Aondê: An Ontology Web Service for Interoperability across Biodiversity Applications

Jaudete Daltio, Claudia Bauzer Medeiros

[1]Institute of Computing - State University of Campinas (UNICAMP)
CP 6176 13084-971 Campinas, SP - Brazil

jaudete@gmail.com, cmbm@ic.unicamp.br

Abstract. *Biodiversity research requires associating data about living beings and their habitats, constructing sophisticated models and correlating all kinds of information. Data handled are inherently heterogeneous, being provided by distinct (and distributed) research groups, which collect these data using different vocabularies, assumptions, methodologies and goals, and under varying spatio-temporal frames. Ontologies are being adopted as one of the means to alleviate these heterogeneity problems, thus helping cooperation among researchers. While ontology toolkits offer a wide range of operations, they are self-contained and cannot be accessed by external applications. Thus, the many proposals for adopting ontologies to enhance interoperability in application development are either based on the use of ontology servers or of ontology frameworks. The latter support many functions, but impose application recoding whenever ontologies change, whereas the first supports ontology evolution, but for a limited set of functions.*

This paper presents Aondê – a Web service geared towards the biodiversity domain that combines the advantages of frameworks and servers, supporting ontology sharing and management on the Web. By clearly separating storage concerns from semantic issues, the service provides independence between ontology evolution and the applications that need them. The service provides a wide range of basic operations to create, store, manage, analyze and integrate multiple ontologies. These operations can be repeatedly invoked by client applications to construct more complex manipulations. Aondê has been validated for real biodiversity case studies.

Keyword: Ontology Management, Web Services, Biodiversity Information System, Semantic Integration.

1. Introduction

Biodiversity research is a multidisciplinary field that requires cooperation of many kinds of scientists that collect, correlate and analyze data on living beings and their habitats, and construct models to describe species' interactions. Available data are collected all over the world by distinct teams and published in many formats, following a variety of standards. This scenario is characterized by its intrinsic heterogeneity – not only of data and models, but also of requirements and profiles of the experts who collect and analyze the data. Data volume and species diversity contribute to complicate the issue: while roughly 2 million species have been identified, estimates for the number of species in the world vary from 10 million to more than 100 million [27].

In order to advance their knowledge of the world, scientists require means to support cooperation among research groups. Therefore, sophisticated mechanisms for data storage, management, sharing and retrieval are required to manage the huge amount of data produced, and their integration, correlation, fusion and interpretation. One additional issue is temporal heterogeneity. Not only do ecosystems evolve; ecological models and species' taxonomic classifications also change, reflecting the evolution of scientific knowledge about the real world. Hence, interoperability and manipulation of heterogeneous data is one of the major challenges faced by these scientists.

Biodiversity Information Systems (BIS) [53] provide partial solutions to some of these problems, publishing data sets and providing functions that allow analysis of species and their interactions. Queries in BIS typically combine textual data on species to geographical data (characterizing the ecosystems where the species are observed). *Occurrence records* (also called *collection records*) are one of the most important data sources. They describe observations of living beings (when and where they are observed, by whom and how). In spite of providing scientists with sophisticated analysis functions, BIS require data and model standardization, and interoperability across BIS is still an open problem.

The use of ontologies has been pointed out as a means to solve some of the above problems. Ontologies are descriptions of an abstract model of terms, related among themselves [26]. They model portions of a domain: its entities, relations and constraints, aiming to define a common agreement on that domain.

Several domains in biology already provide consensual ontologies – e.g., in bioinformatics, the Gene Ontology [9], or the TAMBIS Ontology [10]. In biodiversity, however, there are too many kinds of expertise involved, and consensual structures do not yet exist. A few multinational projects have been started to construct such ontologies – e.g., GBIF [23] – but they are still in their infancy.

In spite of the extensive research conducted on the use of ontologies to help interoperability and support cooperation across research groups, there is still much to be done. While there exist sophisticated toolkits to create and manage ontologies, they do not allow external access by client applications. Thus, in order to ensure ontology sharing and management across groups, application developers have to resort either to development environments to construct applications that use consolidated ontologies, or to create programs that invoke operations from servers that publish specific ontologies. The first kind of solution does not support ontology evolution, whereas the second limits application flexibility by providing a restricted set of access functions, with no possibility of handling multiple ontologies simultaneously. Moreover, in most cases there is not a clear separation between storage concerns and semantic manipulation, hampering ontology reuse and application development. Additionally, most approaches do not provide metadata on ontologies. Thus, applications (and users) are unable to determine factors such as ontology quality, or reliability.

This paper presents a SOAP-compliant Web Service, named Aondê[1], which meets these requirements. As will be seen, it combines the facilities provided by environments

[1] Aondê means "owl" in Tupi, the main branch of native Brazilian languages, denoting both the domain supported by the service, and the standard ontology language used by its implementation.

with the flexibility offered by servers to support ontology evolution. Moreover, it clearly separates storage management issues from high level ontological operations. The parameters of its basic set of functions can be dynamically defined by client applications, to query, search, rank, compare and integrate ontologies. Finally, Aondê allows access to multiple ontologies at a time, as long as they are published using Web Services. Ontology management is enhanced by associated metadata structures, thus fostering collaborative management of ontologies.

Aondê was designed and implemented to meet the needs of WeBios [57], a BIS being developed within a joint initiative of biodiversity and computer science researchers at the University of Campinas – UNICAMP – Brazil. The goal of WeBios is to provide scientists with a system that supports sharing of distributed biodiversity data sources on the Web. Though geared towards biodiversity applications, Aondê has been specified in a generic way, so that it can be extended and adopted by other kinds of application domains with similar volume and interoperability requirements.

This paper contributes therefore to the solution of problems of semantic heterogeneity in the Web, by presenting an ontology Web Service characterized by: (1) use of distributed repositories to manage and store ontologies and their metadata, separating low-level storage from higher level semantic concerns; and (2) specification and implementation of service operations that support integrated manipulation of sets of ontologies. Applications can thus enhance their semantics, and interoperate by becoming clients of this service, thereby exchanging, reusing, integrating and adopting concepts from ontologies published on the Web.

The rest of this paper is organized as follows. Section 2 contains a brief description of WeBios. Related work is described in section 3. Section 4 presents the specification of Aondê and its architecture. Section 5 concerns implementation aspects. A real case study is presented in section 6. Finally, section 7 concludes the paper, commenting on lessons learnt and ongoing work.

2. Overview of WeBios' Architecture

This section briefly presents WeBios, to show the principles behind the design of Aondê, while at the same time illustrating a typical context where this service can be used. WeBios is a biodiversity information system developed within a joint initiative of computer science and biodiversity researchers. Its goal is to provide the latter with a system that supports exploratory queries over heterogeneous and distributed biodiversity data sources on the Web. It has a service-oriented architecture and employs Semantic Web technologies.

Figure 1 presents a high-level view of WeBios' architecture, organized according to four main layers: *Storage Layer Services*, *Supporting Web Services*, *Enhanced Web Services* and *Client Applications*. This paper is concerned with the two outlined boxes: the Aondê Ontology Service and the Semantic Repositories Service.

While some services are already implemented, others are still being designed. The architecture shows the basic data and software organization that underlies our work. First, distinct kinds of data sources are published via Web Services. Second, services grow in complexity via their composition. Finally, applications can invoke services whenever they

need specific kinds of data and functions on these data. Consequently, the architecture supports interoperability on the Web and flexibility in application development.

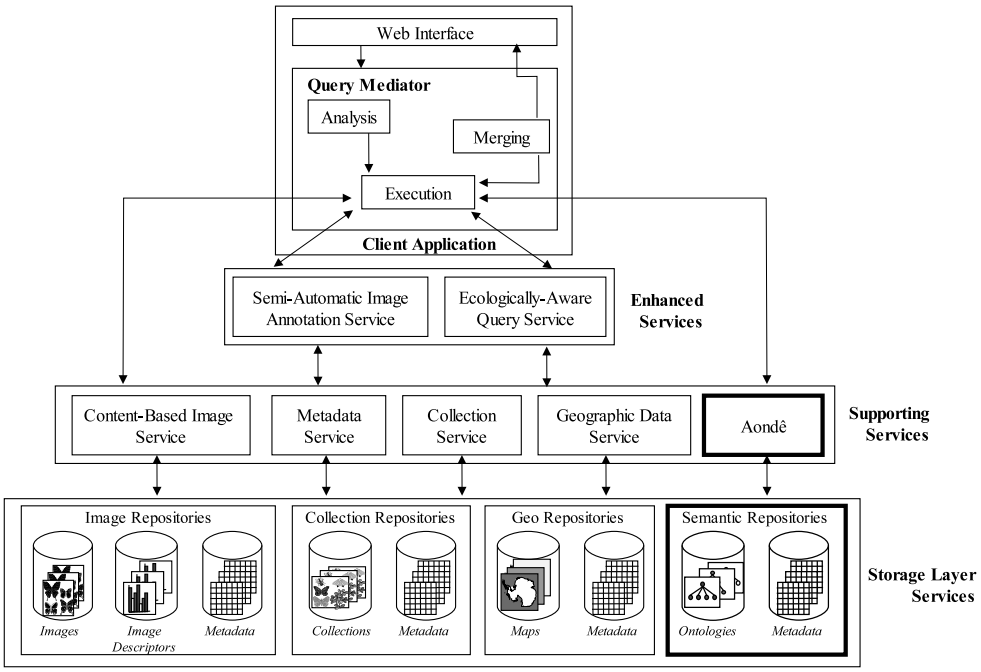

Figure 1. The WeBios Architecture - Main Components

Intuitively, a query at the *Web Interface* level is translated into a set of requests by the *Query Mediator*. These requests are dispatched to appropriate services. The *Enhanced Services* invoke the *Supporting Services* to answer requests that demand combined access to distinct kinds of data sources. The Enhanced and Supporting services retrieve the data from the *Storage Layer* services, process the requests and return the results to the Mediator, which will merge them and return the final answer to be shown at the Interface.

The *Storage Layer Services* are responsible for data storage and low-level data management in distributed repositories, which are fed by distinct biodiversity research projects. There are four kinds of primary data sources: images (Image Repositories), species' occurrence records (Collection Repositories), geographical and ecological data (Geo Repositories), and ontologies (Semantic Repositories).

Each of these sources has associated metadata. *Ontology data* are provided by the scientists to describe their work context, e.g., phylogenetic trees, taxonomic descriptions, ecological relations or habitat definition. Though figure 1 shows one single repository of each kind, several such repositories can exist, e.g., managed by distinct research projects, scientists or institutions.

The Supporting Services Layer comprises five Web Services, each of which dedicated to a specific data retrieval modality. The *Content-based Image Retrieval Service* processes requests based on image content. The *Metadata Service* retrieves information based on metadata parameters, allowing for different standards. The *Collection Service* provides access to occurrence records. The *Geographic Data Service* retrieves spatial information, used to create maps and species distribution in a given area. The *Aondê* Ontology Service is described in Sections 4 and 5.

So far, two Enhanced Services have been designed and partially implemented: Image Annotation and Ecologically-aware Queries. The *Image Annotation* service [22] helps users to annotate images with metadata and ontology terms. The *Ecologically-aware query* service [25, 31] allows users to pose queries on ecological relationships among species (e.g., predator-prey). Most of the *Storage Layer Services* have been implemented as prototypes, using a DBMS, but not as full-fledged Web services. Only *Aondê*, the *Semantic Repository* and the *Geographic Data Service* are implemented as Web services.

Our case study (Section 6) relies on taking advantage of the independence offered by service invocation. Complex data manipulations within an application are supported through sequences of invocations to distinct services. Ontology repositories are central to semantic disambiguation of queries and are detailed in Section 5.1. Aondê is detailed in Sections 5 and 4.Further details on WeBios are outside the scope of this paper.

3. Related Work

Our paper concerns ontology management, including studies on tools and operations. Though Aondê has been constructed for biodiversity information systems (BIS), it was specified and implemented in a general way. Thus, we will not discuss related work on BIS, just inserting appropriate explanations when needed. For the purpose of this paper, it suffices to know that all BIS rely on correlating ecological and geographic data, information on species and species collection records. Moreover, they can be specialized – i.e., concerning one specific kind of living being [29, 51] – or general purpose, covering a wide range of species (e.g., [50]). WeBios' design supports the latter.

3.1. Ontologies and Tools to Manipulate them

One of the most widely used definitions of ontology is [26]: *"an ontology is an explicit specification of a conceptualization"*. From a computer science perspective, an ontology can be viewed as a data model that represents a set of concepts within a domain and the relationships between those concepts. Knowledge in an ontology is formalized using four kinds of components:

- **Classes:** sets, or kinds of objects (concepts or categories of concepts in the domain), usually organized in taxonomies;
- **Instances:** the objects in the domain, represented as instances of a class;
- **Properties:** used to describe instances/classes. Properties may express object attributes or relationships;
- **Constraints:** abstractions that use properties to describe a class.

Many languages may be used to represent an ontology, such as RDF (Resource Description Framework) [36] and OWL (Web Ontology Language) [8]. Ontologies are usually displayed using graphs – nodes are classes, terminal nodes are instances, and edges represent properties, class hierarchies or relations between instances of classes. Constraints are expressed as axioms in description logic and are not represented graphically. Tools and languages for ontologies also take advantage of the graph structure.

Though ontologies help interoperability, they impose a new kind of burden on systems (and people) – how to define the ontologies of interest and, moreover, to specify how they should be managed. In many application domains, experts construct the ontologies using toolkits, often importing and reusing existing structures. Reuse is fostered

by publishing ontology repositories on the Web – e.g., DOME [16], OntoServer [55] or Swoogle [17].

There are many tools to manipulate ontologies, with varying number of functionalities, such as ontology development, merge, evaluation, annotation, storage and querying. The most popular tools are Protégé, WebODE, OilEd, Ontolingua and OntoBuilder [24, 44]. Though many of these tools similar functions, they neither interoperate nor cover all the activities of the ontology life cycle. This lack of interoperability causes significant problems when integrating an ontology into the ontology library of a different tool, or if two ontologies built using different tools or languages are integrated using merging tools. This, in turn, prompted research on tools that process and allow comparison of ontologies specified in different languages – e.g., [58]. As will be seen, Aondê supports the most common functions offered by toolkits, assuming language homogeneity (the OWL standard).

3.2. Ontology Servers and Frameworks

Toolkits are self-contained and their operations cannot be accessed from external applications. Thus, to effectively share ontologies across applications, software engineers rely on two kinds of solution: ontology frameworks and ontology servers.

Examples of frameworks are Jena [14], SNOBASE [34] and SOFA[2]. Usually, such frameworks provide functions to access ontologies that have been stored in distinct formats (such as DAML+OIL, RDF, RDF(S), or OWL), using specific query languages.

Application development in this context uses a framework's functions and data structures. In many cases, however, frameworks do not support the development of applications that consider ontology evolution. Indeed, since ontologies describe knowledge about a given domain, they must evolve to reflect knowledge acquisition – e.g., in biodiversity, when new species are found or taxonomies are revisited. Since the application code depends on the framework's (static) structures, ontology evolution may demand considerable recoding. This defeats the purpose of using ontologies to provide flexibility and interoperability across applications.

Hence, ontology servers have been proposed to solve the need for dynamic management [19, 35, 52]. Servers are mostly concerned with handling storage issues. Some of these servers provide access to the ontologies via their URIs – e.g., [19], while others store the ontologies in a local repository – e.g., [35]. These servers can only provide access to an ontology at a time, and thus are not appropriate to work in distributed, multi-ontology, scenarios. Moreover, they offer a limited range of functions on ontologies – some may support queries to ontologies and, in some cases, provide inference engines.

Aondê combines the use of servers (for dynamic ontology management) to frameworks (to support flexibility in application development). The following sections briefly describe some of the ontology functions that have been proposed in the literature, as well as toolkits that have been developed to deal with them, indicating some of our design choices. As will be seen, there is no solution that covers all needs. Moreover, some of these functionalities are not yet available in a Web implementation. Aondê fills this gap, providing web based support to the most common functions.

[2] http://sofa.dev.java.net

3.3. Ontology Ranking

The goal of ranking mechanisms is to determine ontologies that are potentially relevant to a given knowledge domain. Some mechanisms use metrics that consider the number of links and references among ontologies, similar to the notion of page-ranking on the Web. This solution suffers from limitations, since available ontologies seldom point at each other, having low inter-connectivity. Another solution to ranking is to analyze the internal structure of an ontology. This approach is based on metrics that evaluate how an ontology represents the concepts of interest, considering its class hierarchies and properties.

Figure 3.3 synthesizes the main features of ranking tools Swoogle [17], AkTiveRank [3] and OntoKhoj [43]. It shows, for instance, that AkTiveRank uses structural analysis, while the other two adopt reference count.

Tool	Swoogle	AkTiveRank	OntoKhoj
Approach	References among ontologies	Internal structure analysis	References among ontologies
Technique Used	Reference count	Structural analysis metrics	Reference count
Ontology Language	OWL, RDF, RDF(S), DAM+OIL	OWL	RDF, DAML+OIL, OWL
Searched Elements	All elements	Classes	All elements

Figure 2. Comparative table of ontology ranking tools - shaded boxes indicate our design and implementation options

3.4. Computing Differences between Ontologies

Difference computation, in most cases, compares two versions of the same ontology. In the more general case, distinct ontologies can be compared. Simple differences are those that do not consider an ontology's structure – i.e., concerning only names of classes or properties, data types and constraints. Complex differences include detecting modification of class hierarchies (e.g., when a class changes place in a hierarchy, denoting its semantics have changed).

There are several tools that handle difference computation, automatically or semi-automatically (i.e., with or without user interaction). Figure 3.4 compares a few aspects of tools PromptDiff [39] and OntoDiff [54]. It shows, for instance, that PromptDiff compares structures and hierarchies, but does not take ontology instances into account. Shaded boxes indicate the design and implementation options adopted by Aondê.

3.5. Ontology Views

Ontology reuse is a highly recommended practice when creating a new ontology. Reuse provides several advantages, such as avoiding the hard work of building an ontology from scratch. Furthermore, existing ontologies are supposed to be sounder, since they will have undergone checking by domain experts and tested by other applications. Finally, reuse fosters data integration and interoperability among applications that use the same ontology.

Tool	PromptDiff	OntoDiff
Approach	Structural comparison	Structural and instance comparison
Technique Used	Structural and hierarchical analysis	Heuristics for change detection
Support Environment	Protégé	None
Ontology Language	RDF(S), OWL	RDF(S)
Processing	Semi-automatic	Automatic

Figure 3. Table comparing features of tools that detect differences between ontologies

More often than not, however, an ontology may be too large or too complex for a given application need – usually, applications will require only part of an ontology. However, for lack of available alternatives, application developers end up by importing entire ontologies, which creates performance problems, both in space and processing time. This is aggravated by the fact that ontology management and processing usually depends on inference engines, which perform poorly on large ontologies.

A solution to this problem is to use an *ontology view*, which is defined to be a relevant subset of an ontology. The term *view* is borrowed from research on databases – a view is a portion of a database, relevant to a given user or application, extracted by applying a query against the database. Paraphrasing [2], for the user, the view is a "stand-alone" database created from the original database. Similarly, an ontology view is a "stand-alone" ontology, constructed by extracting parts of an ontology and using them as a new ontology.

Figure 3.5 presents the main features of tools OntoPathView [30], View Language [56], View Traversal [41] and Ontology Winnowing [4] used in view creation. The figure shows, for instance, that approaches proposed in [30, 41] are based on the notion of *central concept* – a class around which the view is built and that defines which ontology elements must be part of a view. This approach, chosen by use, is more flexible than that of constructing views using queries.

3.6. Ontology Integration

In distributed and open systems, ontologies alone cannot solve interoperability and heterogeneity issues. Distinct research groups may have different interests, research goals, use diverse computational tools and manipulate knowledge at various levels of detail and abstraction. Thus, in order to provide group cooperation, some kind of ontology integration mechanism must be provided. Several of the heterogeneity issues considered have already been studied in research on database integration (e.g., [11]).

Approaches to ontology integration start from two ontologies o_1 and o_2, and include [13, 32]:

- **Mapping:** preprocessing stage, identifies all concepts in o_1 and o_2 that are identi-

Tool	OntoPathView	View Language	View Traversal	Ontology Winnowing
Approch	Notion of central concept	Based on ontology query languages	Notion of central concept	Automatic view extraction based on ontology queries
Technique Used	Specifies central concept and related elements	Extends RQL	Specifies central concept and related elements	Analysis most frequently queried elements
Ontology Language	RDF(S)	RDF(S)	RDF, RDF(S), OWL	RDF(S), OWL

Figure 4. Comparison of view creation tools - shaded boxes indicate our design and implementation choices.

cal, using matching techniques;

- **Merge:** constructs a new ontology that is based on the mappings between o_1 and o_2, merging equivalent concepts into a new concept. This concept receives the name of the originating concept in o_1 or in o_2;
- **Alignment:** constructs a new ontology that embeds and preserves the original ontologies, which are linked according to the mappings detected.

Integration is always based on some sort of matching process. Matching may identify identical terms (equivalence), or elements that participate in relationships (e.g., part-of, is-a). According to [49], there are two main classifications of matching techniques: element-level matching and structure-level matching. The first computes similarity among terms ignoring their relationships with other terms (e.g., using string matching or linguistic relationship among the terms compared). The second considers that an ontology is a graph structure, and analyzes how an entity appears in this structure [1, 46]. Aondê combines both techniques.

Figure 5 synthesizes a comparative analysis of relevant characteristics of integration tools GLUE [18], Chimaera [38], ODEMerge [47], PROMPT [40] and CATO [20]. Most of these tools are coupled to toolkits, such as Protégé (PROMPT) or Ontolingua (Chimaera). Automatic processing tools may generate incorrect mappings, while interactive processing may impose on the user the burden of manually checking all suggested mappings. All of these tools only match elements of the same kind, i.e., classes with classes, properties with properties, and instances with instances. As will be seen, we have designed and implemented an integration (alignment) module that permits another kind of matching: classes with instances.

4. The Aondê Ontology Service

The goal of the Aondê Ontology Service is to provide to client applications facilities to invoke a wide range of operations on ontologies. It was implemented as a Web service, thereby providing interoperability.

Web services are self-describing and modular business applications that provide business logic as services over the internet through standards-based interfaces and internet protocols (e.g. HTTP), with the purpose of finding, subscribing and invoking those services [7]. Standards adopted in their specification and implementation include XML,

Tool	GLUE	Chimaera	ODEMerge	PROMPT	CATO
Kind of Integration	Mapping	Merge	Merge	Alignment and Merge	Alignment
Techniques Used	Taxonomy-based, string-based (similarity) and constraint-based	Taxonomy-based, string-based (similarity)	Based on linguistic resources (synonyms, hyperonyms)	Taxonomy-based, string-based (similarity), constraint-based and graph-based (Anchor-PROMPT)	Taxonomy-based, and based on linguistic resources (synonyms)
Support Environment	None	Ontolingua	WebODE	Protégé	None
Ontology Language	XML (taxonomy)	Ontolingua, XOL	RDF(S), DAML+OIL	RDF(S)	OWL
Processing	Semi-automatic	Semi-automatic	Automatic	Semi-automatic	Automatic
Kind of Element Matching	Class-Class	Class-Class, Property-Property	Class-Class, Property-Property	Class-Class, Property-Property, Instance-Instance	Class-Class

Figure 5. Comparison of some ontology integration tools

SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language) and UDDI (Universal Description, Discovery and Integration). Web services facilitate the communication between distinct applications and platforms.

4.1. A two-level Web service architecture

Our ontology management architecture distinguishes between ontology persistence issues and semantic manipulation. Figure 6 shows this basic architecture, expanding the boxes that were outlined in Figure 1. Ontologies are stored in several distributed ontology repositories, each of which accessed via a Web service (WS). These repositories can be of two kinds: *Semantic Repositories*, built and managed by our Semantic Repository services, and third party *External Ontology Repositories* that publish ontology data via Web services. Aondê provides an extensible set of modules that can be invoked by client applications to search, rank, query, integrate, create views and compare ontologies.

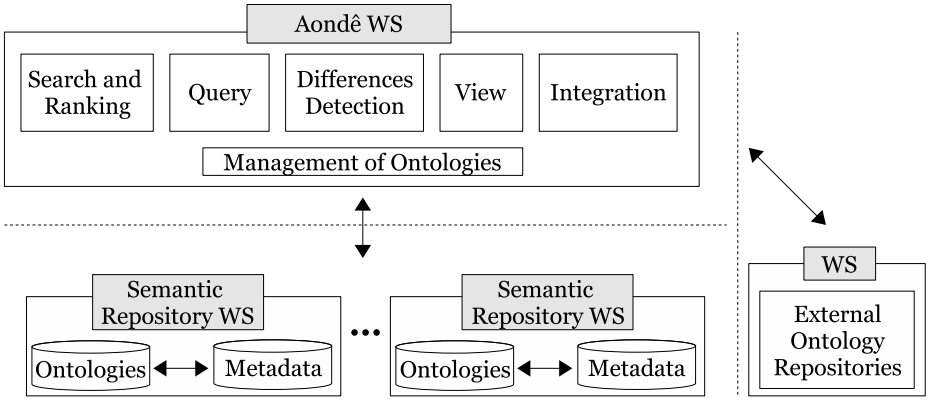

Figure 6. Two-tier architecture for managing ontologies, separating the persistence layer from the semantic operations of Aondê

Client applications can either request these high-level operations from Aondê, or directly access the Semantic Repositories using their service interfaces (e.g., allowing expert data curators to validate or update ontologies and their metadata). The sections

that follow describe the operations provided by Aondê (Section 4.2) and the Semantic Repository Service persistence facilities (Section 4.3).

4.2. Operations offered by Aondê

Aondê is organized in the following modules: *Management of Ontologies* (Section 4.2.1), *Search and Ranking* (Section 4.2.2), *Query* (Section 4.2.3), *Views* (Section 4.2.4), *Integration* (Section 4.2.5) and *Differences Detection* (Section 4.2.6). This choice of functions was based on our study of ontology tools, frameworks and services (Section 3), and of several biodiversity systems (e.g. SinBiota [50], Spire [42] or GBIF [23]). This was complemented by a process of requirements elicitation conducted with the biologists that work in WeBios.

4.2.1. Management of Ontologies Module

This module is responsible for in-memory management of ontologies. It mediates requests from the other modules to the persistence services, and vice versa – i.e., issuing requests to insert/delete/replace/retrieve ontologies and their metadata, as well as additional information (see Section 4.3). Ontologies obtained from the repositories (either from Semantic Repositories or third party External Repositories) are transformed by this module into in-memory structures, to be processed by Aondê's other modules.

This module is also responsible for constructing metadata structures for ontologies created by Aondê or retrieved from third party repositories. Each metadata structure is associated with an ontology through its identifier; an ontology and its metadata structure are stored together.

A Semantic Repository may contain several ontologies and their metadata. Each ontology has a unique (local) identifier within a repository, but a given ontology (and thus its metadata structure) may be stored in more than one Semantic Repository. This happens, for instance, when distinct semantic repositories are managed by different research groups. For this reason, unique identification requires the pair $<$*identifier, URLRep*$>$, which denotes the *identifier* of an ontology in repository addressed by *URLRep*. The term *idOntology*, used in the rest of the paper, designates this pair, whereas *identifier* will refer to a local id.

4.2.2. Search and Ranking Module

This module searches, within a set of source repositories, for ontologies that contain certain terms. It returns a set of ontologies that have classes or instances whose names match (exactly or partially) these terms. The search is performed on repositories designated by the invocation. If the search returns an ontology that is stored in third party repositories, Aondê processes it and stores it into a target Semantic Repository.

There are two kinds of search operation: with and without ranking. Search without ranking is performed to retrieve a single ontology in a specific repository, returning the $idOntology$ of the first ontology found containing the terms provided. There are two source options: generic ontologies, or taxonomic ontologies in biology, respectively:

$$Search(term, sourceRep, targetRep) \text{ and}$$

$$SearchTaxon(taxon, \{directive\}, sourceRep, targetRep),$$

where $term$ is the name searched for (and $taxon$ a scientific species name). The field $sourceRep$ denotes the source repository indicated by the client application and $targetRep$ designates the Semantic Repository where ontologies retrieved from third party external repositories are to be stored.

The search on a taxonomic ontology may return just part of an ontology – $\{directive\}$ specifies which elements will be included in the result. This kind of search helps applications discover taxonomic relationships among species when the source repository contains taxonomic classifications. Directive options can be *ancestors*, *siblings* and *descendants*.

An invocation of a search with ranking has the form:

$$SearchRank(\{term\}, \{weight\}, \{sourceRep\}, targetRep),$$

where $\{term\}$ represents the set of ontology terms passed as search parameters, and $\{weight\}$ indicates the set of weights for ranking metrics. This invocation returns a set of $idOntology$ values corresponding to result ontologies, ordered by ranking values.

Ranking is based on analysis of the internal structure of each ontology retrieved by the search. This analysis applies the metrics proposed by Alani et al. [3]: centrality of the classes, matching and density of classes and instances indicated in $\{term\}$, and the semantic similarity among these classes. These metrics are combined using the values in $\{weight\}$. The sum of weights must be 1.

We adapted the metrics proposed by [3] to rank ontologies – theirs only considers classes, whereas Aondê considers classes and instances. Let I denote a set of input terms (classes or instances) passed as search parameters, and O a set of source ontologies.

- **Exact and Partial Matching:** compares each term in I with each ontology $o \in O$. Computes the number of terms in I that exactly or partially match instances, or names of classes, in o;
- **Density:** applied to each class/instance $t \in o$ whose name matches a term in I. Computes the number of subclasses, superclasses, properties and siblings of t, when it is a class; and superclasses, properties and other instances of the same class, when t is an instance;
- **Centrality:** applied to each class $C \in o$ whose name matches a term in I. Computes all shortest paths between all pairs of classes in o and counts how many times C appears in these paths. The bigger the centrality of a class in an ontology, the higher the probability that the ontology adequately represents the concepts concerning the class, and the richer the ontology as regards this class;
- **Semantic Similarity:** applied to pairs of classes (or terms) $t_1, t_2 \in o$ whose names match a term in I, returns the length of the shortest path between t_1 and t_2. The shorter this path, the more similar the corresponding concepts.

4.2.3. Query Module

Given a source ontology and a query expressed in an ontology query language, this module returns the query result in the format requested. Its invocation has the form:

$$Query(idOntology, language, queryString, inference, outFormat),$$

where *idOntology* denotes the ontology to be queried (i.e., the pair $<$*identifier, URLRep*$>$ – see 4.1) and *outFormat* defines the output format for queries: it can be textual (i.e., in RDF triples or parts thereof) or structured in XML files.

Field *language* is the query language used to specify the query stated in *queryString*. This module allows queries in RDQL [48] or SPARQL [45] languages. Both RDQL and SPARQL query an RDF representation of an ontology, manipulating it as sets of *subject–predicate–object* triples. A query consists in selecting which ontology triples satisfy the query predicates.

The $inference$ parameter is a boolean variable that indicates whether the query should be executed on the source ontology, or on an extended ontology derived from it using inference mechanisms. Inference mechanisms support discovery of new ontological facts. Hence, queries with inference would normally return more results that are meaningful. However, they are more costly (in processing time) than normal queries. For this reason, inference is optional, defined at invocation time.

4.2.4. View Module

This module constructs a view of a source ontology, based on the central concept approach [41] – see Section 3.5 – and stores the view in a target Semantic Repository. The original central concept proposal of [41] constructs views using instances and properties of the source ontology. We extended this approach to manipulate class axioms. The module returns the identifier of the ontology view. An invocation of this module has the form:

$$View(idOntology, concept, \{directive\}, targetRep),$$

where $idOntology$ denotes the source ontology, and $concept$ is the name of the class that represents the central concept of the view. The view is stored in Semantic Repository $targetRep$.

The $directive$ field indicates the elements to incorporate into the view: instances, axioms or property names. These properties can be a subclass, a superclass, an object type or a data type. Besides the class that represents the central concept, classes related with it through axioms or properties are also included in the view. Each property name in a directive is associated with a non-negative integer that specifies its depth (how many levels of indirection must be inserted in the view). A request without directives produces a view containing everything associated to the central concept, with unlimited depth.

4.2.5. Integration Module

This module integrates two source ontologies, and produces a new ontology that is stored in a target Semantic Repository. It returns the $idOntology$ of the new ontology. Alignment is the integration approach chosen (see Section 3.6), since it is expected that almost all of the ontologies required by a biodiversity client application will have complementary or overlapping domains. Two ontologies with different coverages can be used together to

improve the description of the world. Alignment can also translate facts between ontologies with different granularities and show the same fact under distinct perspectives. Merging (an alternative approach) does not preserve the source ontologies in the result and is thus less adequate to our scientists.

Similarity computation is performed in two steps: first, possible mapping candidates are identified using element based techniques; next, structure techniques are employed. The goal is to avoid aligning classes or instances that have similar names, but which belong to distinct contexts in the ontologies.

Two element similarity techniques are applied: use of dictionaries (to find synonyms) and the Jaro metric [15]. The maximum value obtained is multiplied by α – see equation 1. Synonym comparison returns 1 (positive) or 0.

Structure similarity between two terms considers four kinds of factors: their properties, axioms, superclasses and subclasses. Each such analysis contributes with a weight of 0.25 to structure similarity computation. Property similarity compares the labels and the elements related by the property (classes or primitive types). For axioms, the comparison is performed on properties and classes involved. Hierarchy similarity compares super and subclasses common to the compared elements. The similarity value of each analysis is proportional to the ratio (*number of similar elements / total of elements compared*) – e.g., when comparing the superclasses of two terms, the ratio (*number of similar superclasses / total number of superclasses*). It must be stressed that structures may differ for similar elements, when ontologies have different granularities.

We define the *confidence* of a mapping between a pair of elements from different ontologies to be computed by:

$$confidence = \alpha * max\{similarElement\} + \beta * (similarStructure) \quad (1)$$

Field $\{$*similarElement*$\}$ contains the set of values obtained from similarity computation using two distinct element-based techniques, whereas *similarStructure* represents the similarity computed between the structures of these elements.

An invocation of the Integration module has the form:

$$Integration(idOntologyA, idOntologyB, \alpha, \beta, minConfidence, targetRep),$$

where $idOntologyA$ and $idOntologyB$ are the ontologies that will be aligned, and the result is stored in Semantic Repository $targetRep$. Parameter *minConfidence* – a number between 0 and 1 – defines a lower bound to the computation of formula 1. The final aligned ontology will only include the alignments for which this formula yields values above *minConfidence*

4.2.6. Differences Detection Module

This module performs a comparison between two ontologies, detecting their structural differences (classes and properties) and content differences (instances). An invocation of this module has the form:

$$Difference(\textit{idOntologyA}, \textit{idOntologyB}, \alpha, \beta, \textit{minConfidence}),$$

where *idOntologyA* and *idOntologyB* represent the ontologies to be compared. Field *minConfidence* is computed considering element and structure similarity, using formula 1. A mapping between two elements of the ontologies must have at least this value to be effectively considered in difference computation. Similarity computation uses the same steps of Integration – Section 4.2.5.

The result is an XML file that enumerates these differences as the union of three sets: concepts that are similar in B and in A, elements that are in B but not in A and vice-versa – see example in Section 6. We chose this kind of output because we were unable to execute difference operations using tools available on the Web. Moreover, papers discussing ontology difference present results only graphically (e.g., crossing off eliminated elements in an ontology graph) – see Section 3. Since Aondê must serve XML files to client applications, we opted for this solution.

This function has important applications in biodiversity systems, especially when ontologies A and B are different versions of a taxonomic description. In biodiversity, the classification of species may change over time. Moreover, classifications proposed by distinct authors may disagree on species hierarchies. This comparison allows a biologist to detect when distinct taxonomic models were used in a given study, and where the differences lie.

4.3. Semantic Repositories Service

While the previous section concentrated on semantic manipulation of ontologies, the rationale behind the Semantic Repository Service is to provide persistence to ontologies, and low-level ontology storage manipulation. Nowadays, a very wide range of ontologies is available on the Web. Most times, they are published without any additional information that might help their use. As a consequence, potential users cannot find and recognize ontologies of interest, and thus their sharing and reuse are seriously affected. We introduced metadata structures to help diminish this problem.

The Web service interface to Semantic Repositories thus supports extraction, insertion, deletion and replacement operations on ontologies and associated metadata. Its operations and structures extend facilities provided by ontology servers.

A Semantic Repository contains two data spaces: (1) for ontologies and their metadata structures; and (2) for usage information. The latter records data concerning usage patterns of ontologies within the repository – i.e., queries performed against them, Aondê operations used to create them, and provenance data not provided by the metadata structure. This can contribute to optimize ontology management, and also help domain experts in finding out more about ontology usage. The usage data space is still being designed, and will not be discussed here.

All ontologies manipulated by Aondê are stored in Semantic Repositories, including those extracted from third party repositories. The underlying assumption is that ontologies are directly associated with end-user/client application needs and they should be available to future requests. The metadata structure of an ontology is built by the *Management of Ontologies* module (see Section 4.2.1).

Storage manipulation primitives insert, delete or replace entire ontologies and/or their metadata. These primitives use coarse-grained operations: our unit of storage is an

entire ontology (or its metadata structure). Thus, they are our retrieval and update units. Our main goal, at this stage, is to support high-level semantic manipulation at Aondê. Thus, we did not concern ourselves with finer granularity operations at the storage level (e.g., to retrieve or replace parts of an ontology). This has an impact on performance, since real ontologies can be very large.

On the other hand, it does facilitate semantic manipulation. For instance, since Aondê constructs views based on the central concept paradigm, it needs an entire ontology in order to compute a view. The same applies to queries: rather than considering a query to be low-level storage operation, Aondê requests an entire ontology from storage, to perform queries afterwards – e.g., with or without inference. This design decision on storage operations may evolve in the future, when we finish the design of the usage space, and obtain data on user profiles.

Figure 7 illustrates the structure of a Semantic Repository, composed by the ontology and metadata data space, and the usage data space. The figure shows that each ontology stored has an associated metadata structure, indicating, for instance, its URI, the creation date, and associated keywords.

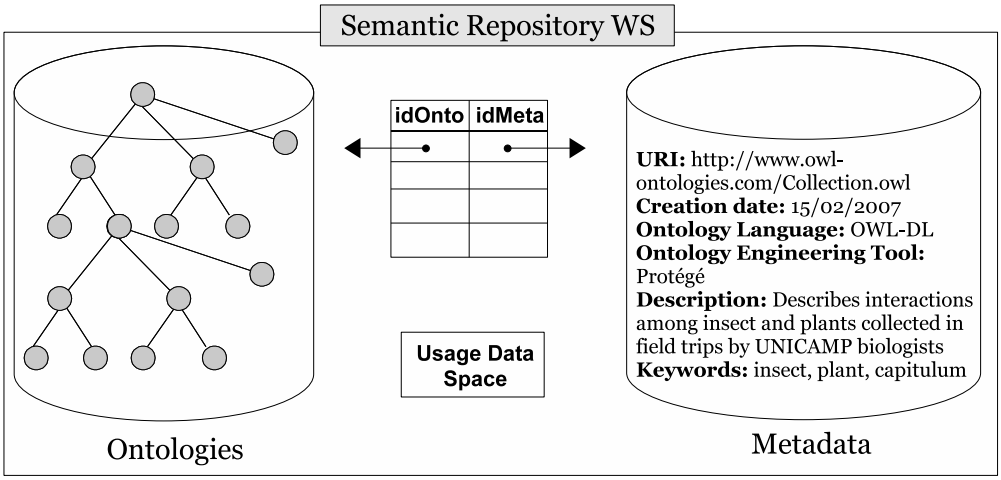

Figure 7. Example of a Semantic Repository

OWL [8], the standard recommended by the W3C Consortium, was adopted to represent ontologies. We have adopted the OMV (Ontology Metadata Vocabulary) standard [28] to provide metadata information. The metadata structure illustrated in figure 7 uses OMV terminology, detailed in Section 5.

Since we are concerned with biodiversity issues, the ontologies of the Semantic Repositories store concepts about: (1) geographical features, (2) biological information and (3) other kinds of information that is relevant to research in biodiversity (e.g. concerning images). Biological ontologies provide descriptions about taxonomy, evolution and morphology of species, as well as ecological and trophic relations (i.e., position occupied by a species in a food chain). Ontologies on geographical features are associated with terms that describe geographical, climatological and environmental characteristics of a region, as well as other natural or artificial elements that may have an impact on an ecosystem.

5. Implementation Aspects

Aondê has been implemented in the Java language. Access and navigation over ontology contents are provided by the Jena framework [14] version 2.4. This version of Jena is composed by an RDF API, an OWL API, in-memory and persistent storage, SPARQL and RDQL query engines and a rule-based inference engine.

Our Web service implementation uses Apache Axis, a widely adopted open source Web service framework. It consists of a Java implementation of the SOAP server, and various utilities and APIs for generating and deploying Web service applications. Sections 5.1 and 5.2 describe persistence details, and section 5.3 the implementation of Aondê's modules.

5.1. Semantic Repositories: Ontology and Metadata Structures

A Semantic Repository is composed of two data spaces: ontology and metadata; and usage data. The second data space is under construction, and outside the scope of this paper (see Section 4.3). The first space contains [*ontology, metadata*] pairs of files, where the ontology is expressed in OWL and the associated metadata structure in OMV. OWL supports description of concepts with distinct granularity details (OWL Lite, OWL DL and OWL Full) – one of the reasons for which it was chosen, since the ontologies that can be manipulated by the service are not known beforehand.

The ontologies used to test Aondê were built in Protégé [24]. This tool has a plug-in to manipulate ontologies in OWL, with a graphic interface to define classes, properties, instances and axioms in description logics.

OMV files are represented in RDF, using the namespace **omv** (*http://omv.ontoware.org/omv-core-1.0#*). Metadata elements are categorized according to their type and purpose as follows:

- **General:** elements providing general information about the ontology: URI, name, acronym, description, creationDate (*required*) and documentation, keywords, status, modificationDate (*optional*);
- **Availability:** information about ontology location: resourceLocator, version (*required*) and hasLicense (*optional*);
- **Applicability:** elements describing the intended usage or scope for the ontology: isOfType (*required*) and hasDomain, naturalLanguage, designedForOntologyTask, hasFormalityLevel (*optional*);
- **Format:** information about the physical representation of the ontology, including the representation language in which the ontology is formalized: hasOntologyLanguage, hasOntologySyntax (*required*);
- **Provenance:** elements about the organizations that contributed to the creation of the ontology: hasCreator (*required*) and hasContributor, usedOntologyEngineeringTool, usedOntologyEngineeringMethodology, usedKnowledgeRepresentationParadigm (*optional*);
- **Relationship**(*optional*)**:** information about relationships with other ontologies: useImports, hasPriorVersion, isBackwardCompatibleWith, isIncompatibleWith;
- **Statistics**(*required*)**:** various metrics on the graph that underlies the ontology: numClasses, numProperties, numIndividuals, numAxioms.

Figure 8 presents OMV metadata for one of the ontologies developed by us, which will be used in our case study. It shows, for instance, that the engineering tool used to build this ontology was Protégé (5th line from bottom), and that description logics was used for knowledge representation (preceding lines). In addition, it contains facts such as: the ontology has 5749 individuals, 22 classes, and was created in 2007-02-15.

```
<rdf:RDF
   xmlns:xsd="http://www.w3c.org/2001/XMLSchema#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:omv="http://omv.ontoware.org/omv-core-1.0#">
  <rdf:Description
       rdf:about="http://www.owl-ontologies.com/Collection.owl#">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  <rdf:type rdf:resource="http://omv.ontoware.org/omv-core-1.0#Ontology"/>
       <omv:hasOntologySyntax>OWL-XML</omv:hasOntologySyntax>
   <omv:hasCreator>Computer science and biology researchers from
   UNICAMP</omv:hasCreator>
   <omv:resourceLocator>unknown</omv:resourceLocator>
   <omv:description>Describes interactions among insects and plants
       collected in capitula Zoology department, biology institute.
       </omv:description>
   <omv:URI>http://www.owl-ontologies.com/Collection.owl#</omv:URI>
   <omv:hasOntologyLanguage>OWL</omv:hasOntologyLanguage>
   <omv:version>1.0</omv:version>
   <omv:numClasses>22</omv:numClasses>
   <omv:numProperties>11</omv:numProperties>
   <omv:numAxioms>6</omv:numAxioms>
   <omv:numIndividuals>5749</omv:numIndividuals>
   <omv:acronym>colUN</omv:acronym>
   <omv:creationDate>2007-02-15</omv:creationDate>
   <omv:isOfType>Domain Ontology</omv:isOfType>
   <omv:name>collection.owl</omv:name>
   <omv:keywords>insect, plant, herbivore, capitulum</omv:name>
   <omv:usedKnowledgeRepresentationParadigm> Description Logics
       </omv:usedKnowledgeRepresentationParadigm>
   <omv:usedOntologyEngineeringTool>Protégé
       </omv:usedOntologyEngineeringTool>
   <omv:naturalLanguage>English</omv:naturalLanguage>
  </rdf:Description>
</rdf:RDF>
```

Figure 8. OMV Metadata for one of the ontologies used in the case study

5.2. The Semantic Repository Web Service

We had several choices for implementing Semantic Repositories. We adopted Jena to access ontologies because its persistence engine allows RDF graphs to be stored in relational databases. Thus, all data in Semantic Repositories are actually stored in databases. We chose the PostgreSQL database management system to implement persistence, where both ontologies and their metadata are stored as RDF triples.

We conducted tests on 3 distinct semantic repositories, building a Web service for each; the largest ontology contains 5749 instances. In implementations of Web services, invocations and results are SOAP messages with attached OWL/RDF files. Ontology (and metadata) insertion in Jena requires the client application to create the ontology identifier that is to be inserted in Jena's database. If the identifier provided already exists in the database, the insertion operation will not be accepted. This is a problem, since it

places on the client application the burden of "inventing" identifiers for new ontologies and associated metadata. To solve this, our ontology/metadata insertion code generates unique identifiers.

The operations allowed by a Semantic Repository Web Service are:

- Insertion, replacement and deletion of an ontology: receives an ontology identifier and, in case of insertion and replacement, an $idOntology$, or an OWL file containing the ontology to be inserted or to replace the ontology identified;
- Insertion, replacement and deletion of a metadata structure: receives the identifier of the ontology and, in case of insertion and replacement, the URL of a remote metadata file or an RDF file;
- Retrieval of an ontology and/or associated metadata structures: receives an ontology identifier, builds the corresponding OWL/RDF file and returns it;
- List ontologies: lists all ontology identifiers stored in a repository;
- Requests on usage information – retrieve/insert/delete/modify usage information (under design).

Figure 9 presents the WSDL specification of method *ManagementOntology*, that performs insertion and replacement of ontologies. The two $<$*wsdl: message*$>$ sections at the beginning respectively indicate the parameters of the SOAP message sent to the service, and the output parameter. For instance, a *ManagementOntologyRequest* message to the service (3rd line) must indicate a repository URL, an operation ("insertion" or "replacement"), an identifier and an OWLFile. It returns a string to indicate if it was successful or not. A failure occurs when a replacement invocation has an identifier that is not in the database. Other methods are similarly specified.

5.3. Implementing Aondê

Aondê was implemented as a Web service. We recall that all operations are performed in memory, after retrieving ontologies from a persistent storage structure, either from Semantic Repositories or from third party External Repositories. All operations offered by Aondê's interface have a parameter that contains the (set of) URL denoting source Repositories and another URL where the result of the operation should be stored, when it creates an ontology – the *targetRep* parameter of the operations.

All modules have been tested on at least one Semantic Repository and at least one third party External Repository. Since there are no Web Services available to access biodiversity ontologies – just portals – third party repositories used in Aondê are accessed by HTTP requests. These requests have the form: "ServiceURI ?Parameters", where parameters are combined by connector "&".

Search and Taxonomic Search

Tests were conducted as follows. The external repository used to test taxonomic (non-ranked) search was the Spire portal[3] [42] accessed via URI *"http://spire.umbc.edu/ont/ethan-part.php"*. Since it does not offer a Web service interface, we had to implement this external access using HTTP requests, which return an OWL file. The OWL file is processed by Aondê (module *Management of Ontologies*) to generate an OMV metadata structure. The ontology and its metadata are stored in the

[3] http://spire.umbc.edu/ont/

```
<wsdl:definitions
targetNamespace="http://localhost:8080/axis/services/WSSemanticRepository
<wsdl:message name="ManagementOntologyRequest">
   <wsdl:part name="urlSemanticRepository" type="soapenc:string" />
   <wsdl:part name="operation" type="soapenc:string" />
   <wsdl:part name="identifier" type="soapenc:string" />
   <wsdl:part name="OWLFile" type="apachesoap:DataHandler" />
</wsdl:message>
<wsdl:message name="ManagementOntologyResponse">
   <wsdl:part name="ManagementOntologyReturn" type="soapenc:string" />
</wsdl:message>
<wsdl:portType name="WSSemanticRepository">
   <wsdl:operation name="ManagementOntology"
      parameterOrder="urlSemanticRepository operation ontName dh">
      <wsdl:input message="impl:ManagementOntologyRequest"
      name="ManagementOntologyRequest" />
      <wsdl:output message="impl:ManagementOntologyResponse"
      name="ManagementOntologyResponse" />
   </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="WSSemanticRepositorySoapBinding"
   type="impl:WSSemanticRepository">
   <wsdlsoap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http" />
   <wsdl:operation name="ManagementOntology">
   <wsdlsoap:operation soapAction="" />
   <wsdl:input name="ManagementOntologyRequest">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://webService_pckg.OntologyService" use="encoded" />
</wsdl:input>
<wsdl:output name="ManagementOntologyResponse">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/services/WSSemanticRepository"
use="encoded" />
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
</wsdl:definitions>
```

Figure 9. WSDL of ManagementOntology method

designated target Semantic Repository. If the search is instead conducted in a Semantic Repository, no ontology is created.

Ranked Search

The external ontology repository used by ranked search is the Swoogle Web service[4] [17], accessed via URI *"http://logos .cs.umbc.edu:8080/swoogle31/q"*. This search is implemented in three steps: (1) retrieval of ontologies and metadata from the source repository, (2) eventual storage of new ontologies into the target Semantic Repository, and (3) ranking. Step (1) is subdivided into two phases. First, the Swoogle "Search ontology" operation returns an RDF file containing URLs of ontologies and Swoogle metadata. Next, each of these ontologies is accessed by it URL. Step (2) stores Swoogle ontologies in the target Semantic Repository. In addition, Aondê automatically creates OMV metadata for each ontology, based on Swoogle metadata.

Finally, in step (3), the retrieved ontologies (that contain classes or instances whose names match search terms exactly or partially) are ranked and returned with their

[4] http://swoogle.umbc.edu

rank values, combining several metrics. Two of these metrics are based on computation of the minimal path between concepts in an ontology graph. This required implementing code that transformed an OWL ontology specification into a graph. The conversion of an ontology into a graph considers classes and instances as vertices and *is-a* relationships and class properties as edges.

Query

The Query method supports RDQL and SPARQL languages. We have selected Jena version 2.4. We did not use version 2.5 because it does not support RDQL queries. The reasoner adopted is the default OWL reasoner, which can be considered to be instance-based. It uses rules to propagate the *if* and *only–if* implications of the OWL constructs on instance data. Reasoning about classes is done indirectly – for each declared class a prototypical instance is created.

Views

The View method builds a view (a new ontology) extracting parts of the source ontology. The main difficulty in coding this method concerned graph traversal. Indeed, view construction requires navigation along source ontology paths to retrieve the elements defined by directives and rebuild these parts into the view. This navigation is more difficult when the domain or the range of parameters or axioms are composed by union or intersection of many elements. An ontology identifier is automatically created and returned by the method. We point out that views containing more than one central concept can be built by first creating separate views using the View method, and then aligning them using the Integration method.

Integration

The Integration method aligns two source ontologies, automatically creating an identifier for the alignment result. It is performed in two steps - first looking for string similarity, and then for structural similarity, which uses Jaro metrics [15]. Synonym identification uses the ITIS database (Integrated Taxonomic Information System)[5] [37] for comparison of biology taxonomic terms, and the WordNet dictionary [33] (version 3.0) for other comparisons. Mapping candidates are analyzed structurally as regards properties and axioms, to avoid cases in which classes with the same name, but with different contexts and meanings, are identified as similar.

In the resulting aligned ontology, similar concepts receive OWL tags. Alignments discovered between classes are represented by the <*owl:equivalentClass*> tag, and between instances by the <*owl:sameAs*> tag. Often, however, distinct ontologies differ in their level of detail in describing a domain's entities – e.g., a class in a given ontology has the same meaning as an instance in another. For this reason, Aondê also discovers matchings between classes and instances, and represents these alignments by tag <*owl:sameAs*>.

Our alignment techniques were adapted to deal not only with classes, but also with instances (in biodiversity, taxon terms). In many situations, scientific names of taxons only differ in their suffixes. For example, the strings *"Asterales"* and *"Asteraceae"* have high string similarity, although they refer to different taxons: the *ales* suffix indicates

[5] http://www.itis.gov

order and the *aceae* suffix indicates family. Furthermore, this kind of string needs specific synonym dictionaries, containing other scientific names used for the same taxon or vernacular names. For this reason, our alignment implementation uses the ITIS database [37] to check taxon synonyms. Since ITIS is only available on the Web in textual form, we created a relational database for its contents. This required downloading ITIS files, parsing them, and constructing the database via SQL insert commands.

Difference

The Difference method produces an XML file containing three lists, describing the result of comparing two input ontologies, $ontA$ and $ontB$. The first list contains all elements considered similar in $ontA$ and $ontB$. The second list contains all elements found in $ontA$ without any similar element in $ontB$, and the third contains elements found in $ontB$ but without corresponding element in $ontA$. Similarity is computed using the same techniques explained for the Integration method.

6. Case Study

Our case study uses real data, and was provided by biologists who work in the WeBios project. It involves interactions among insects and plants, with data collected by many teams over 20 years. This particular study is concerned with specific interactions between insects and a *capitulum* (the latin term for flower heads). This section uses a running example, which will concentrate on the need for resolving a request to *"retrieve insect species"* under different constraints. This will be enough to illustrate distinct possibilities for invocations of Aondê, showing how it solves typical end-user requests, including those not treated elsewhere. Each solution involves one or more invocations of operations offered by Aondê.

Our biology partners concentrate their research on plants of the family *Asteraceae*. The insects of interest for our running example are those of orders *Diptera* (flies) and *Lepidoptera* (butterflies), and belong to *endophagous* species – their larvae live within and feed upon flower head seeds. For a detailed description on data concerning biological aspects of the problem, the reader is referred to [5, 6, 21].

6.1. Constructing the Semantic Repository

The first step was to construct an ontology to describe the semantic backbone that our biology partners use in their work. Figure 10 shows a portion of this ontology, created in OWL, using the Protégé ontology engineering tool. Edges labeled with *io* connect classes with their instances, whereas edges labeled with *isa* indicate sub-/superclass relationships. All other labels correspond to properties (attributes or relationships).

To clarify our examples, we must briefly explain our experts' collection methodology to detect insect–plant interactions. They take field trips and collect flower heads (*capitulae*), which they take back to their university lab. Flowers may contain insect eggs or larvae, which will eventually hatch to become insects. It is only then (sometimes weeks after the field trip) that the insect can be identified. The collection unit is therefore the *capitulum*.

Figure 10 shows, for instance, that an insect of the *Tomoplagia reimoseri* species has an interaction with plant species *Vernonanthura membranacea*. This interaction is

obtained following the ontology associations between the insect and plant species via node *Capitulum G0904*. The figure also shows parts of the taxonomic tree of collected plant species – species *Vernonanthura membranacea* belongs to genus *Vernonanthura*, itself part of the *Asteraceae* family.

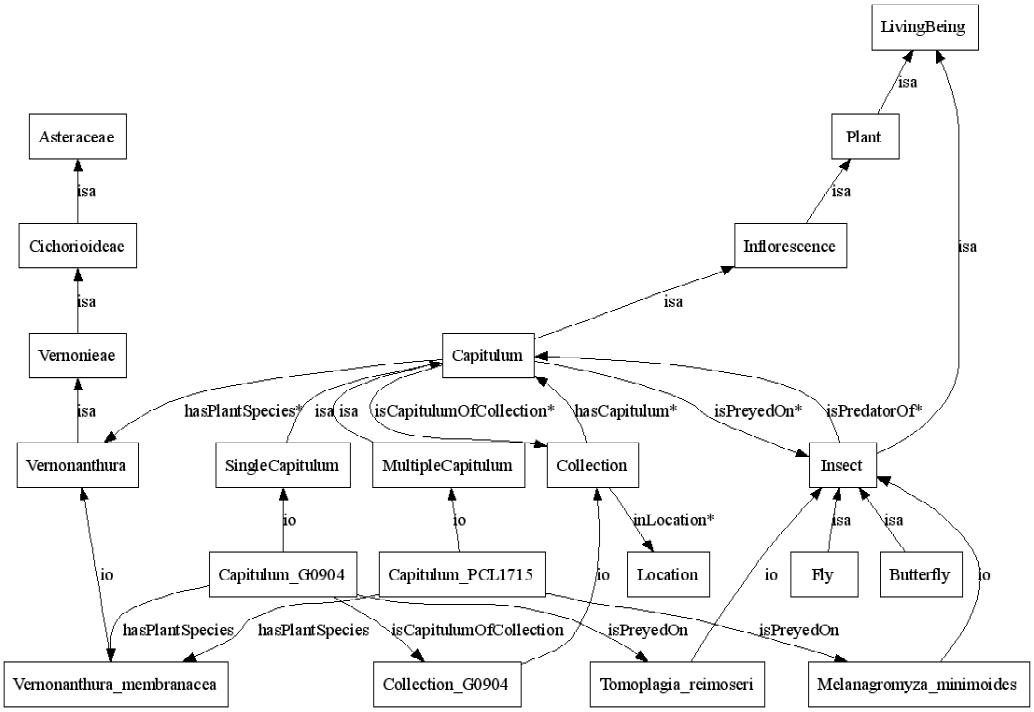

Figure 10. Part of the $colUN$ ontology, constructed with help of WeBios' biologists. It describes interactions among insects and plants, in the context of their collection methodology

We have omitted most instances of the ontology, to simplify the figure. It describes 1468 collection records, and interactions among 281 different insect species and 623 different plant species. A Semantic Repository was created to store this ontology and associated metadata. The repository URL is *http://143.0106.23.89:8080/OntRepUNICAMP*. The $idOntology$ in invocations of Aondê is $<$*colUN, http://143.0106.23.89:8080/ OntRepUNICAMP*$>$, where $colUN$ is the ontology local identifier within the repository designated by the URI.

6.2. Basic Query Scenario

Consider a typical query in the plant-insect interaction scenario: *"Return insect species that prey on plant species Eupatorium odoratum and that were collected in the Atlantic Rainforest"*. This requires several kinds of semantic disambiguation – what is an "insect", what does "prey on" mean, what is an "Eupatorium odoratum", what insect species were collected in field trips, what is the "Atlantic Rainforest" and its geographical extent. Each such consideration may require an invocation of Aondê, depending on data available on a cache or on local repositories (e.g., the meaning and extent of "Atlantic Rainforest" may have been precomputed elsewhere).

We will just retain, in the running example, the sub-query *"Retrieve insect species that prey on plant species Eupatorium odoratum"*. The other issues can also be dis-

ambiguated using Aondê, or by other means – e.g., through a combination of database queries, or even using user input (e.g., entering the coordinates of the Atlantic Rainforest polygon). For instance, collection records basically state facts: species name, observation date and methodology, habitat classification, GPS location, scientist responsible, and so on. Thus, a *"scientist = Lewinsohn"* predicate is normally resolved through an SQL query to the collection database.

Issues concerning combined ontological and (relational) database query processing are beyond the scope of this paper. Here, the assumption is that ontological disambiguation may be needed before or in parallel with query processing.

6.3. Ontology Query

Consider that biologists need the following information: *"Retrieve insect species that prey on plant species Eupatorium odoratum"*. This can be solved by sending a *Query* invocation to Aondê:

Query (<colUN,http://143.0106.23.89:8080/OntRepUNICAMP>, SPARQL, queryStr, false, XML)

Invocation parameters show that the query is specified in *SPARQL* and is to be executed against ontology *colUN* stored in repository pointed at by *http://143.0106.23.89:8080/OntRepUNICAMP*. Field *false* indicates that the query will not require inference. Figure 11 shows the contents of parameter *queryStr*. Prefix **http://www.owl-ontologies.com/Collection.owl#** represents *colUN*'s namespace. Parameter *XML* indicates that the result should be an XML file (standard result for SPARQL [12]).

```
PREFIX col:<http://www.owl-ontologies.com/Collection.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT distinct ?insect
WHERE {
?capitulum rdf:type col:Capitulum .
?capitulum col:hasPlantSpecies col:Eupatorium_odoratum .
?capitulum col:isPreyedOn ?insect }
```

Figure 11. Query *"Retrieve insect species that prey on plant species Eupatorium odoratum"* expressed in SPARQL

This query returns 18 insect species names (out of 281 instances). Figure 12 shows part of the resulting XML file.

6.4. Ontology View

The ontology of Figure 10 embeds a special knowledge about the collection methodology of our biologists: insect observations may have been recorded per flower head (one observation recorded per single *capitulum*) or per set of flowers (one observation recorded per multiple flowerheads). This kind of difference in methodology may have considerable impacts in conclusions drawn by experts. Consider, therefore, that the query in Section 6.3 has been modified and that now biologists want the following: *"Retrieve insect species that prey on plant species Eupatorium odoratum, for studies where species were counted*

```
<?xml version="1.0"?>
<sparql
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:xs="http://www.w3.org/2001/XMLSchema#"
   xmlns="http://www.w3.org/2005/sparql-results#" >
  <head> <variable name="insect"/>  </head>
  <results ordered="false" distinct="true">
    <result> <binding name="insect">
        <uri>http://www.owl-ontologies.com/Collection.owl#Cecidochares connexa</uri>
      </binding> </result>
    <result> <binding name="insect">
        <uri>http://www.owl-ontologies.com/Collection.owl#Melanagromyza minimoides</uri>
      </binding> </result>

                                    ...
   <result> <binding name="insect">
        <uri>http://www.owl-ontologies.com/Collection.owl#Phalonidia squalida</uri>
      </binding> </result>
    <result> <binding name="insect">
        <uri>http://www.owl-ontologies.com/Collection.owl#Unadilla erronella</uri>
      </binding> </result>
  </results>
</sparql>
```

Figure 12. Excerpt of XML file returned by Query module invocation

per single capitulum". One possibility to solve this query is to add more restrictions in the *where* clause of the SPARQL query of figure 11.

Another possibility is to restrict the ontology to consider only elements involving one *capitulum*. This alternative is executed through two successive invocations of Aondê: (1) create a view for insects that were observed per single *capitulum*; (2) pose query of Section 6.3 on this view. Step (1) is expressed as:

View (<colUN,http://143.0106.23.89:8080/OntRepUNICAMP>, SingleCapitulum,
{superclasses:1, isPreyedOn:2, isCapitulaeOfCollect:1, hasPlantSpecies:1, instances},
http://143.0106.23.89:8080/OntRepUNICAMP)

Parameter *SingleCapitulum* is the view's central concept. The view is extracted from *colUN*, in the repository *colUN,http://143.0106.23.89:8080/ OntRepUNICAMP*, and is to be stored in the same repository (last parameter). The parameters in the directive field are interpreted with respect to the central concept as follows: (a) *superclasses:1* – the view must include all immediate superclasses of *SingleCapitulum*, i.e., *Capitulum* – see Figure 10; (b) *isPreyedOn:2* include all classes whose distance to *SingleCapitulum* via *isPreyedOn* is either 1 or 2; (c) and (d) similar to (b), respectively for properties *isCapitulaeOfCollect* and *hasPlantSpecies*; (e) *instances* indicates that instances of all these classes must be included in the view.

The view generated is partially illustrated, with a few instances, in Figure 13. The two *Capitulum* nodes have been expanded to show a few details of the view's contents. Notice that the view contains relationships among instances (e.g., between *Collection_PIC02130* and *Capitulum_PIC02130*). Relationships that appear amongst view instances are those that were specified in the directives – i.e., the view will preserve not only relationships concerning classes, but also the same relationships among the instances. This implementation extends the views generated by Protégé, which only preserve rela-

tionships among classes.

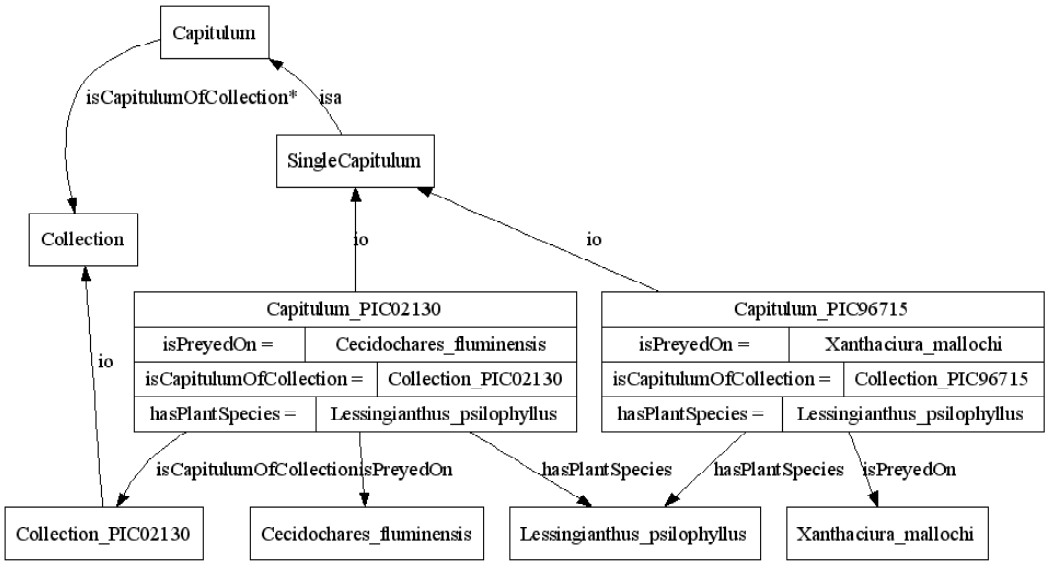

Figure 13. View with central concept *"SingleCapitulum"*

The service invocation returns an ontology identifier created for this view, *colUN-SingleCapitulum*. Once the view is created and stored, step (2) is executed, through another invocation of Aondê, a query to *colUN-SingleCapitulum*. In our study, this query returns two species names: *Xanthaciura mallochi* and *Cecidochares fluminensis*.

6.5. Ontology Search

Consider now that the biologists want to restrict their data to butterflies, i.e., *"Retrieve butterfly species that prey on plant species Eupatorium odoratum"*. Butterflies are insects of order *Lepidoptera*. However, *colUN* concentrates on insect species, and does not contain information on their taxonomic orders (a higher taxomonic level). Thus, there is a need to perform a search elsewhere to get this information.

In other words, this request can be solved through three invocations to Aondê: (1) search elsewhere for taxonomic information on *Lepidoptera*; (2) align the ontology retrieved in step (1) with the *colUN* ontology, to identify which species are butterflies; (3) invoke a query operation on the aligned ontology, using a query similar to the one presented in Section 6.3.

Step (1) is achieved by invoking Aondê with a *SearchTaxon* request, on external third party repositories – here, Spire[6] [42]:

SearchTaxon (Lepidoptera,{descendants},Spire,
http://143.0106.23.89:8080/OntRepUNICAMP)

Parameter *Spire* indicates that the search will be conducted in Spire, and *Lepidoptera* is the scientific name to be searched for. The directive *descendants* requests to return all lower taxons. The resulting ontology, partially illustrated in Figure 14, is a taxonomic hierarchy rooted at *Lepidoptera*. Notice that species names are not all at the same

[6]http://spire.umbc.edu/ont/

distance from the root. This occurs because some species have taxonomic subclassifications, such as subfamilies and tribes.

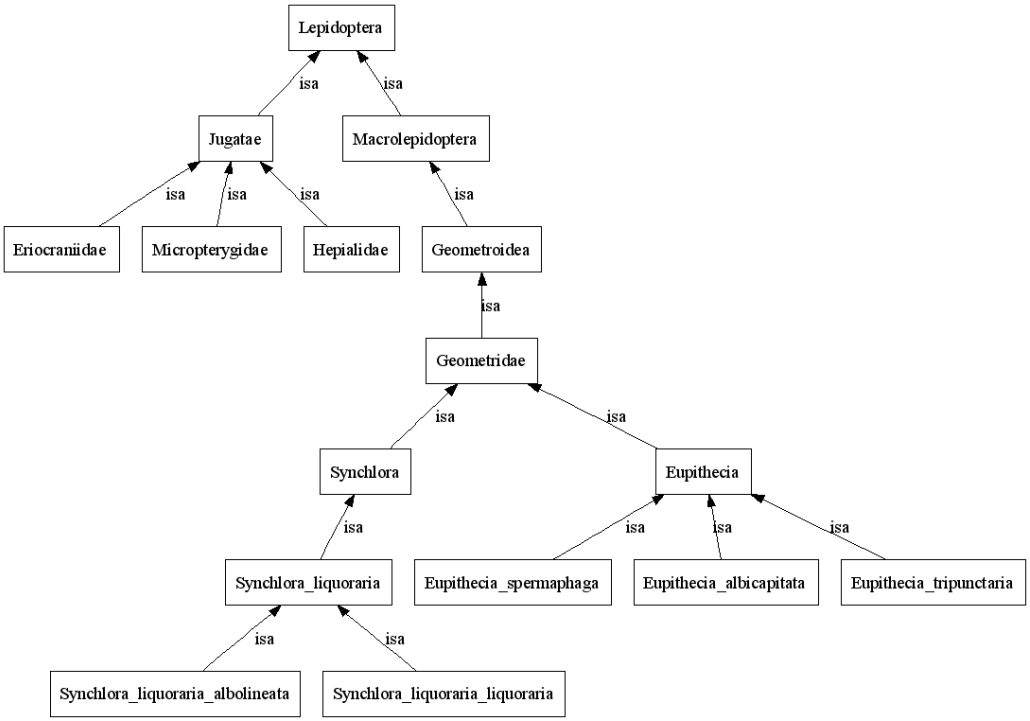

Figure 14. Part of ontology on *Lepidoptera* order retrieved from Spire

This ontology will be stored in the repository *http://143.0106.23.89:8080/ OntRepUNICAMP*; the identifier returned is *lepdDesc*. The second step (alignment), needed to solve the scientists' request, will be treated further on, in Section 6.7.

6.6. Differences Detection

Taxonomic classifications of living beings change with time, reflecting new knowledge about species. This involves not only adding new species, or modifying their names, but also revisions that bring about structural changes (e.g., creation of new branches, or moving a species to another branch). Furthermore, distinct research groups may prefer different classifications. Since our biology partners have been collecting data for 20 years, they must constantly check the terms they used in the past. Consider, thus, that they want to check their ontology *colUN* against published data on the *Asteraceae* plant family.

This can be performed via three successive invocations to Aondê, as follows: (1) perform a *SearchTaxon* on Spire, to retrieve published taxonomic data on taxon *Asteraceae* and its *descendants* – the result, partially illustrated in Figure 15 (a), is stored in a Semantic Repository, receiving identifier *astDesc*; (2) to speed up comparison, restrict the information contained in ontology *colUN* to data on *Asteraceae*, by creating a view with this term as central concept (analogous to view creation in Section 6.4) – the result, partially illustrated in Figure 15 (b), will be materialized in a new ontology, identified by *colUN-Asteraceae*; (3) compute the difference between *colUN-Asteraceae* and *astDesc*.

Step (3) is specified as:

Difference(<astDesc,http://143.0106.23.89:8080/OntRepUNICAMP>,
<colUN-Asteraceae,http://143.0106.23.89:8080/OntRepUNICAMP>,
0.6, 0.4, 0.7)

The first two parameters indicate the ontologies to be compared; parameters 0.6 and 0.4 respectively represent weights α and β of formula 1 – see Section 4.2.6; and *0.7* is the confidence threshold above which element similarities identified by the module are considered to be modifications between ontologies.

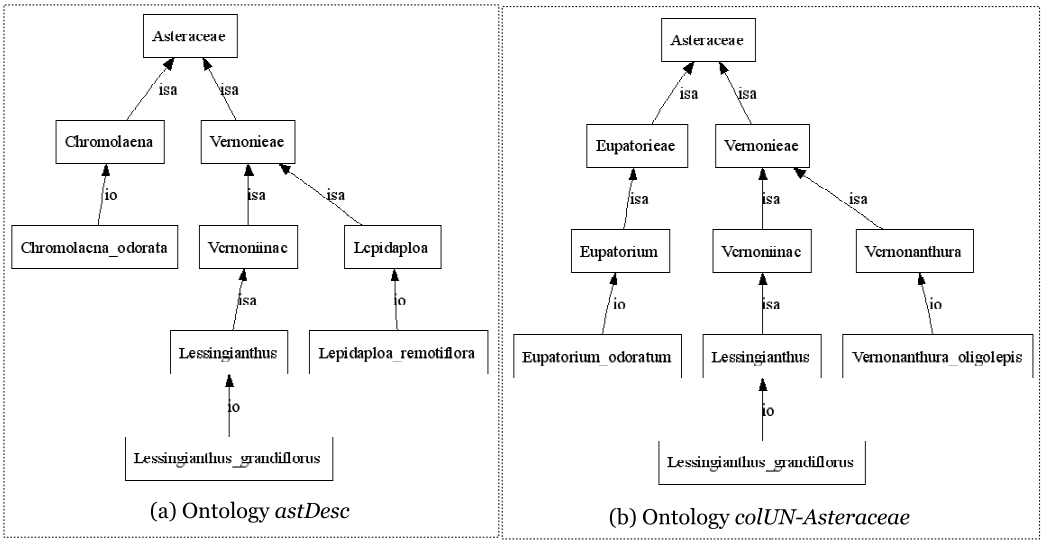

Figure 15. Taxonomies for the Asteraceae family - (a) was retrieved from Spire, (b) view created from *colUN* using a View operation

The result of this invocation is composed of three lists of URIs: *ListModification*, *ListA-B* and *ListB-A*, respectively containing URIs of elements that were identified as similar, and those that appear in A but not in B, and vice-versa. *ListModification*, moreover, specifies the kinds of difference detected between similar elements. Figure 16 shows part of the result.

The figure shows that taxons *Eupatorium* (in *colUN-Asteraceae*) and *Chromolaena* (in *astDesc*) in fact correspond to the same genus within *Asteraceae* – not only were these terms identified as synonyms by ITIS, but they also share a common superclass. Hence, they are listed within *ListModification*; their difference lies in their class labels and hierarchy. Species *Eupatorium odoratum* and *Chromolaena odorata*, also similar, differ both in class labels and hierarchy. Taxons *Eupatorieae*, *Vernonanthura* and *Vernonanthura oligolepis* appear in *colUN-Asteraceae*, but not in *astDesc*, being thus characterized as members of *ListB-A*, while taxons *Lepidaploa* and *Lepidaploa remotiflora* are absent in the former and present in the latter, hence belonging to *ListA-B*.

The result shows that the plant species *Eupatorium odoratum*, of our running example, has a new official scientific name – *Chromolaena odorata*. Thus, if our scientists want to integrate their data with information from other sources, they need to take this into consideration.

```
<?xml version="1.0" encoding="UTF-8"?>
<Difference>
   <ListModification>
      <Modification>
         <ElemInOntA>http://spire.umbc.edu/ontologies/EthanAnimals.owl#
                     Chromolaena</ElemInOntA>
         <ElemInOntB>http://www.owl-ontologies.com/Asteraceae.owl#
                     Eupatorium</ElemInOntB>
         <Type>class labels</Type> </Modification>
       <Modification>
         <ElemInOntA>http://spire.umbc.edu/ontologies/EthanAnimals.owl#
                     Chromolaena odorata</ElemInOntA>
         <ElemInOntB>http://www.owl-ontologies.com/Asteraceae.owl#
                     Eupatorium odoratum</ElemInOntB>
         <Type>class labels and class hierarchy</Type> </Modification>
   </ListModification>
   <ListA-B>
      <A-B>
         <ElemInOntA>http://spire.umbc.edu/ontologies/EthanAnimals.owl#
                     Lepidaploa</ElemInOntA> </A-B>
      <A-B>
         <ElemInOntA>http://spire.umbc.edu/ontologies/EthanAnimals.owl#
                     Lepidaploa remotiflora</ElemInOntA> </A-B>
   </ListA-B>
   <ListB-A>
      <B-A>
         <ElemInOntB>http://www.owl-ontologies.com/Asteraceae.owl#
                     Eupathorieae</ElemInOntB> </B-A>
      <B-A>
         <ElemInOntB>http://www.owl-ontologies.com/Asteraceae.owl#
                     Vernonanthura</ElemInOntB> </B-A>
      <B-A>
         <ElemInOntB>http://www.owl-ontologies.com/Asteraceae.owl#
                     Vernonanthura oligolepis</ElemInOntB> </B-A>
    </ListB-A>
</Difference>
```

Figure 16. Part of the result of the request to identify differences between the ontologies from Spire and from the project, for the*A*steracea family

6.7. Ontology Integration

Let us now return to the question of Section 6.5: *"Retrieve butterfly species that prey on plant species Eupatorium odoratum"*. Scientists are now aware that *Eupatorium odoratum* has a new name *Chromolaena odorata*, which must be considered in processing the request.

This requires the following invocation sequence: (1) search for *Lepidoptera* information to identify butterflies (this was already performed in Section 6.5, and its result stored in *lepdDesc*); (2) align *colUN* with *lepdDesc*, to identify insects that are butterflies, producing *colUN-Lep*; (3) align *colUN-Lep* with *astDesc*, which was produced by another taxonomic search; (4) query the ontology resulting from step (3).

The third step will allow putting together butterfly species recorded in $colUn$ that prey on both *Eupatorium odoratum* and *Chromolaena odorata*. The alignment requests of steps (2) and (3) are specified as follows:

Integration(<colUN,http://143.0106.23.89:8080/OntRepUNICAMP>,
<lepdDesc,http://143.0106.23.89:8080/OntRepUNICAMP>,
0.8, 0.2, 0.7, http://143.0106.23.89:8080/OntRepUNICAMP)

Integration(<colUN-Lep,http://143.0106.23.89:8080/OntRepUNICAMP>,
<astDesc,http://143.0106.23.89:8080/OntRepUNICAMP>,
0.8, 0.2, 0.7, http://143.0106.23.89:8080/OntRepUNICAMP)

The first two parameters of each request provide the source ontologies; all results will be stored in the same repository (last parameter). Parameter *0.7* represents the confidence threshold for an alignment to be included; *0.8* and *0.2* respectively stand for the values for α and β to compute similarity confidence – see Section 4.2.5.

The final aligned ontology resulting from the second Integration request is partially displayed in Figure 17 – its leftmost branch originates from *lepdDesc* (see Figure 14), the central part from *colUN* (see Figure 10), and the rightmost part from *astDesc* (see Figure 15(a). Class alignments are represented by tag <*owl:equivalentClass*>, and instance alignment by <*owl:sameAs*>.

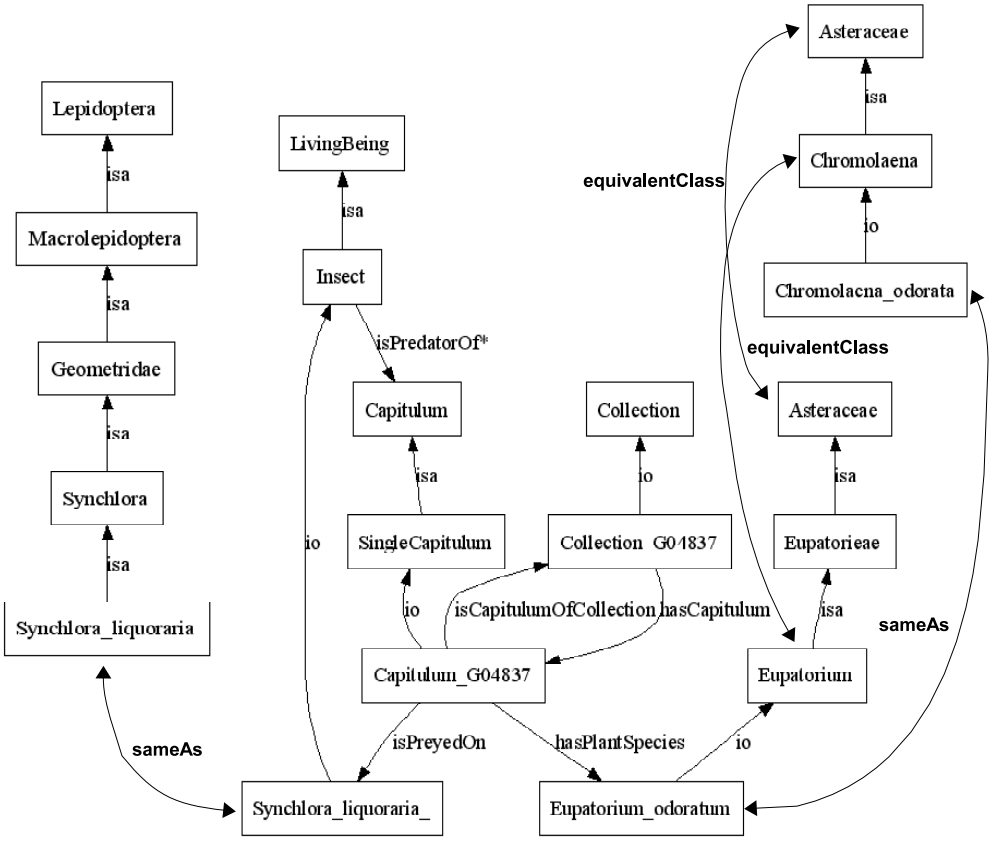

Figure 17. Part of *colLep* ontology, produced by invoking the Integration module

The first integration (between *colUN* and *lepdDesc*) may seem straightforward at first glance (i.e., akin to a "natural join between two ontologies"). However, it merits several remarks. First, it provides alignments between classes of one ontology with instances of another, correlating species instances of *Insect* (in *colUN*) with species subclasses of *Lepidoptera* (in *lepDesc*). This is a new kind of alignment, not available in existing tools, which are limited to trying to match elements of the same kind (e.g., class–class, instance–instance). We, instead, support other kinds of matching, as described in Section 5.3, using <*owl:sameAs*> links. Another observation is that finding similar terms required knowledge of taxonomic terminology, which was included in our string match-

ing implementation. Without these kinds of comparisons, the resulting aligned ontology would not identify these correspondences.

The result of step (3) (second integration, between *colUN-lep* and *astDesc*) could only be obtained through other features necessary in biodiversity applications. First, we take advantage of synonyms in taxonomic classifications to discover matchings. Second, we combine string similarity with structural similarity to compute alignment confidence. The latter is the case of classes *Chromolaena* and *Eupatorium*: they had one instance aligned by synonym identification (*Chromolaena odorata* and *Eupatorium odoratum*) and their superclasses (*Asteraceae*) were identified as equivalent. String and structural similarity computation were also used in other cases (e.g., difference, alignment of step (2)), but we chose this example to show that both are needed to find appropriate correspondences.

The result of the two successive alignments can now be used to obtain the required information: *"Butterfly species that prey on plant species Eupatorium odoratum"*, using an invocation to a $Query$ operation. However, unlike previous queries, this one requires inference pre-processing. It must return insect species that are in all subclasses of *Lepidoptera*, and that prey on flowerheads of plants species that are equivalent to species *Eupatorium odoratum*. Inference must be used to compute subclasses of *Lepidoptera* (transitivity of property *subClassOf*). It must also be used to retrieve equivalent (aligned) elements – e.g., taking advantage of the symmetry of properties *equivalentClass* and *sameAs*. This query returned 6 butterfly species, among which *Synchlora liquoraria* and *Adaina bipunctata*.

6.8. Ranked Search

Suppose that, now, the biologists want to obtain additional information about the *plant* concept, such as plant structure and their interactions with others biotic entities. This information is not available in any ontology mentioned so far. This task requires a search for the "plant" concept in third party External Ontology Repositories. Here, since *plant* is not a domain-specific term, the search request was sent to the Swoogle[7] [17] repository:

SearchRank ({*plant*}, {*0.4,0.2,0.4,0*},{*Swoogle*}, *http://143.0106.23.89:8080/OntRepUNICAMP)*

Like the rest of the examples, the resulting ontologies are stored in the Semantic Repository named in the Search invocation, to be subsequently reused in other operations. The field {*plant*} indicates the term to search for in *Swoogle*. The set of positional weights are assigned as follows: match = 0.4; density = 0.2; centrality = 0.4 and similarity semantics = 0. This invocation searches for a single term, so the *similarity metric* is not applicable and receives weight = 0. We recall that the sum of weights must be 1. In this example, we wanted to assign more importance to centrality and match metrics, and less importance to density metrics. Weights *0.4*, *0.4* and *0.2* indicate this decision. This specific invocation returns a list with 10 identifiers (i.e., Swoogle has 10 ontologies with *plant*) ordered according to ontology rank, and store all the ten ontologies in the designated target repository.

The Figure 18 shows the ontology with the highest ranking value, available in the URL *http://wow.sfsu.edu/ontology/rich/EcologicalConcepts.owl*. The figure shows that

[7] http://swoogle.umbc.edu

the *plant* term appears in several class names, outlined in the graph – *Plant, PlantDescription, PlantPartDescriptiveTrait, PlantParts, PlantSpecies, AboveGroundPlantParts* and *PlantTrait*. This high incidence of this term (the highest of all ontologies returned by the search) put this ontology as the first in the rank, using the metrics of *Partial and Exact Matchings*. Moreover, this ontology contains many other concepts related to these classes (centrality criterion) – such as classes *Organism, BioticItem,Leaves, Seeds* and *Stems*, also contributing to its rank.

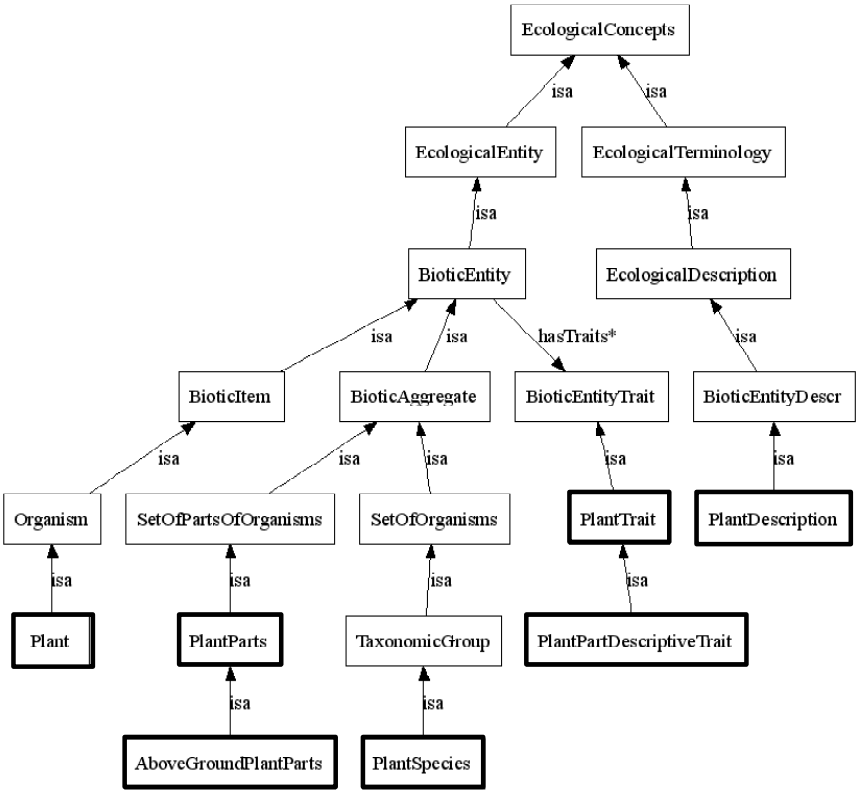

Figure 18. Ontology *ecoConcept*, highest ranked in the search for *plant* in Swoogle

7. Conclusions

This paper discussed the specification and the implementation of Aondê – a Web service that provides distinct kinds of operations on multiple ontologies at the same time, on the Web. Aondê supports semantic and interoperability needs by combining the use of ontologies to that of Web services. It was specified and built to support the requirements of biodiversity researchers, who need to have transparent access to distributed and heterogeneous data sources. One of the important issues in this context is that biodiversity studies have to accommodate and combine multiple views and needs of a wide range of experts whose research involves deriving new relationships among species, for specific geographic regions.

Unlike all other Web ontology services available, Aondê supports management of many ontologies at a time (in ranking, difference and integration). The design and implementation of its functions combine and extend facilities offered by other tools – e.g., enhancing central concept computation in view extraction. Though motivated and

developed by biodiversity concerns, Aondê can be used in other domains that present the same requirements, typically those in e-Science.

The design and development of Aondê showed us there are still many challenges to be met when dealing with ontologies, some of which are discussed in the subsequent paragraphs. One important lesson is that there is a wide gap between several proposals for ontology management and their implementation. Many published experiments are based on small examples, and do not show solutions to real life problems – e.g., handling only standard textbook examples such as those involving teachers and students, or papers and authors. Thus, we could not adopt many solutions because they could not deal with our problems – e.g., having to handle multiple valid ontologies.

Another lesson concerns the usefulness of applying design principles of Software Engineering and Database Systems to our design and implementation efforts. One example of such a decision was the clear separation between persistence of ontologies and their semantic manipulation. This lent flexibility to our implementation, and will help maintain independence between these two layers, which can evolve separately. This kind of concern is not found in most proposals dealing with ontology management.

Though W3C recommends the adoption of OWL, and is leaning towards the adoption of SPARQL as a standard, most large ontologies (e.g., TAMBIS [10]) have not been written in OWL. Thus, in spite of standardization efforts, there is a need for translating these ontologies into OWL, or to use tools that handle ontologies written in multiple languages – e.g., as in [58]. This is not an easy task, because of the differences in expressive power among ontology specification languages, and remains an open issue.

For many authors, ontologies only serve as sophisticated dictionaries to look for synonyms of terms (e.g., they only consider is-a relationships among terms). This subutilization of such a powerful resource is largely due to the limitations of available ontology toolkits and environments. Even when ontologies are completely defined using description logics, the absence of tools that allow adequate exploration of these logics forces researchers to limit themselves to more prosaic uses.

Web services are advertised – and recognized – to be a good solution to interoperability. Nevertheless, the actual options for design and implementation of new services are not discussed in the literature. For instance, our persistence service publishes ontologies within files that are attached to SOAP messages. Other options exist, such as embedding the ontology in the message, or just returning its URI. Each solution has its pros and cons. For instance, embedding the ontology in the message would require treating it as an object, with many methods associated, thus creating very long messages. On the other hand, several Web service tools (including the one we used, Axis 1.4) do not work well when attachments are too large.

The adoption of Web services to access the persistence layer provides independence and interoperability, at the cost of lower security. Thus, another research issue concerns authorization mechanisms, to avoid undesirable updates to curated ontologies. This has an impact on ontology reliability, and is a research topic that needs to be attacked.

Finally, we point out that our decision to store intermediate ontologies in Semantic Repositories has several advantages, in spite of taking up storage space. The main issue is that such ontologies are expensive to create (especially those that are imported

from external repositories, and aligned ontologies). Thus, storing them helps speed up subsequent access to their contents.

Ongoing and future work concerns several research and implementation aspects. We are building a suite of test cases, to analyze the service's performance when many repositories are accessed at one operation. Programming aspects also include the incorporation of Aondê into WeBios.

Many research issues need to be tackled. First, integration is based on alignment. This must be extended to, at least, ontology merging, which poses several problems involving merge priorities and result computation. Second, one of the central ideas is that any new (sub-) ontology must be first stored in some Semantic Repository, and only then be submitted to other Aondê operations. Though this may speed up retrieval, it brings about issues of storage allocation and ontology replication. Thus, alternative design options have to be considered – e.g., creating some kind of repository directory. A third direction is to endow Aondê with more intelligence, so that it can support more complex requests by combining elementary operations, e.g., by constructing composite service invocations. Still another issue to investigate is alternatives to alignment implementation. Our solution is to relate classes of one ontology with instances of another via <*owl:sameAs*> tags. This immediately "promotes" the ontology language to OWL Full, which still needs better inference mechanisms.

Finally, another ongoing effort concerns finishing the design and implementation of the ontology usage data space within Semantic Repositories. This data space will provide additional performance and usage functionality to the Semantic Repository Service, similar to auxiliary low-level DBMS structures that help performance tuning. A first version of this space and associated operations has been implemented, but we need to define appropriate tuning and usage parameters before we can validate its design.

Acknowledgments

This research was partially financed by Microsoft Research, the initial supporter of the WeBios project, and by Brazilian funding agencies CNPq, CAPES and FAPESP (Proc. 05/57424-0). We also thank the reviewers for their insightful comments.

References

[1] S. Abels, L. Haak, and A. Hahn. Identification of common methods used for ontology integration tasks. In *IHIS '05: First International Workshop on Interoperability of Heterogeneous Information Systems*, pages 75–78. ACM Press, 2005.

[2] S. Abiteboul, R. Goldman, J. McHugh, V. Vassalos, and Y. Zhuge. Views for Semistructured Data. In *Proc. International Workshop on Management of Semistructured Data*, 1997.

[3] H. Alani, C. Brewster, and N. Shadbolt. Ranking Ontologies with AKTiveRank. In *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2006.

[4] H. Alani, S. Harris, and B. O'Neill. OntologyWinnowing: A Case Study on the AKT Reference Ontology. In *CIMCA/IAWTIC*, pages 710–715. IEEE Computer Society, 2005.

[5] A. M. Almeida, C. R. Fonseca, P. I. Prado, M. Almeida-Neto, S. Diniz, U. Kubota, M. R. Braun, R. L. G. Raimundo, L. A. Anjos, T. G. Mendonça, S. M. Futada, and T. M. Lewinsohn. Diversidade e ocorrência de Asteraceae em cerrados de São Paulo. *Biota Neotropica*, 5:27 – 43, 2005.

[6] A. M. Almeida, C. R. Fonseca, P. I. Prado, M. Almeida-Neto, S. Diniz, U. Kubota, M. R. Braun, R. L. G. Raimundo, L.A. Anjos, T. G. Mendonça, S. M. Futada, and Thomas M. T. M. Lewinsohn. Assemblages of endophagous insects on Asteraceae in São Paulo Cerrados. *Neotropical Entomology*, 35:458 – 468, August 2006.

[7] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services - Concepts, Architectures and Applications*. Springer Verlag, November 2004.

[8] G. Antoniou and F. van Harmelen. Web Ontology Language: OWL. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*, pages 76–92, 2003.

[9] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nature Genetics*, 25(1):25–29, May 2000.

[10] P. G. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, and R. Stevens. TAMBIS–Transparent Access to Multiple Bioinformatics Information Sources. In *Int Conf Intelligent Systems for Molecular Biology*, volume 6, pages 25–34, Montreal, Canada, June 1998.

[11] C. Batini, M. Lenzerini, and S. B. A. Navathe. Comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[12] D. Beckett and J. Broekstra. SPARQL Query Results XML Format. Technical report, W3C, April 2006.

[13] J. Bruijn, F. Martin-Recuerda, D. Manov, and M. Ehrig. State-of-the-art survey on Ontology Merging and Aligning. Technical report, SEKT project D4.2.1, 2004.

[14] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: implementing the semantic web recommendations. In *WWW Alt. '04: Proc. of the 13th international World Wide Web*, pages 74–83. ACM Press, 2004.

[15] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proc. of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, pages 73–78, August 2003.

[16] Z. Cui and P. O'Brien. Domain Ontology Management Environment. In *HICSS '00: Proc. of the 33rd Hawaii International Conference on System Sciences-Volume 8*, Washington, DC, USA, 2000. IEEE Computer Society.

[17] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a Search and Metadata Engine for the Semantic Web. In *CIKM '04: Proc. of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659, New York, NY, USA, 2004. ACM Press.

[18] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to Map between Ontologies on the Semantic Web. In *WWW '02: Proc. of the 11th international conference on World Wide Web*, pages 662–673. ACM Press, 2002.

[19] M. Duke and M. Patel. An Ontology Server for Agentcities.NET. Technical report, September 2003.

[20] C. H. Felicíssimo. Semantic Web Interoperability: One strategy for the Taxonomic Ontology Alignment (in Portuguese). Master's thesis, Pontifical Catholic University of Rio de Janeiro (PUC–FRJ, August 2004.

[21] C. R. Fonseca, P.I. Prado, M. Almeida-Neto, U. Kubota, and T. M. Lewinsohn. Flowerheads, herbivores, and their parasitoids: food web structure along a fertility gradient. *Ecological Entomology*, 30:36–46, February 2005.

[22] R. B. Freitas and R. S. Torres. Ontosaia: An ontology-based tool for image retrieval and semi-automatic annotation (in portuguese). In *I Workshop in Digital Libraries, Proc. XX Brazilian Symposium on Databases - SBBD 2005*, pages 60–79, October 2005.

[23] Global Biodiversity Information Facility (GBIF). GBIF website. http:// www.gbif.org (accessed February 26, 2007).

[24] J. H. Gennari, M. A. Musen, R. Fergerson, W. E. Grosso, M. Crubzy, H. Eriksson, N. F. Noy, and S. W. Tu. The Evolution of Protege: An Environment for Knowledge-Based Systems Development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[25] L. C. Gomes Jr and C. B. Medeiros. Ecologically-aware Queries for Biodiversity Research. In *Proceedings GeoInfo - Brazilian Geoinformatics Symposium*. INPE - SBC, 2007. Electronic proceedings, 12 pages.

[26] T. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human-Computer Studies*, 43(5-6):907–928, 1995.

[27] P. Hammond, B. Aguirre-Hudson, M. Dadd, B. Groombridge, J. Hodges, M. Jenkins, M.H. Mengesha, and W. Stewart Grant. The current magnitude of biodiversity. Global biodiversity assessment, 1995.

[28] J. Hartmann, Y. Sure, P. Haase, R. Palma, and M. C. Suárez-Figueroa. OMV – Ontology Metadata Vocabulary. In *ISWC 2005 - In Ontology Patterns for the Semantic Web*, November 2005.

[29] J. S. Hong, H.Y. Chen, and J. Hsiang. A digital museum of taiwanese butterflies. In *ACM Digital Library*, pages 260–261, 2000.

[30] E. Jiménez, R. Berlanga, I. Sanz, M. J. Aramburu, and R. Danger. OntoPathView: A Simple View Definition Language for the Collaborative Development of Ontologies. *In B. López et al. (Eds.): Artificial Intelligence Research and Development*, pages 429–436, 2005.

[31] L. C. Gomes Jr. An architecture to query biodiversity data on the Web (in portuguese). Master's thesis, State University of Campinas - UNICAMP, May 2007.

[32] Y. Kalfoglou and M. Schorlemmer. Ontology Mapping: the State of the Art. *Knowledge Engineering Review*, 18(1):1–31, 2003.

[33] H. Kong, M. Hwang, and P. Kim. A New Methodology for Merging the Heterogeneous Domain Ontologies Based on the WordNet. In *NWESP '05: Proc. of the International Conference on Next Generation Web Services Practices*, page 235, Washington, DC, USA, 2005. IEEE Computer Society.

[34] J. Lee. An Application Programming Interface for Ontology. Technical report, November 2003.

[35] Y. Li, S. G. Thompson, Z. Tan, N. Giles, and H. Gharib. Beyond Ontology Construction; Ontology Services as Online Knowledge Sharing Communities. In *International Semantic Web Conference - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 469–483. Springer, 2003.

[36] F. Manola and E. Miller. Resource Description Framework (RDF) Model and Syntax Specification, February 2004. http://www.w3.org/TR/rdf-primer/.

[37] R. McDiarmid. The Integrated Taxonomic Information System. In *Proc. of the Taxonomic Authority Files Workshop*, June 1998.

[38] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. The Chimaera Ontology Environment. In *Proc. of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, pages 1123–1124, 2000.

[39] N. F. Noy, S. Kunnatur, M. Klein, and M. A. Musen. Tracking Changes During Ontology Evolution. In Sheila A. Mcilraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Third International Semantic Web Conference*, pages 259–273. Springer Berlin, November 2004.

[40] N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Seventeenth International Joint Conference on Artificial IntelligenceAAAI/IAAI*, pages 450–455, 2000.

[41] N. F. Noy and M. A. Musen. Specifying Ontology Views by Traversal. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 713–725, 2004.

[42] C. S. Parr, A. Parafiynyk, J. Sachs, L. Ding, S. Dornbush, T. W. Finin, D. Wang, and A. Hollander. Integrating Ecoinformatics Resources on the Semantic Web. In *Proc. in 15th International Conference on World Wide Web*, pages 1073–1074. ACM, 2006.

[43] C. Patel, K. Supekar, Y. Lee, and E. K. Park. OntoKhoj: a Semantic Web Portal for Ontology Searching, Ranking and Classification. In *WIDM '03: Proc. of the 5th ACM international workshop on Web information and data management*, pages 58–61, New York, NY, USA, 2003. ACM Press.

[44] A. G. Perez, J. Angele, M. F. Lopez, V. Christophides, A. Stutt, and Y. Sure. A survey on ontology tools. Deliverable 1.3, EU IST Project IST-2000-29243 OntoWeb, 2002.

[45] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report, World Wide Web Consortium - W3C, 2006.

[46] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001.

[47] J. A. Ramos. Mezcla automática de ontologías y catálogos electronicos. Technical report, 2001.

[48] A. Seaborne. RDQL: A Query Language for RDF. Technical report, World Wide Web Consortium - W3C, 2003.

[49] P. Shvaiko and J. Euzenat. A Survey of Schema-based Matching Approaches. Technical report, 2004.

[50] Sinbiota. São Paulo Biodiversity System site. http://sinbiota.cria.org.br/ (accessed May 10, 2007).

[51] J. A. Sánchez, C. A. Flores, and J. L. Schnase. Mutant: Agents as guides for multiple taxonomies in the floristic digital library. In *ACM Digital Library*, pages 244–245, 1999.

[52] H. Suguri, E. Kodama, M. Miyazaki, H. Nunokawa, and S. Noguchi. Implementation of FIPA ontology service. In *Proc. of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents*, May 2001.

[53] R. S. Torres, C. B. Medeiros, M. A. Gonçalves, and E. A. Fox. A Digital Library Framework for Biodiversity Information Systems. *International Journal on Digital Libraries*, 6(1):3 – 17, February 2006.

[54] M. Tury and M. Bieliková. An Approach to Detection Ontology Changes. In *ICWE '06: Workshop proceedings of the sixth international conference on Web engineering*, page 14, New York, NY, USA, 2006. ACM Press.

[55] R. Volz. ONTOSERVER - Infrastructure for the Semantic Web (Position Paper). In *Proc. of Semantic Web Working Symposium - SWWS*, Stanford, California, USA, 2001.

[56] R. Volz, D. Oberle, and R. Studer. Implementing Views for Light-Weight Web Ontologies. In *Proc. of Int. Database Engineering and Application Symposium - IDEAS*, Hong Kong, China, 07 2003.

[57] WeBios. WeBios - Web Service Multimodal Tools for Strategic Biodiversity Research, Assessment and Monitoring. Home page http://www.lis.ic.unicamp.br/projects/webios, 2007.

[58] P. Ziegler, C. Kiefer, C. Sturm, K. R. Dittrich, and A. Bernstein. Detecting Similarities in Ontologies with the SOQA-SimPack Toolkit. In *10th International Conference on Extending Database Technology (EDBT 2006)*, volume 3896 of *Lecture Notes in Computer Science*, pages 59–76. Springer, 2006.