)$ and $\phi \rightarrow \Theta$ (ϕ is syntactic sugar for the eight leaf types in the taxonomy of types of granularity). Thus, to retrieve a level's contents with *getC*, the nested function tgL has to be used to query for the type of granularity that a level adheres to (details can be found in section 4.2 of [13]). In an implementation, this may well be an intermediate query or database view in the ontology-stored-in-a-database, done with a method in a C++ program, and so forth. For instance, to retrieve Macrophage (in Q2), we first consult the type of granularity that λ_3 adheres to, i.e., $tgL(\lambda_3)$, which returns nrG by which we also obtain the type of granulation relation used, which is proper parthood (see [13] for details). Subsequently, the contents of level λ_3 is retrieved with $getC(\lambda_3)$ and, by knowing the type of granularity—hence, also the mechanism required for retrieval—it is ensured that a taxonomy is returned as query answer (in casu, a taxonomy of cell types); to select the entity type of interest, Macrophage, the *selectE* function will be introduced below.

Once the content is retrieved, it can be used for further processing, such as intersecting contents of two levels. To this end, the *intersect* function (F.8) is introduced, which first retrieves the contents of each level with *getC* and subsequently intersects them. For instance, to answer a query like "retrieve those molecules that are also hormones" we have, among others, the molecule Insulin in the molecule-level λ_5 of π_1 in Fig.1-B but also in λ_3 of the function-perspective π_2 where it is categorised as a hormone. Obviously, this can be scaled up to intersection of more than two levels.

Type selection to retrieve its levels. We introduce five functions to achieve unambiguous selection of types (universal/class/concept) from an ontology and to query for the level(s) it resides in to simplify scalability and reusability.

F9. ***Goal:*** *select a type from the ontology (subject domain). Input is a type, $C \in d^s$, and the output of the selection ensures that the selected type resides in some level λ_i already, hence $C \in E$.* ***Specification:*** *select$E : \delta^s \mapsto \mathcal{E}$* *(F.9)*

F10. ***Goal:*** *given a selected type C, retrieve the level λ_i that C resides in. Input is a type $C \in d^s$ and output of the function is a level that is a subset of $\mathcal{L}$.* ***Specification:*** *grain : $\delta^s \mapsto \mathcal{L}$* *(F.10)*

F11. ***Goal:*** *given a selected type C, retrieve all the levels $\lambda_1, \ldots, \lambda_j$ (in different perspectives $\pi_1, \ldots, \pi_k$, $k = j$) that C resides in. Input is a type $C \in d^s$, and output of the function is a set of levels that is a subset of $\mathcal{DL}$.* ***Specification:*** *grains : $\delta^s \mapsto \mathcal{DL}$* *(F.11)*

F12. ***Goal:*** *given multiple selected types $C_1, \ldots, C_n$, retrieve their levels $\lambda_1, \ldots, \lambda_j$ within one perspective π_i. Input are types $C_1, \ldots, C_n \in d^s$, uses grain as nested function, and output of the function is a set of levels within one perspective (a subset of $\mathcal{L}$).* ***Specification:*** *grain$M : \delta^s \mapsto \mathcal{L}$* *(F.12)*

F13. ***Goal:*** *given multiple selected types $C_1, \ldots, C_n$, retrieve their levels $\lambda_1, \ldots, \lambda_j$ among multiple perspectives $\pi_1, \ldots, \pi_k$. Input are types $C_1, \ldots, C_n \in d^s$, uses grains as nested function, and output of the function is a set of levels as subset of $\mathcal{DL}$.* ***Specification:*** *grains$M : \delta^s \mapsto \mathcal{DL}$* *(F.13)*

The basic function to retrieve the level an entity resides in, is *grain*. Its neat simplicity, however, does not suffice [5]. The main limitations are that it is perspective-unaware and is based on the assumption that any entity type C_i can be categorised only in one level in the whole granularity framework. Although it is possible that a particular granularity framework contains only π_1 or constrain it to that [17, 4], it is more realistic that multiple perspectives have been declared and that C_i is located in more than one granular level across perspectives [5]. To remedy this, two scalable and reusable options are at our disposal. First, one can restrict usage to one limited case: if one knows which perspective to search, one can construct a rule alike "if π_i then do $grain(x) = \lambda_i$" or precede it with the selection operator *select$P$*. The same approach can be used to decompose the retrieval of multiple levels into sequential steps of the *grain* function, but this requires additional process management. Second, to define a new function that retrieves *all* levels the selected type resides in, i.e. *grains* (F.11).

In a more complex subject domain than the OBO Foundry setting depicted in Fig.1, such as human infectious diseases that could rely on the granulations of the OBO Foundry, we then can retrieve all levels of, say, Cholera_toxin, with the *grains* function and retrieve $\{\pi_2\lambda_9, \pi_6\lambda_2\}$; that is, using the granulation from

[5], the Cholera_toxin resides in the Molecule-level (from structural component perspective π_2) as well as the Inhibitor-level (from π_6, a function the molecule has in the Second Messenger System). The four functions (F.10-F.13) address all permitted options to find the level(s) of entities (/types). Other combinations, such as one type in one perspective residing in multiple levels, violate the constraints of granularity framework $\mathcal{G}$. For instance, if a user had allocated Nisin (a type of bacteriocin) both in the Peptide (λ_k) and in the Quaternary protein structures (λ_i, where $i < k$) levels within the same perspective, then something is wrong about the knowledge of what Nisin is, the user is confused, the domain granularity framework has been ill-designed, or all of them, because Nisin cannot be both a peptide and complex protein. The constraint to have an entity in no more than one level within the same perspective should have prevented this double allocation, or have returned an error upon checking the granulated system for consistency.

Cross-granular queries and other functions. The functions (F.1-F.13) introduced in the preceding subsections enable formulation of more complex queries, such as Q1-Q3 in section 2. It is possible to define many more functions for a granularity framework, also because, in principle, the same approach could be taken for knowledge bases and, among others, ontology-enhanced Data WareHouses (DWH) or modularized ontology-driven information systems. DWH implementations in particular focus on querying the information system with advanced queries. Functions for such queries can be easily added. For instance, with $\mathcal{P}$ still the set of perspectives, and adding $\mathcal{C}$ as the set of criteria, then $\text{crit} : \mathcal{P} \mapsto \mathcal{C}$ enables us to retrieve the criterion of a granular perspective, $\text{tgP} : \mathcal{P} \mapsto \Theta$ to retrieve the type of granularity of a perspective, and plain functions to query the entity types and relations of the ontology proper are also still possible [18, 13] either independently or together with the granularity framework by using a granulation relation from Γ. Moreover, *one can define functions to retrieve each component* of the granularity framework $\mathcal{G}$ and the more comprehensive the representation of granularity-components, the more versatile and well-founded the queries one can pose over a $\mathcal{G}$. There are limitations to it, however, in particular if the ontology is represented in a common DL-based Semantic Web ontology language. Most notably, then we can neither represent and query over the weak entity type that Π is nor combine querying for Θ and Υ in one go due to known undecidability results with the role composition operator [19, 20], likewise if one would want to introduce a newly defined path Π_has_Γ such that $\Pi\_has\_\Gamma \triangleq R_G \circ uses_\gamma$ to retrieve the granulation relation of the entity types residing in the levels in a particular perspective (e.g., π_1 in Fig.1-B uses proper parthood as γ). Implementing the coordinated modularization with granularity and such advanced querying for ontologies stored in database is, of course, an option. On the other hand, given the current state of granulation in bio-ontologies, these issues are not of great concern, yet. It will be the modeling exercise to meaningfully and in a structured way *represent* a particular granularity framework that will be of most use to organise the relatively large

amount of bio-ontologies at this stage. In addition, this eases transparent coordination of the distributed development of the yet to be developed ontologies to fill the 'gaps' in the granulation hierarchies as well as to maintain the existing ontologies—often also developed in a distributed fashion—on a long-term basis.

Looking at prospects for immediate use, the functions obviously could be hard-coded and pre-computed as is customary in bioinformatics, be it as successive steps or also with additional compound queries. However, with the granularity framework, one has a management structure in place so that (i) queries can be executed that suits the user *on-demand* as opposed to being limited to the imagination of the software developer, and (ii) F.1-F.13 are defined in an implementation-independent and a reusable way so that practicalities of a software system can be hidden from the domain experts so as to avoid burdening them to learn yet another query language, to re-write the whole query for each permutation (e.g., macrophages in tissues), and to keep in mind which levels there are whereas that can be added to the informationsystem now. At present, Q1-Q3 can be performed by standard relational database management systems (RDBMS), such as the FMA in PostgreSQL [21] with queries in StruQL. This requires more effort for OWL ontologies due to traversal over a path of an arbitrary but finite amount of DL-roles for Q2 [22].

5 Conclusions

To address the problems of both structured coordination of linked and modularised ontologies and to query a large dynamic ontology system like proposed by the OBO Foundry, we proposed a basic granularity framework and a set of functions to query such a very large granulated system. The granularity framework enforces a constrained modularization so that dividing a large body of represented information as well as re-linking the pieces is amenable to automation. The proposed functions enhance cross-granular querying in transparent and scalable way, because they rely on the unambiguous management layer provided by the granularity framework, and are reusable for ontologies represented and stored in different formats. We are currently working on transforming the theory of granularity [13] to a practically usable OWL version to ease experimentation in the real-life setting with the many available bio-ontologies.

References

1. OMG: Superstructure specification. Standard 2.1.2, Object Management Group (2007) http://www.omg.org/spec/UML/2.1.2/.
2. Cuenca Grau, B., Parsia, B., Sirin, E., Kalyanpur, A.: Modularity and web ontologies. In: Proc. of KR'06. (2006) Lake District, UK.
3. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Just the right amount: Extracting modules from ontologies. In: Proc. of WWW-2007. (2007) Canada.
4. Kumar, A., Smith, B., Novotny, D.D.: Biomedical informatics and granularity. Comparative and Functional Genomics **5**(6-7) (2005) 501–508

5. Keet, C.M., Kumar, A.: Applying partitions to infectious diseases. In Engelbrecht, R., Geissbuhler, A., Lovis, C., Mihalas, G., eds.: Connecting Medical Informatics and bio-informatics (MIE2005), Amsterdam: IOS Press (2005) 1236–1241 Geneva, Switzerland, 28-31 August, 2005.
6. Tange, H.J., Schouten, H.C., Kester, A.D.M., Hasman, A.: The granularity of medical narratives and its effect on the speed and completeness of information retrieval. J. Am. Med. Inf. Assoc. **5**(6) (1998) 571–582
7. Smith, B., Ashburner, M., Rosse, C., Bard, J., Bug, W., Ceusters, W., Goldberg, L., Eilbeck, K., Ireland, A., Mungall, C., OBI Consortium, T., Leontis, N., Rocca-Serra, A., Ruttenberg, A., Sansone, S.A., Shah, M., Whetzel, P., Lewis, S.: The OBO Foundry: Coordinated evolution of ontologies to support biomedical data integration. Nature Biotechnology **25**(11) (2007) 1251–1255
8. : OBO Foundry (2006) http://obofoundry.org.
9. Rosse, C., Mejino Jr, J.V.: A reference ontology for biomedical informatics: the foundational model of anatomy. J. of Biomedical Informatics **36**(6) (2003) 478–500
10. : Common anatomy reference ontology (2006) http://www.bioontology.org/wiki/index.php/CARO:Main_Page.
11. Bouquet, P., Euzenat, J., Franconi, E., Serafini, L., Stamou, G., Tessaris, S.: Specification of a common framework for characterizing alignment. Technical report, KnowledgeWeb Deliverable D2.2.1, v1.2, 3-8-2004 (2004)
12. Klein, M.: Combining and relating ontologies: Problems and solutions. In: IJCAI Workshop on Ontologies. (2001) Seattle, USA.
13. Keet, C.M.: A Formal Theory of Granularity. Phd thesis, KRDB Research Centre, Faculty of Computer Science, Free University of Bozen-Bolzano, Italy (2008)
14. Keet, C.M.: A taxonomy of types of granularity. In: Proc. of GrC2006. IEEE Computer Society (2006) 106–111 Atlanta, USA, May 10-12 2006.
15. Keet, C.M., Roos, M., Marshall, M.S.: A survey of requirements for automated reasoning services for bio-ontologies in OWL. In: Proc. of OWLED2007. Volume 258 of CEUR-WS. (2007) 6-7 June 2007, Innsbruck, Austria.
16. Bloesch, A.C., Halpin, T.A.: Conceptual Queries using ConQuer-II. In: Proc. of ER'97. Volume 1331 of LNCS., Springer (1997) 113–126
17. Bittner, T., Smith, B.: A Theory of Granular Partitions. In: Foundations of Geographic Information Science. London: Taylor & Francis Books (2003) 117–151
18. Keet, C.M.: Enhancing comprehension of ontologies and conceptual models through abstractions. In Basili, R., Pazienza, M., eds.: Proc. of AI*IA 2007. Volume 4733 of LNAI., Springer Verlag (2007) 814–822 Rome, September 10-13, 2007.
19. Schmidt-Schauss, M.: Subsumption in KL-ONE is undecidable. In: Proc. of KR'89. (1989) 421–431
20. Wessel, M.: Obstacles on the way to qualitative spatial reasoning with description logics: some undecidability results. In Goble, C.A., McGuinness, D.L., Möller, R., Patel-Schneider, P.F., eds.: Proc. of DL'01. Volume 49 of CEUR WS. (2001) Stanford, CA, USA, August 1-3, 2001.
21. Mork, P., Brinkley, J.F., Rosse, C.: OQAFMA querying agent for the foundational model of anatomy: a prototype for providing flexible and efficient access to large semantic networks. Journal of Biomedical Informatics **36**(6) (2003) 501–517
22. Keet, C.M.: Granular information retrieval from the Gene Ontology and from the Foundational Model of Anatomy with OQAFMA. KRDB Research Centre Technical Report KRDB06-1, Free University of Bozen-Bolzano, Italy (2006)