COEUS: AN APPLICATION FRAMEWORK FOR ENHANCED SERVICE COMPOSITION

The Semantic Web has provided bioinformatics developers with better paradigms, standards and technologies to solve common problems such as data heterogeneity, diversity or distribution. The Bio2RDF warehouse or the Biocatalogue library, amongst other systems, have shown how valuable semantic web technologies can be for the general life sciences software field. However, semantic web's potential is still out of reach of the bioinformatics developers' community. There is a clear absence of tools to enhance the migration of existing platforms to new environments, to ease the development of information systems from scratch or to disrupt with past strategies by deploying fully interoperable software.

Hence the introduction of COEUS, a framework to tackle these challenges by empowering developers with a "Semantic Web in a box" software stack, and ensuing a more agile development workflow for new semantic web systems. The COEUS open-source project is available at http://bioinformatics.ua.pt/coeus/.

This chapter discusses the devised strategies and their implementation, leading to COEUS development. Starting with the demand for more modular and dynamic software packages in the life sciences, we move on to a brief assessment of semantic web use in bioinformatics, highlighting the opportunities for a new kind of application development strategy that can empower the next generation of bioinformatics software.

Dynamic Software Infrastructures for Life Sciences

Reusable Assets

The cornerstone of current software development is the idea of "reusing instead of rewriting". This rather basic proposition is applied not only to the construction of data models, where defining new schemas or entire structures is a complex research practice, but also to the set of programming toolkits being used. Regarding the latter, the vast number of applications, libraries, services or packages, makes it very easy for developers to find a solution to an implementation problem. Even when they end up implementing the desired feature from scratch, they do so acknowledging the already existing algorithms, their limitations and their strengths.

Common modeling, service access, knowledge acquisition or data exploitation problems have been solved before. Associated with the facility in finding existing solutions, developers are now endowed with tools to quickly integrate miscellaneous libraries in their projects, such as Maven[1], Node.JS NPM[2] or Ruby's Gems[3]. Hence, the software development process is streamlined to a three-stage identify-assess-reuse practice.

As stated in section 3.1, this is leveraging the use of rapid application development frameworks. Likewise, the bioinformatics field is also becoming aware of these quicker application deployment strategies, and new toolkits are starting to emerge.

Evaluating Rapid Application Development in Bioinformatics

Rapid Application Development (RAD) is a strategy for generating entire application, including databases, code and services, from a set of configuration files. This permits launching new tools much faster than otherwise, thus reducing the "time-to-market" cost. When assessing RAD strategies, the major conclusion is the traditional poor component availability. Available frameworks either generate one or two good components (going feature-deep in each one) or generate multiple components with basic functionality (going for a wider coverage breadth). Nevertheless, these frameworks permit creating complete application stubs in no time, making them suitable for initial prototypes or low-end solutions.

[1] http://maven.apache.org/

[2] http://npmjs.org/

[3] http://rubygems.org/

The Molgenis framework is a “generic, open source, software toolkit” to quickly produce bioinformatics user-friendly and scalable software. This toolkit provides developers with a simple modelling language to design data structures and user interfaces. From two valid configuration files, Molgenis’ generator creates a "feature-rich, ready-to-use web application including database, user interfaces, exchange formats, and scriptable interfaces".

The automatic code generation tools are one of Molgenis’ highlights. They reduce the amount of code one has to write by hand. The template-based nature of available methods leverages a straightforward generation process, easing the transformation from the configuration file to SQL, Java, R or HTML code.

Molgenis’ use has been growing over the last few years. Biomedical applications for miscellaneous areas, including genome-wide association studies, proteomics, biobanking or next-generation sequencing, have already been launched using this toolkit. Bioinformaticians usually seek Molgenis’ great adaptability. This allows them to generate entire application structures much faster when the resulting skeleton can be optimized or to iteratively generate solutions until the final system is ready.

ProteoWizard, BioJava and AIBench are some of Molgenis competitors. ProteoWizard is a C++ framework very similar to Molgenis in the sense that it provides a comprehensive set of features to speed up the development of applications requiring some kind of proteomics data manipulation. As the name states, BioJava is a set of libraries that can be used in any Java application project and that reduce the complexity of dealing with biological data. This widely used package facilitates reading and parsing data, performing simple statistical and analytical tasks and accessing common bioinformatics features such as sequence alignment or protein structure exploration. At last, AIBench was initially built as a rapid application development framework for data mining, but its use is being extended to biomedicine. Also in Java, AIBench enables code annotations, scripting and custom plugins to be included in a predefined application skeleton, reducing the huge effort of implementing desktop application interfaces from scratch.

Combining rapid service development with semantic technologies is SADI’s framework goal, which promotes guidelines and ontologies to exploit the composition of semantic web services, through a straightforward strategy. Input and output interfaces accept and expose data in RDF format: service data are OWL class instances. Inward data are enriched with new relationships until they match the desired output, they are then sent as the service reply, streamlining the web service dataflow. SADI includes patterns for describing the service interfaces and enables the creation of client applications with “strikingly rich semantic behaviours”. Henceforth, it is clear that rapid application development strategies must be mixed with the semantic web paradigm to deploy richer bioinformatics application frameworks.

Towards a Semantics-enabled Architecture

Research from Slater *et al.* and Kozhenkov *et al.* , among others, analyses current software development strategies, concluding that there is a clear need for new approaches adopting distinct ideals and based on a different set of skills. Like next-generation sequencing hardware improves genetic reads in a multitude of ways, the Semantic Web may be seen as a next-generation software development paradigm, capable of breeding a new wave of biomedical software solutions. However, despite its growing momentum, semantic web strategies are still subject of a slow adoption process.

Taking in account the need for novel bioinformatics software with improved integration and interoperability features , the use of semantic web technologies to tackle innate life sciences challenges will permit that entirely different computational systems exchange and accurately interpret knowledge. With an ever-increasing amount of data, produced in both novel software and hardware platforms, and a prolific research community constantly demanding best-of-breed tools, this field is evolving exponentially and reaching user types far beyond the traditional wet-lab biologist. Semantic knowledge discovery, reasoning and inference are now a part of state-of-the-art research.

Despite the key role that bioinformatics software and hardware developments have played over the last three decades, the life sciences technological ecosystem is still fragmented and characterized by immeasurable entropy. The majority of data are scattered through closed independent systems, disregarding any good-practice for integration and interoperability features. Furthermore, even in notable state-of-the-art tools, the overwhelming scale and complexity of collected data and features generates an information overload, making it impossible for researchers to grasp any deep insights from available knowledge.

Interoperable bioscience data are essential to keep up with the bioinformatics evolution momentum and extract the added value from the vast swathes of digital life sciences data. This demands new strategies for getting the data out of primitive systems, using independent formats and non-standard terminologies, into a state-of-the-art open knowledge federation environment.

Furthermore, reusable data and reusable components are key for reproducible research and easily accessible knowledge. Making new systems part of a bigger network, such as the Linked Open Data cloud, will ultimately result in better access to data, promoting research collaboration and further increasing community buy-in.

To overcome these challenges we envisaged a new application development paradigm that boosts the integration of distributed and heterogeneous data and promotes interoperability through multiple application programming interfaces. Tying all this with semantic web developments results in a powerful methodology for improving existing biomedical software and streamlines the deployment workflow.

An Architecture for Knowledge Federation

Combining biomedical software engineering with semantic web ideals, we can pinpoint two broad and distinct approaches for enriching existing datasets with integrated connections amongst resources. On the one hand, there are strategies based on data warehousing techniques, centralizing content from heterogeneous resources. On the other hand, there are solutions involving integrated access to distributed data sources, federating available content through a middleware layer. Both approaches are shown in Figure 0-1.

In opposition to warehousing, federation strategies acknowledge the distinct setup of each specialized instance. The integration of distributed resources requires some kind of middleware, a federation layer, to connect data available in each federated instance. Once this layer is deployed, data access becomes transparent. Even though performance may be poorer than in warehousing solutions, federation strategies can easily scale to accommodate distinct ontologies, regular data updates in each independent node and long-term improvements. Federation is hidden from end-users as they can access data in the same way as with warehousing repositories. Moreover, the federation layer handles query distribution and deals directly with each repository native API.

Furthermore, federation is innate to Semantic Web technologies and fits well within the biomedical applications domain. The SPARQL specification was designed from scratch to ease this process. Publishing data through SPARQL endpoint enables access to data in more advanced ways than traditional SQL. Not only does this permit development of general federation tools, but it also promotes the creation of more complex software frameworks, sustained by native Semantic Web federation.

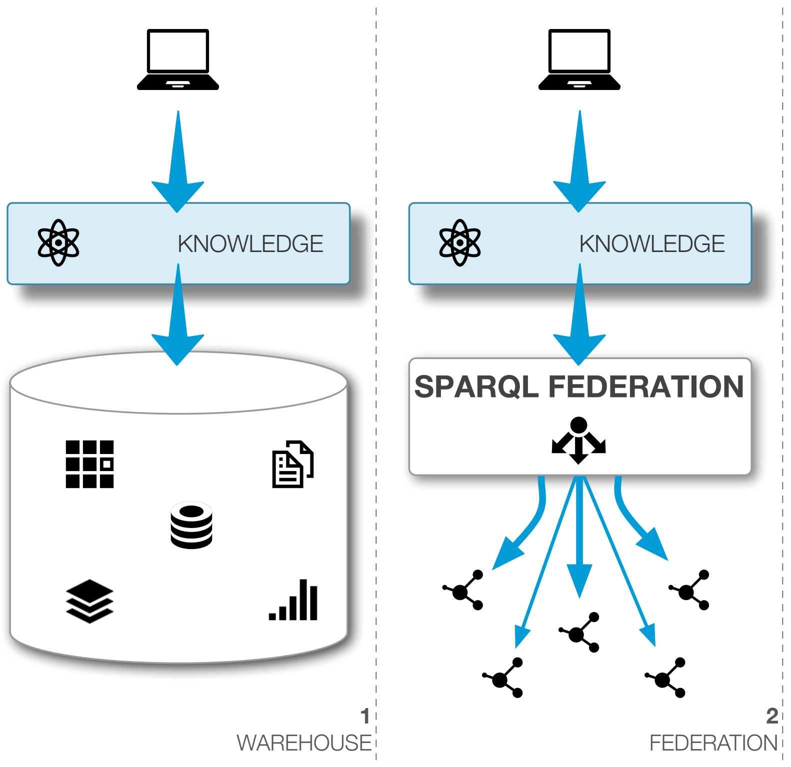


Figure 0-1. 1) Warehouse integration strategy, multiple resources are replicated in a central warehouse for prompt access to knowledge. 2) Federation strategy based on SPARQL endpoints providing direct access to each resource.

Semantic Web State of the Art in Bioinformatics

The semantic web migration process, moving systems from flat-file or relational environments to semantic infrastructures, has been the subject of extensive research. The major emphasis is given to the development of translation languages, enabling the mapping from relational connections to the semantic web graph. On the one hand, basic languages map tables and columns to a new model following a proposed ontology. On the other hand, more innovative systems permit the extension of existing data connections, enriching their meaning and expressiveness. In this topic, two approaches are common. Some mappings are dedicated to forming new triple sets from existing relational databases whereas other languages enable publishing semantic views over relational data.

These languages are complemented with translation applications, using the newly mapped model to provide a semantic data version. Triplify , Virtuoso[4] and D2R server have managed to employ new techniques that allow for semantic views and provide access to

[4] http://virtuoso.openlinksw.com/

existing relational data. Instances with DBLP[5], SIDER[6], DrugBank[7], DailyMed[8] and Diseasome[9] data were created using D2R, enabling SPARQL data integration endpoints.

Despite these advances in migration technology, the resulting systems are just a semantic version of pre-existing relational data. Thus, there is a lack of data insertion and triplification features, which are challenging tasks being backed by large-scale research projects.

Bio2RDF or DBPedia collect a gigantic amount of data in outsized triplestores. With the same decision-support goals as traditional warehouse systems, these applications adopt advanced extract-transform-load techniques to triplify existing data into a semantic format, storing them in triplestores. DBPedia offers a triplified Wikipedia version, containing its entire dataset, along with multilanguage support and category organization. Bio2RDF's focused biology environment enables it to be a remarkable life sciences semantic database, collecting data pointers from a wide array of domains, from genes to proteins up to pathways and publications.

Despite these large semantic systems' quality, they are not fit for common niche fields. Whilst Bio2RDF diversity and size will expand its use to the level of systems like UniProt or BioMART, these features also make it less suitable for smaller and restricted environments such as specific gene, disease or model organism information systems. Even if new software includes connections to Bio2RDF data, the system's core will be composed of small datasets and other precise information bits gathered from external databases or wet-lab file systems.

S3DB proposes a new data management model for integrating biomedical knowledge capable of helping in miscellaneous niche environments. S3DB provides developers with tools to construct their own ad-hoc Semantic Web applications, instead of beginning the development with an empty box. The proposed solutions for managing ontologies or locked data repositories make it adequate for closed environments. Data integration features are still very primitive, though. In these areas, it is imperative to provide mechanisms for importing data in various formats into the triple store, a process essential for obtaining enhanced collections of data.

[5] http://www4.wiwiss.fu-berlin.de/dblp/
[6] http://www4.wiwiss.fu-berlin.de/sider/
[7] http://www4.wiwiss.fu-berlin.de/drugbank/
[8] http://www4.wiwiss.fu-berlin.de/dailymed/
[9] http://www4.wiwiss.fu-berlin.de/diseasome/

Opportunities for Building a Semantic Web Framework

Evolving current applications to the semantic web ecosystem is a necessary leap in upcoming years. With the currently available tools, successful migrations are limited to a below-reasonable level. Moreover, developers must take in account the needs of future software: the integration and interoperability challenge must be tackled from the start. The combination of these factors with biomedical software requirements demands a new kind of application framework, thriving under the vast potential and opportunities brought about by semantic web technologies. The reasoning for developing COEUS is summarized next, in four overarching integration and interoperability opportunities.

- As previously stated, the principles for **rapid application development** practices, already common in the general computer science field, are gaining traction within the bioinformatics community. With this methodology, the opportunity arises to promote the use of streamlined development packages, libraries and frameworks.
- The adoption of **semantic web integration** strategies, based on advanced knowledge triplification procedures, is a vital opportunity to improve existing Extract-Transform-Load tasks in data warehousing. Easing the transition process from CSV files or relational databases into semantic web triplestores is the cornerstone for publishing knowledge online.
- With data being generated at a very high throughput rate, connecting it and making it available is essential to fully explore and understand its inner wisdom. Hence, there is a clear opportunity to employ new **semantic interoperability** standards, like SPARQL or LinkedData, to enhance knowledge sharing, broadcasting, reasoning and inference.
- **Federated** data networks will play a key role in the future dissemination of knowledge from any science field. The demand of more integrative and interoperable data leverages the opportunity to build new systems where these features are standard and available by default.

With COEUS we introduce a solution that embraces these opportunities, being able to scale and adapt to future unforeseen scenarios. The COEUS framework offers flexible schema mappings for data integration and future-proof interoperability, making it the ideal candidate for improving the complex process of developing new semantic web application ecosystems.

Requirements Analysis and Design Issues

Leveraging on the aforementioned opportunities to build a new semantic web-based environment, we conducted a careful analysis of the requirements behind such system.

These requirements are generically entailed in the following guidelines: (R1) enhanced rapid application development, (R2) suitable integrative ontology, (R3) semantic data management, (R4) flexible integration, (R5) customizable web applications, (R6) interoperability with software tools, (R7) federation architectures and (R8) open-source availability. Next, a lightweight overview of these requirements is introduced:

- **(R1) Enhanced rapid application development.** The COEUS framework must bring rapid application development in bioinformatics one step further. This should be particularly evident in the adoption of semantic web technologies and in the bioinformatics-driven platform design.
  - **(R1.1) Streamlined instance configuration.** The configuration of new COEUS instances must be streamlined to require a minimal set of instructions.
  - **(R1.2) Simple instance boot.** COEUS instance creation process must be simplified and the majority of tasks automated to enable the quick launch of new applications.
  - **(R1.3) Usable in any programming environment.** The resulting framework must make data available for any client-side development environment.
- **(R2) Suitable integrative ontology.** COEUS' development must include the design of a new integration ontology, tailored to the devised integration strategies.
  - **(R2.1) Rich resource description.** The description of integrated resources must be as rich as possible to allow for a flexible integration environment.
  - **(R2.2) Ontology-based data mappings.** COEUS' ontology for resource description must enable the mapping of non-semantic data into any ontology from any field.
- **(R3) Semantic data management.** COEUS must be supported by a semantic knowledge base, thus enabling data management through a semantic layer.

- **(R3.1) Triplestore knowledge base.** COEUS' knowledge base should be supported by a semantic triplestore, whether through in-memory, file-based on relational-based strategies.
- **(R3.2) Semantic data translation.** COEUS must enable the translation of data from existing non-semantic environments into its internal semantic knowledge base.
- **(R3.3) Knowledge reasoning and inference.** As an integral part of any semantic web system, features must be available in COEUS to permit the effective reasoning over acquired knowledge and the inference of new data relationships.

→ **(R4) Flexible integration.** Resource integration in COEUS must be a flexible process to allow the integration of data from distributed and heterogeneous sources.

- **(R4.1) Data loading from SQL, CSV, XML or SPARQL sources.** Automated integration of data from CSV or XML files, or from SQL or SPARQL query results is mandatory.
- **(R4.2) Extensible integration architecture.** In addition to (R4.1), COEUS must support the creation of custom integration plugins.

→ **(R5) Customizable web applications.** COEUS must empower the eased creation of client-side web applications, through normalized infrastructures.

- **(R5.1) Internal API.** An internal Java API must be available to enable the creation of client-side Java applications.
- **(R5.2) JavaScript API.** Modern web applications rely on advanced JavaScript interactions. Therefore, COEUS must also include a direct JavaScript interface to its knowledge base.

→ **(R6) Interoperability with software tools.** The COEUS framework must assure interoperability with any external system.

- **(R6.1) REST API.** A generic REST API must be made available to permit the use of data from COEUS' knowledge base within any external system.

→ **(R7) Federation architectures.** COEUS' setup must support the creation of intelligent knowledge networks through the federation of data collected in each instance.

- **(R7.1) SPARQL API.** A SPARQL endpoint must be accessible to allow direct queries to each COEUS instance knowledge base.
- **(R7.2) LinkedData API.** A view adopting the LinkedData principles must be publicly available.

→ **(R8) Open-source availability.** All developed COEUS components must be provided through open-source licensing schemes.

Table 0-1 summarizes the relationships between these requirements and the problems encountered during our investigative literature analysis.

Table 0-1. Summary of relationships between the defined requirements and the issues overviewed in the researched scientific literature.

	R1	R2	R3	R4	R5	R6	R7	R8
Swertz *et al.*	●							●
Wilkinson *et al.*	●			●		●		
Cannata *et al.*	●	●	●			●	●	
Slater *et al.*				●		●	●	
Kozhenkov *et al.*			●	●				
Hepp *et al.*					●	●		
Cannata *et al.*		●	●					
Marx *et al.*			●	●				●
Cheung *et al.*						●	●	
Hazber *et al.*			●	●				

COEUS: A Semantic Web Application Framework

Semantic Web tools enable translucent relationships amongst data. The semantic web itself is a truly intelligent data network, with rich connections allowing for a better understanding of available knowledge. However, despite the immense possibilities surrounding semantic web technologies, its adoption has been dimmer than anticipated. Whilst stakeholders from all domains acknowledge the benefits of having a fully semantic information system, the difficult transition from traditional flat-file or relational database supported systems to the semantic web is a challenging roadblock.

Framework Setup

COEUS' "Semantic Web in a box" strategy envisages the inclusion, in a single package, of all the tools required to launch a new Semantic Web based application. In addition to this, COEUS' setup must also account for a flexible and scalable deployment environment. Many of the architectural decisions observed when implementing this framework had these ideals in mind. Hence, various tools and platforms were evaluated in the search for the optimal combination of components and integration/interoperability strategies that could transform semantic web rapid application deployment.

To better explain COEUS' strategy we employ a naming strategy that adopts a gardening metaphor. A single COEUS instance is entitled as **Knowledge Seed**, or simply **seed**. In scenarios with multiple **seeds** deployed in a true application ecosystem, this federated structure is envisaged as a **Knowledge Garden**.

Knowledge Representation

As mentioned in chapter 3, data in the Semantic Web are stored in formal triple statements: Subject-Predicate-Object. These statements employ different vocabularies and languages to identify each component. We can make a simple analogy to basic sentences with a **subject**, a verb representing action or meaning - the **predicate**, and what relates to the **subject** - the **object**. One last thing to consider is that predicates relate to object or data properties. Figure 0-2 highlights this division in a common sentence matched to a single statement.

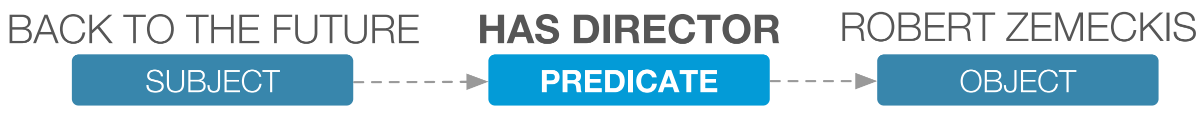


Figure 0-2. Sample triple statement, subject – predicate – object.

Taking in account the multitude of data models we can integrate within a single COEUS instance, the general semantic web knowledge representation strategy is more than fit. This allows us to map any content from CSV columns or SQL query results into a set o RDF statements.

For the knowledge storage framework component we identified and assessed various RDF management tools, as briefly covered in Table 0-2. The Jena framework is the most suitable alternative for COEUS' knowledge base. Its Java-based nature, easy integration with other tools and extensibility, make it ideal for use in a component-based framework. Jena's

API has basic support for reading and writing triple statements in Java in an in-memory or database-supported triplestore.

Table 0-2. Knowledge storage and representation technologies comparison.

FRAMEWORK	DESCRIPTION
Jena[10]	Widely used semantic web package for Java. Includes several features to easily deploy new applications, including support for SPARQL queries, RDF and OWL APIs, and inference. Provides multiple storage and reasoning mechanisms and also allows the integration of custom data processing mechanisms.
Sesame[11]	Widely used RDF framework and server. Includes support for SPARQL queries and an HTTP server interface. It is packaged with multiple storage and reasoning mechanisms and also allows the integration of custom mechanisms.
Virtuoso[12]	Widely used commercial solution for semantic web development. Includes a platform agnostic solution to access data through SPARQL queries, manage knowledge bases and integrate heterogeneous resources.
Redland[13]	Collection of RDF libraries for C, with bindings for various other languages. Provides RDF API, parsers, and query interfaces.
LinqToRDF	Semantic Web framework for.NET built on the Microsoft Language-Integrated Query (LINQ) Framework (language-independent query and data processing system).
OWL API[14]	OWL API and implementation for Java. Includes an OWL API that is built on the functional syntax of OWL 2 and contains a common interface for many other reasoners.

Components

The basic COEUS setup requires a Java application server (Tomcat is recommended) and a relational database (for the triplestore backend). All the other necessary components are included in COEUS package, further facilitating the creation of new systems from scratch.

Figure 0-3 highlights the component interactions in each standalone instance and Table 0-3 describes all used components and their purpose within the framework.

[10] http://jena.apache.org/
[11] http://www.openrdf.org
[12] http://virtuoso.openlinksw.com/
[13] http://librdf.org/
[14] http://owlapi.sourceforge.net/

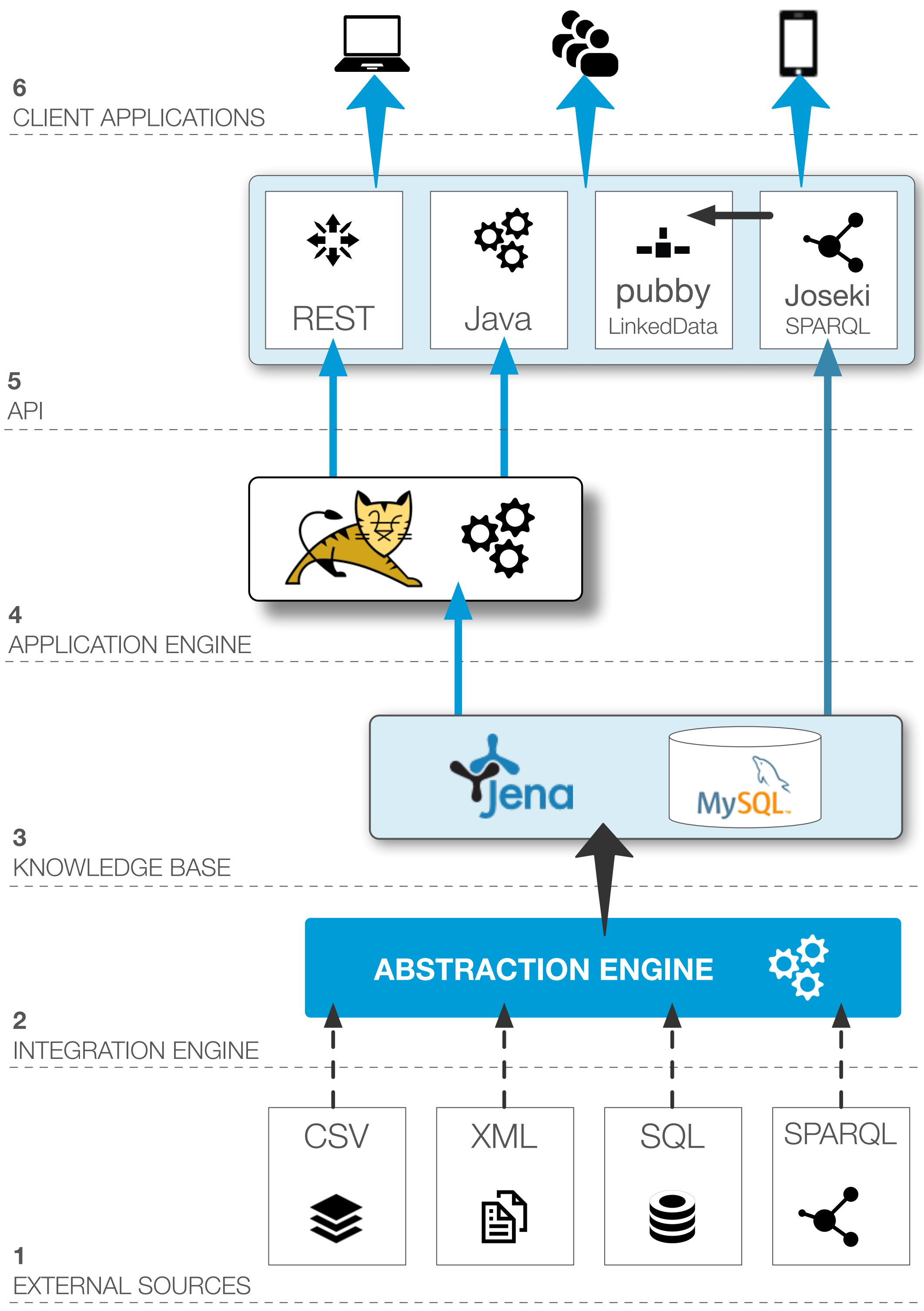


Figure 0-3. COEUS seed component interactions diagram. 1) External data are integrated from multiple sources using the available CSV, XML, SQL or SPARQL connectors. 2) The abstraction engine translates read data into the seed knowledge base. 3) COEUS internal triplestore is supported by Jena with a MySQL relational database backend. 4) Data in the knowledge base are accessed through the application engine for the Java and REST APIs. 5) The SPARQL endpoint, provided by Joseki, allows direct access to the knowledge base and is used by pubby to enable the LinkedData views.

Table 0-3. COEUS' framework libraries listing and descriptions.

LIBRARY	DESCRIPTION
Jena	Jena is used as the core semantic web package within COEUS, mediating input access to the knowledge base when building the triplestore and output access to the Java API.
Joseki[15]	Provides the SPARQL endpoint functionality.
Pubby[16]	Provides the LinkedData interface.
Sparql.js	JavaScript library to query remote SPARQL endpoints.
Tomcat	Java application server.
MySQL	Backend support to the Jena SDB triplestore.

Architectures

COEUS' application models and internal seed architecture can be combined in a single view, highlighting the knowledge flow from the data integration connectors to the interoperability API, detailed further in this chapter. The architecture for a single seed is show in Figure 0-4.

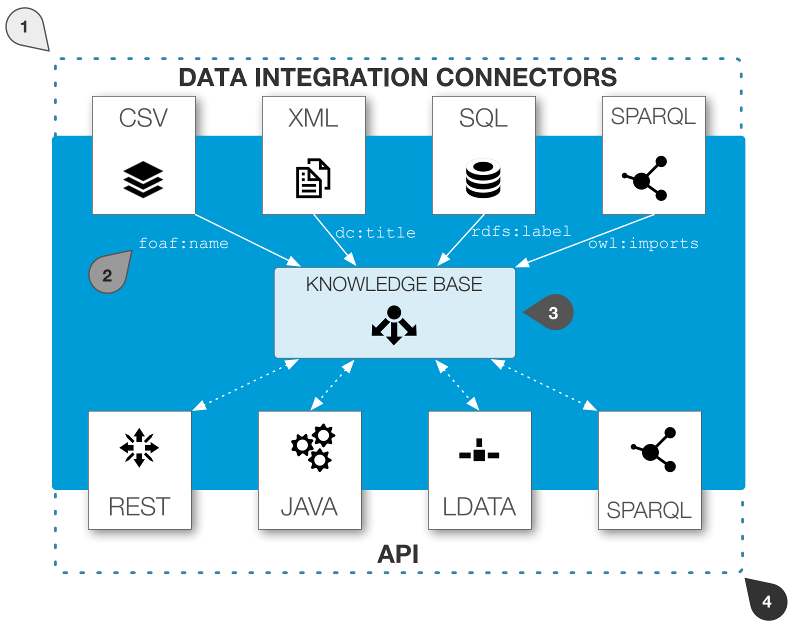


Figure 0-4. COEUS architecture for a single seed. 1) Data integration connectors for CSV, XML, SQL and SPARQL enable the triplification of data into each seed's semantic storage. 2) Data are selected from each external resource to match specific ontology predicates, generating a rich knowledge base. 3) COEUS central knowledge base includes the triplestore data repository and respective data access methods. 4) Acquired data are available through COEUS API, using Java methods, REST services, a SPARQL endpoint and the LinkedData view.

[15] http://www.joseki.org/

[16] http://www4.wiwiss.fu-berlin.de/pubby/

To further increase COEUS' innate flexibility, multiple seeds can be combined in a knowledge garden, providing a virtual holistic access layer to collected knowledge, regardless of its original location (Figure 0-5).

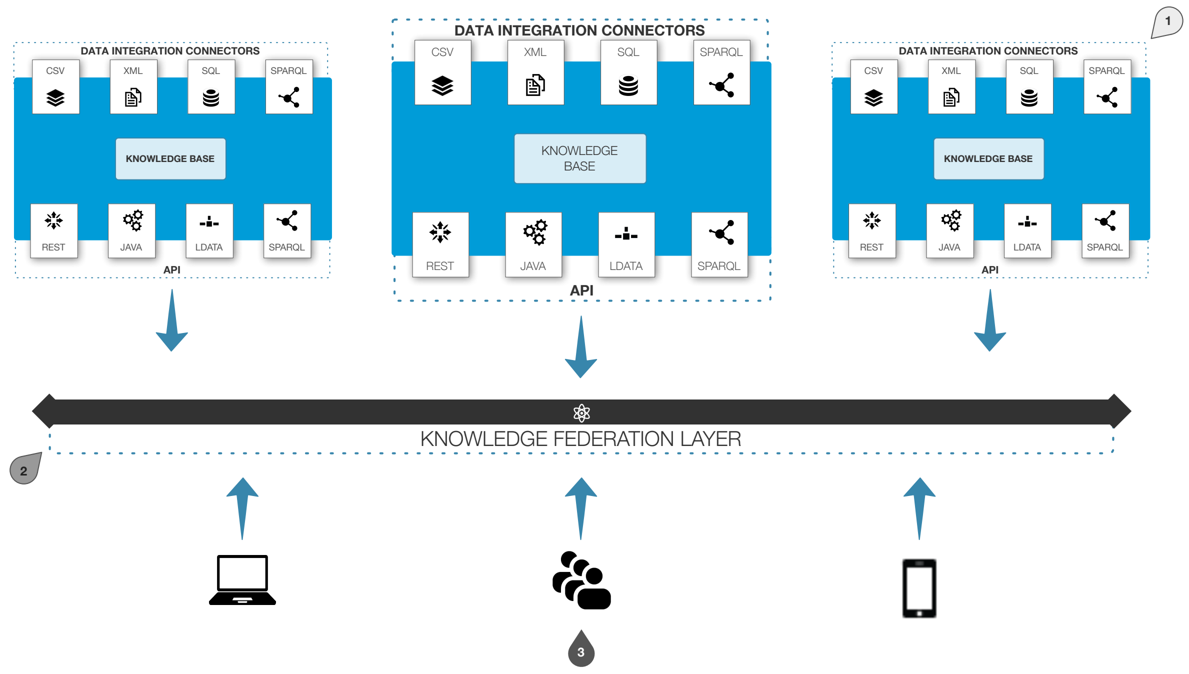


Figure 0-5. COEUS' garden architecture overview. 1) With one or more seeds in place, the COEUS platform enables a knowledge federation layer. 2) The distributed knowledge federation layer is capable of answering specific research questions. 3) This distributed entry point enables the deployment of multiple applications to web, desktop or mobile environments.

Application Models

The COEUS framework can be used to deploy distinct application models (Figure 0-6). On the one hand, a seed can accommodate multiple end-user applications, on distinct devices for instance. This permits that a single centralized data source can be built to support any number of web, desktop or mobile applications. On the other hand, multiple specialized seeds can be connected to supply a single holistic application. In this strategy all seeds work independently, and can be seen as nodes in a data sources network, providing access to an overarching tool. Along with these opposite strategies, hybrid architectures are also possible, combining multiple applications and seeds in a distributed data and applications ecosystem.

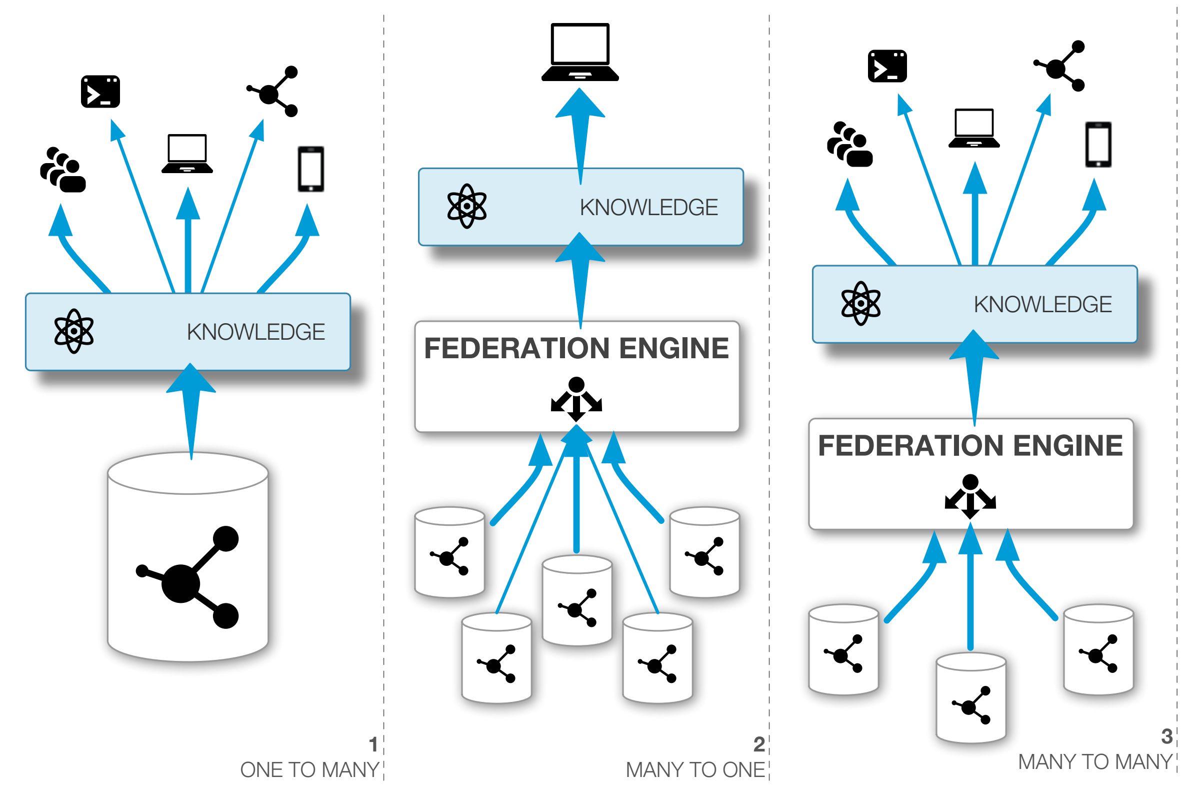


Figure 0-6. Basic COEUS application models. 1) One to many: one seed provides data to multiple client applications. 2) Many to one: using SPARQL federation multiple seeds are dynamically interlinked and accessed through a single application platform. 3) Many to many: hybrid model where multiple seeds are accessed by multiple client applications through a distributed federation layer.

Internal Ontology

To achieve the desired COEUS' scalability and flexibility, the basic platform model is organized in a tree-based structure. Data relationships are mapped to **Entity**-**Concept**-**Item** structures, which are connected to **Resources** and **Bridges**, supporting integration and exploration settings, respectively (Figure 0-7). This ontology is available online[17] and must be used in COEUS' configuration files. Describing COEUS entire ontology is out of this thesis scope. For further information regarding all classes, data and object properties or the entire structure, refer to COEUS' online documentation at http://bioinformatics.ua.pt/coeus/documentation/. A short description for each core class in the ontology follows.

- A **Seed** is a single framework instance. In COEUS' model, **Seed** individuals are used to store a variety of application settings, such as component information, application descriptions, versioning or authors. **Seed** individuals are also connected to included entities through the **:includes** property (inverse: **:isIncludedIn**). This

[17] http://bioinformatics.ua.pt/coeus/ontology/coeus_1.0b.owl

permits access to all data available in the seed, providing an overarching entry point to the system information.

- **Entity** individuals match the general data terms. These are "umbrella" elements, grouping concepts with a common set of properties. For example, to gather proteomics information, the model has a "Protein" entity or for disease information there is also a general "Disease" entity. To better understand this organization, object-oriented structures, their inheritance and variable subtypes must be remembered.
- **Concept** individuals are area-specific terms, aggregating any number of items (the **:isConceptOf** property) and belonging to a unique entity (the **:hasEntity** property). Continuing the previous scenario, "UniProt", "PDB" and "InterPro" databases are concepts within the "Protein" entity. Note that an **Entity** may have any number of concepts, but a **Concept** belongs to a single **Entity.**
- **Item** individuals are the basic terms, with no further granularity and representing unique identifiers from integrated datasets. In the above proteomics scenario, "P51587", "P02461" are items under the "UniProt" concept, each matching a unique term from the original UniProt database. For the disease entity, the "104300" individual is a match for Alzheimer's disease entry in the OMIM database concept. In the knowledge base, items can be associated to other items directly (predicate **:isAssociatedTo**) or through connections from their parent concepts. Entities are also connected to concepts, and these to items, making **Seed** individuals a central registry for COEUS' seeds.
- **Resource** individuals are used to store external resource integration properties. The configuration is further specialized with **CSV**, **XML**, **SQL** and **SPARQL** classes, mapping precise dataset results to the application model, through direct concept relationships. In the proteomics scenario, a Resource individual contains information for the "UniProt" concept original data source, including its location and how to extract each item. This resource is connected to several **XML** individuals (predicate **:loadsFrom**), each containing an XPath query whose results will map to application model properties. With the **:hasResource** property, the framework knows exactly what resources are connected to each concept and, subsequently, how to load data for each independent concept, generating new items.

→ **Brigde** individuals are also mapped to concepts, storing concept visualization and exploration features. That is, bridges tell the system how concept items can be shown to users. This configuration permits any number of internal properties as long as they are understood by the final client application. This means that we can include parameters for advanced data visualizations, triggering web service calls or composing simple links. An example of the latter is a bridge for the "UniProt" concept, declaring a structure for building valid UniProt links, replacing *#replace#* in http://www.uniprot.org/uniprot/#replace# with individual item identifiers.

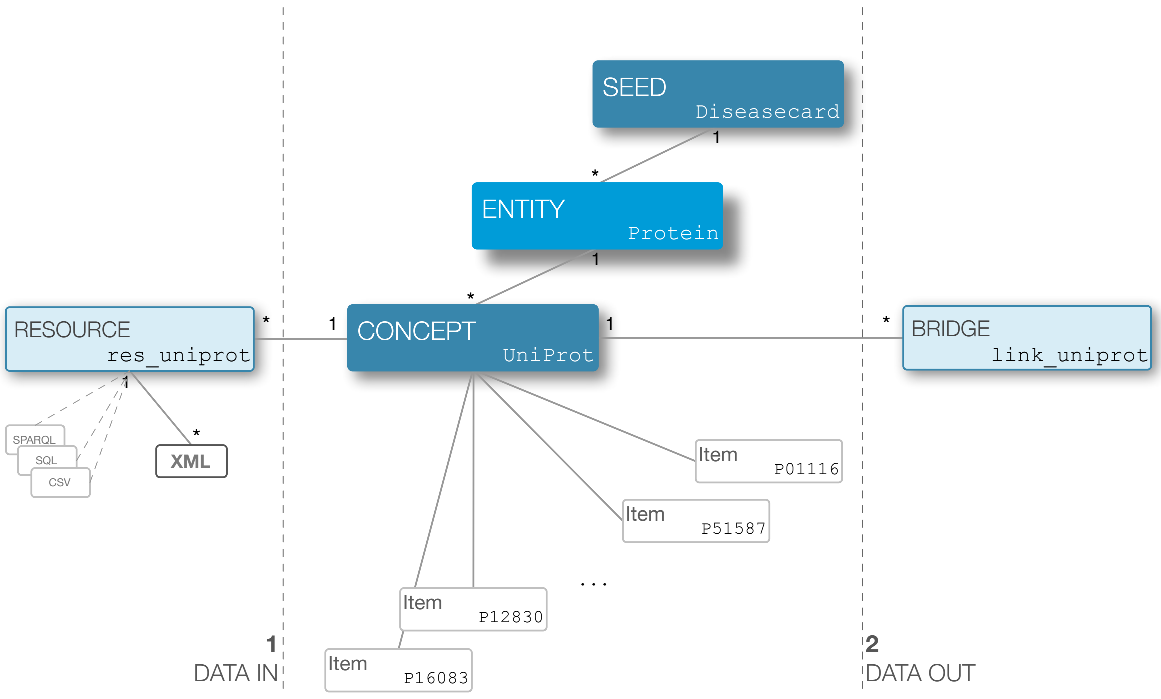


Figure 0-7. COEUS' ontology model for the internal tree-based structure highlighting relationships amongst the various individual classes. A seed can have multiple entities, and each entity can be related to one or more concepts. Concepts aggregate unique items and are connected to resource and bridge information. The data import process uses resources' properties (1) and custom methods can be defined to display data (2). Sample seed data for a "Diseasecard" seed is shown at each element, listing "UniProt" items belonging to a "Protein" entity.

One of the great advantages of using semantic web technologies is that any external ontology can be used to complement or extend COEUS' internal model. As long as new properties are understood by the seed applications, any number of properties can be added, mapping concepts or entities to existing ontologies or adding further properties to describe resources or bridges.

Data Flow

The COEUS framework gives developers total control over the data flow, from distributed repositories to the internal semantic knowledge base and from this to any end-user application. From a data input perspective, the goal behind this strategy is to provide developers with advanced methods to load precisely what they want, how they want it and from where they want it. Furthermore, on a data output perspective, we also want to provide enough flexibility for developers to build their own applications in any programming environment. To summarize, the inwards data flow establishes COEUS as a data integration platform (Figure 0-8) and the outwards data flow demonstrates its advanced interoperability features (Figure 0-9).

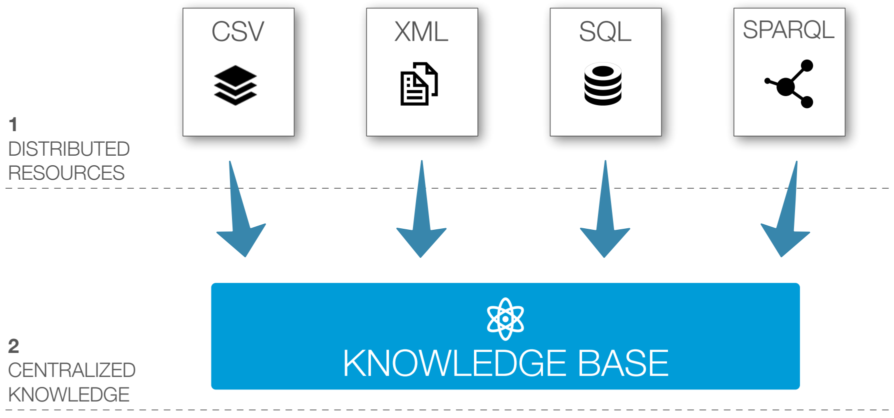


Figure 0-8. COEUS' inward data flow, from external distributed and heterogeneous resources (1) into a centralized knowledge base (2).

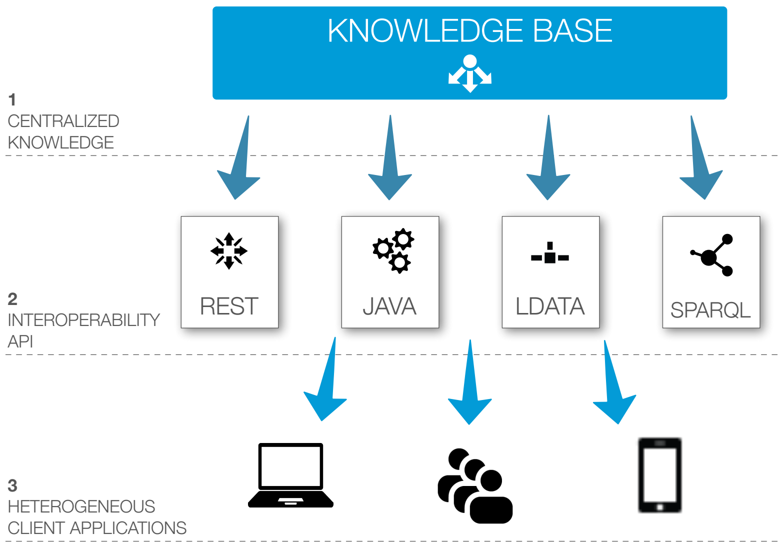


Figure 0-9. COEUS' outward data flow. Knowledge collected in the centralized knowledge base (1) is accessible through an interoperability API composed of four key interfaces: REST, Java, LinkedData and SPARQL (2). 3) The API enables the creation of miscellaneous client applications, target at multiple environments.

Extract-Transform-Load

Data integration is a perennial challenge in modern bioinformatics. As discussed in previous chapters, caveats such as resource distribution and heterogeneity transform integration into a demanding computer science challenge. One of COEUS' goals is to tackle this challenge. This framework provides features to facilitate the integration of heterogeneous data from distributed resources in an elegant fashion. Traditional warehousing techniques revolve around advanced algorithms for extracting data from a specific source, transforming it into the warehouse model and actually replicating the data in the new integrated dataset. In COEUS, this **Extract-Transform-Load** process is specialized to a semantic web environment, enhancing the inwards data flow from CSV, XML, SQL or SPARQL data to sets of triple statements.

Heterogeneity also appears in the distinct data models of each integrated resource. COEUS tackles this issue with a semantic web translation process. Due to COEUS roots, the internal knowledge base is model-agnostic, liberating integrated data from the restrictions of CSV tables or relational databases. Since all data are stored as triple sets, the limitations adjacent to foreign keys or table columns are replaced by meaningful relationships.

In most cases, the various properties stored in object-oriented models or XML structures can be re-engineered through the adoption of existing ontologies or the creation of new ones. As mentioned before, the usage of controlled ontologies augments the flexibility of internal data models, enabling the creation of tightly integrated datasets.

The physical and logical content heterogeneity issues impose the development of a generic data-loading tool. For simplicity purposes, a seed's configuration file includes the type of resource being loaded and the URI to access it. This way, all COEUS needs is an Internet connection to access REST or SOAP services, SQL databases or SPARQL endpoints. Furthermore, the URI naming scheme also permits the identification of local resources and SQL database connections can be made to a local host. This results in having the same structure in COEUS for importing local or remote data.

This immense amount of variables and configuration properties for integrating data lead to the appearance of the **connector** and **selector** concepts, explained further in this chapter.

Collecting Distributed Data

The initial problem that arises when building new warehouses or integrated datasets relates to the diversity of formats involved in the data import process. Whether we are accessing a REST web service or a MySQL database, most programming technologies allow configuring this access through a simple URI. For instance, a sample JDBC connection string to a MySQL database is

```
jdbc:mysql://thedbhost.com:3306/thedbname?user=thedbuser&password=thedbuserpwd
```

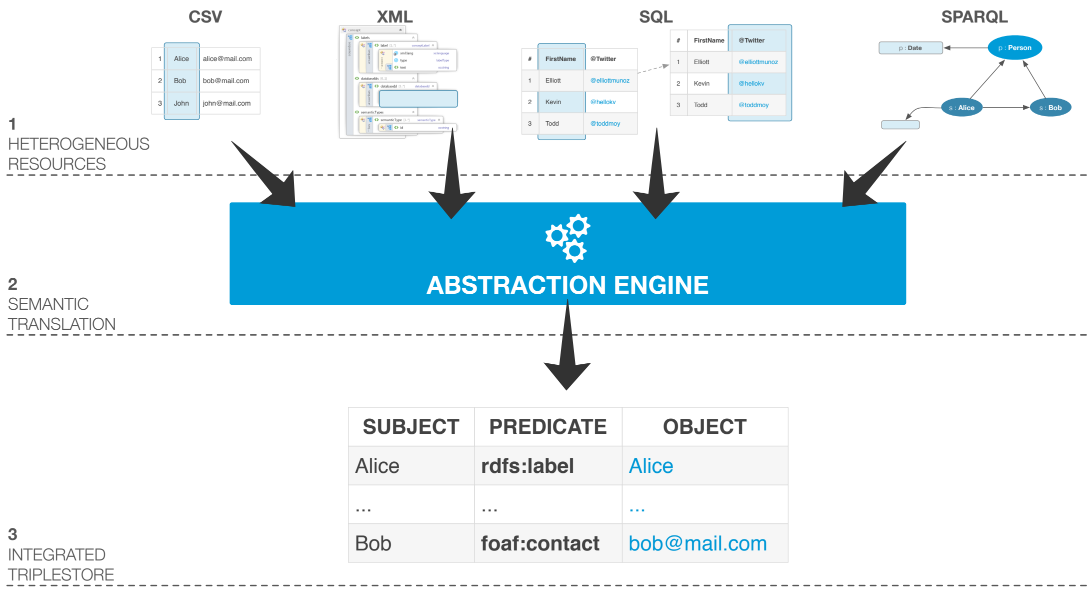
and a sample URL for accessing Twitter's API is

`https://twitter.com/#!/search/%40term.`

The similarities are clear and enable the simplification of the external resources configuration. All resources will have a URI property for instance.

If data format heterogeneity poses the initial threat for a linear data integration process, the data model heterogeneity further heightens it. We know from the start that data will come through in all sorts of formats and models. To overcome these caveats, COEUS adds an intermediate abstraction layer between the external resources and the internal knowledge base - Figure 0-10.

The idea behind this abstraction layer is to convert the data being integrated to a general model-independent format. In practice, the implemented method simply generates a network graph for each new item, mapping the configured predicates to the values from external resources. With this data abstraction, the triplification process can take place, enabling the generation of triple statements from the abstracted data model for further storage in COEUS' knowledge base.



SUBJECT	PREDICATE	OBJECT
Alice	rdfs:label	Alice
...	...	...
Bob	foaf:contact	bob@mail.com

Figure 0-10. COEUS' data abstraction process. 1) Data from external, distributed and heterogeneous resources is prepared for triplification. 2) COEUS connectors initiate the semantic translation using the abstraction engine. 3) Generate triples are stored in the seed knowledge base, a fully integrated triplestore.

Connectors and Selectors

The integration task consists in the acquisition of data from heterogeneous and distributed resources to populate a seed. This complex strategy required the construction of purpose-specific wrappers. These methods access external resources and process data, using the **connectors**, based on a set of configuration properties, the **selectors**.

Selectors are property sets defining the data location in a specific resource and what predicate will be added to the knowledge base during the integration process. Connectors control these particular data mappings: independent and generic modules to load information from external resources in CSV, XML, SQL or SPARQL formats. They possess a common set of configuration properties defining the data type, where the data are located, the relationships to existing data, and other module-specific definitions. This information is stored in the seed configuration files, exemplified in the following chapter. For instance, XML module configuration must include the original data source address and a collection of selector properties, XPath queries, which will be performed against the read XML, corresponding to the data being mapped. Likewise, SQL query column names, CSV column numbers, or SPARQL variables are used as selectors in their respective modules.

The data loading process uses connectors to initiate a data triplification process. Data are enriched through the dynamic generation of new triples based on specified configuration properties. With this Semantic Web-based Extract-Transform-Load we are augmenting the scope of data in one-dimensional CSV files or bi-dimensional SQL tables to a multi-dimensional triplestore.

The richness of this triplification process resides on connector's flexibility. The selectors within a given connector allows us to match any content into our semantic graph using a primary key for the subject, any property mapped from the seed ontology as predicate, and selection results as objects. These are then used to generate each new item map on-the-fly, which is then converted into a set of statements and inserted into the knowledge base.

Triplifying Content

The triplification process highlighted in Figure 0-11 may proceed in two modes: explicit and implicit. Explicit translations are required when data are read from CSV, XML or SQL resources. As data does not possess any related semantics (only columns or objects), explicit descriptions for the new predicates need to be set up. Since we can map data in XML, CSV or SQL to any predicate in any ontology and to more than one predicate at once, we can expand the meaning of non-semantic data by explicitly declaring it as the object of a specific statement.

SPARQL resources provide implicit semantics; data are already in the same semantic format (triple statements) as required by the storage engine. While loading data with the SPARQL connector, the selector can match data into new predicates or the original predicate. This way, COEUS can simply replicate triples or expand them to richer entities.

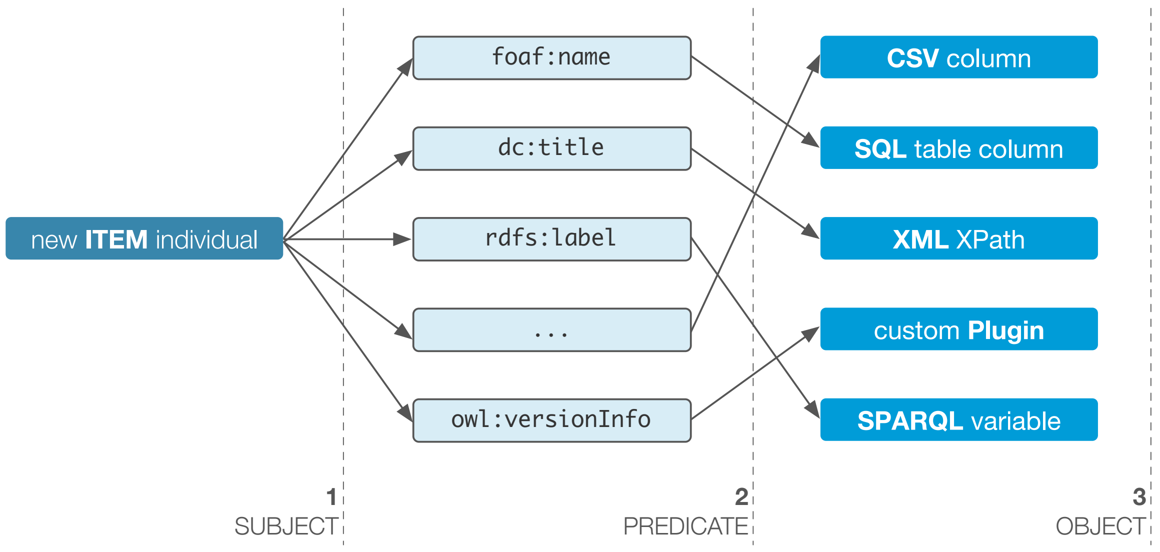


Figure 0-11. Triplification process overview. 1) Subjects, new Item individuals, are generated at runtime. 2) Predicates are read from the configuration file to match any predicate from any ontology. 3) Objects read from the external heterogeneous resources finish the triple statement.

In addition to these two triplification modes, data integration can also be performed using three distinct approaches according to the seed needs or to how data are provided by each service. These methods are defined by the **:method** property in a resource configuration. *Cache* is the default method and enables standard data loadings from external resources, generating new items and triplifying all data. The *complete* method adds new triples to items already in the seed triplestore. At last, the *map* integration method enables the creation of custom direct relationships amongst individuals. With this, we can create entire sets of new mappings amongst items after the data are loaded. Both the *map* and *complete* integration methods use the **:extends** configuration predicate from COEUS' internal ontology to define the Concept whose individual item list will be enriched.

With COEUS' triplification strategy, the data integration approach is abstracted from the data itself. Since there is ground for one or more common underlying ontologies, new axioms can be established disregarding traditional software constraints. Data can be collected and connected using distinct methods and miscellaneous import formats. This is ideal for optimizing all kinds of new data-powered applications, namely on the life sciences field, where heterogeneous data models with limited relationships are common.

Configuring a New Seed

The seed configuration controls the entire instance operability. At the moment, three separate files are used to set up application properties, the application model and resource integration properties.

- The ***config.js*** file stores volatile application properties. These include the application name, version, short description, deployment environment and the list of ontologies used in the seed. Using a JSON object for the configuration permits faster reads when compared to XML, while maintaining a good object-oriented structure, in comparison to simple properties files.
- The information system ontology. In most scenarios, the "reuse instead of rewrite" principle does not suffice for the entire application ecosystem. As such, COEUS allows the creation of custom ontologies to use in one or various seeds. Developers are able to organize their own applications models, taking full advantage of RDF/OWL's modelling flexibility.
- The application setup file includes the data integration and exploration configurations. In this file we define the individuals for each class, configuring entities, concepts, bridges and resources. Summarily, content in this file is used to guide the entire framework instance setup, from the handling of external resources in the connectors to the labelling rules for each Item individual.

COEUS' future developments include the addition of a user-friendly GUI to configure new seeds. In the meanwhile, and considering the setup files OWL/RDF nature, relying on Protégé is advisable to ease the configuration process. In this widely used ontology-modelling tool, the configuration can be written, tested and visually organized.

The amount and variety of configuration options is even greater than WAVe's. Hence, the best option is to look at the examples and online documentation at http://bioinformatics.ua.pt/coeus/documentation/. For mere descriptive purposes, a subsection from a real configuration file is included next. This code sample configures the loading of known human genes into a seed. This list, mentioned in WAVe's integration description, is maintained by the HUGO Gene Nomenclature Committee, which provides a REST service for getting the list in CSV format. This resource loads the list into our seed, populating a *HGNC* **Concept** under the *Gene* **Entity**. The **resource_HGNC** individual is configured to load the data from the selected **:endpoint** object, send it for processing using the CSV connector, property **dc:publisher**, and map the results from two **:CSV** individuals, property **:loadsFrom**. In these, the **:query** predicate defines which column object will map to the predicates listed in **:property**. Hence, in the **csv_HGNC_id** individual, data obtained from column *0* of the *HGNC* CSV file, property **:query**, will be mapped into two triple statements with the same subject, the **dc:source** and **dc:identifier** predicates and with the same object, property **:query**.

```
# HGNC Resource configuration
:resource_HGNC rdf:type :Resource , owl:NamedIndividual ;
   rdfs:label "resource_hgnc"^^xsd:string ;
   dc:title "HGNC"^^xsd:string ;
   :method "cache"^^xsd:string ;
   dc:publisher "csv"^^xsd:string ;
   :endpoint "http://www.genenames.org/cgi-
   bin/hgnc_downloads.cgi?title=HGNC+output+data
&hgnc_dbtag=onlevel=pri&=on&order_by=gd_app_s
ym_sort&limit=&format=text&.cgifields=&.cgifi
elds=level&.cgifields=chr&.cgifields=status&.
cgifields=hgnc_dbtag&&where=&status=Approved&
status_opt=1&submit=submit&col=gd_hgnc_id&col
=gd_app_sym&col=gd_app_name&col=gd_status&col
=gd_prev_sym&col=gd_aliases&col=gd_pub_chrom_
map&col=gd_pub_acc_ids&col=gd_pub_refseq_ids" ^^xsd:string ;
   :extends :concept_HGNC ;
   :isResourceOf :concept_HGNC ;
   :hasKey :csv_HGNC_id ;
   :loadsFrom :csv_HGNC_id,:csv_HGNC_symbol.

# HGNC CSV connector configuration for HGNC identifier
:csv_HGNC_id rdf:type :CSV , owl:NamedIndividual ;
   rdfs:label "csv_hgnc_id"^^xsd:string ;
   :query "0"^^xsd:string ;
   dc:title "HGNC_id"^^xsd:string ;
   :property "dc:source|dc:identifier"^^xsd:string ;
   :loadsFor :resource_HGNC ;
   :isKeyOf :resource_HGNC.

# HGNC CSV connector configuration for HGNC name
:csv_HGNC_name rdf:type :CSV , owl:NamedIndividual ;
   rdfs:label "csv_hgnc_name"^^xsd:string ;
   :query "2"^^xsd:string ;
   dc:title "HGNC_name"^^xsd:string ;
   :property "rdfs:comment|dc:description"^^xsd:string ;
   :loadsFor :resource_HGNC.
```

For improved dependency management, seed configurations are organized as graphs. That is, developers can implement dependencies amongst resources, enabling the loading of data based on previously collected individuals, selected with the **:extends** property. This allows for the creation of advanced data integration workflows, combining multiple concepts, thus enabling the aggregation of millions of triples in the seed's knowledge base.

APIs

COEUS tackles the lack of interoperability in existing life sciences information systems. Drawbacks such as poor web service availability, complex and closed data models, or vendor-specific formats are common in bioinformatics. To overcome these clear issues in semantic interoperability, COEUS includes, by default, a comprehensive API to explore collected data.

Available methods were developed with two goals in mind. On the one hand, data must be easily available for the creation of new applications within a COEUS seed. On the other hand, integrated data must also be published externally, making it available for any external system. Hence, COEUS' API is organized in two sections: internal and external, despite their natural promiscuity.

The internal API comprises the Java methods and Javascript libraries. The former provides an abstraction over Jena's basic data access functions and are adequate for scenarios where the seed client-side application is also being developed in Java. These methods permit data access in both ways, allowing for streamlined data access and traditional data insertions. The available Javascript library simplifies the process of accessing a SPARQL endpoint using Javascript. Combining this tool with modern user interface frameworks (such as jQuery) makes it very easy to query a seed's knowledge base and use the response data in the application. Consequently, developers can create highly interactive user interfaces, in any development framework. Moreover, custom endpoints can be configured in the JavaScript library to access data from external SPARQL endpoints. This enables the creation of modern semantic data mashups on the client-side.

The external API comprises a set of REST services, a SPARQL endpoint and a LinkedData viewer. The available REST services allow accessing content in multiple formats (CSV, JSON, RDF/XML or HTML). Likewise, the SPARQL endpoint is open for querying. With this endpoint, any query can be performed to exploit the wealth of integrated data. At last, the LinkedData perspective makes the knowledge base content available to any LinkedData browser, delivering an advanced structured interface to access data.

Java and Jena

While Jena provides a developer-friendly API for adding and retrieving data in Java, COEUS includes an additional set of methods to ease these tasks and facilitate data access.

COEUS Java API is an abstraction over Jena's internal methods, providing a more direct way to access COEUS data structures. Hence, accessing items, concepts or entities, or adding new statements actions are more straightforward. Next, there are the signatures for functions to add new statements and retrieving the result set of a SPARQL query. More examples and full documentation can be found online in the Java documentation at http://bioinformatics.ua.pt/coeus/javadoc/.

REST

COEUS' RESTful services API provides a set of methods to access data in the knowledge base through simple GET requests. REST services are currently the most widely used strategy for systems interoperability. Modern service-oriented architectures rely on these types of services due to their flexibility in regard of formats and operation types. The trade-off between having a more standardized (though constrained) services platform using SOAP and a more "open" alternative with REST was acceptable for COEUS, promptly pushing the latter as the only viable solution for supported services in COEUS.

Furthermore, in spite of the relatively low number of REST services available by default, more services can be easily added through the combination of internal Java methods with Stripes' powerful URL binding mechanisms. The Stripes framework has a very light learning curve, enabling the addition of new actions and services an easy job even for non-experienced Java web developers.

The highlight from the REST service set is the triple request method. This service enables building custom statements with specific subject, predicate or object properties, which are mapped into a SPARQL query to an instance's knowledge base (Figure 0-12). For example, http://bioinformatics.ua.pt/coeus/api/triple/coeus:hgnc_COL3A1/pred/obj/js returns a JSON object with all statements where the item **coeus:hgnc_COL3A1** (human gene COL3A1) is the subject from COEUS sample dataset. Similarly, http://bioinformatics.ua.pt/coeus/api/triple/sub/coeus:hasEntity/obj/xml returns XML detailing all subjects and objects related with a **coeus:hasEntity** predicate. In COEUS' ontology, this lists all concepts and respective entities.

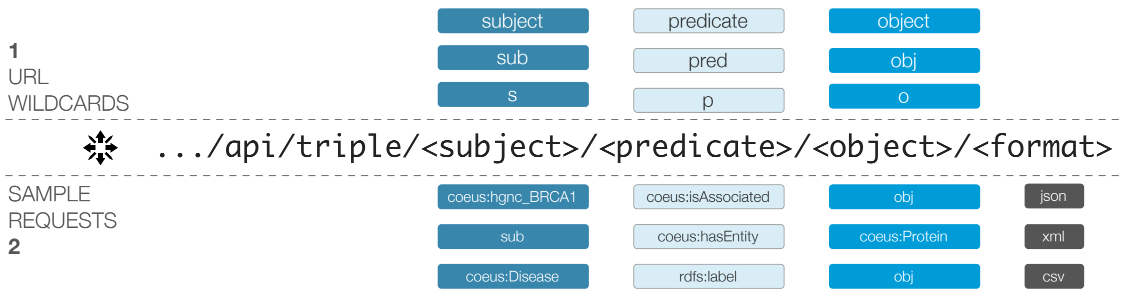


Figure 0-12. COEUS REST API summary. 1) Various wildcards can be combined to form valid requests and access all data in the knowledge base. 2) Sample REST requests highlighting the different output formats and wildcard use.

At last, the major advantage of using the available REST services is the access of data in multiple formats. Whereas requesting data in JSON format is optimal for lightweight web application development, one might need to import data in CSV format into an Excel sheet or transform XML content into a new database. This variety further increases COEUS' overall flexibility, improving its usage in modern application platform environments.

SPARQL

Another COEUS' API feature is the default SPARQL endpoint. With an open SPARQL endpoint, users or developers have full access to a seed's knowledge base, enabling complex queries and more insightful data retrieval operations.

Much like the set of REST services, the SPARQL endpoint also enables getting data in multiple formats, promoting its easier integration with client-side applications (discussed in the Advanced User Interactions section next). A form for querying each seed triplestore is available by default in all seeds at*../sparqler*. This form allows developers to test their SPARQL queries before including them in the application code.

LinkedData

Nowadays, the hottest topic in data sharing and interoperability is LinkedData. Through its multiple subdivisions, the LinkedData guidelines empower a completely interoperable knowledge ecosystem, where resources are directly accessible through their URIs and their semantic descriptions establish meaningful connections to other miscellaneous data types.

COEUS uses the **pubby** package to publish the knowledge base as LinkedData. A simple configuration file defines the connection properties to access the seed SPARQL endpoint and retrieve data. For each resource being browsed, the application issues a DESCRIBE to obtain all object relationships.

```
<!-- DESCRIBE <http://bioinformatics.ua.pt/coeus/resource/uniprot_P51587> -->
<?xml version="1.0"?>
```

```
    <rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
        xmlns="http://bioinformatics.ua.pt/coeus/"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
        <isAssociatedTo
rdf:resource="http://bioinformatics.ua.pt/coeus/prosite_PS50138"/>
        <isAssociatedTo
rdf:resource="http://bioinformatics.ua.pt/coeus/mesh_D010051"/>
        <isAssociatedTo
rdf:resource="http://bioinformatics.ua.pt/coeus/interpro_IPR015525"/>
        <isAssociatedTo
rdf:resource="http://bioinformatics.ua.pt/coeus/pdb_3EU7"/>
        <dc:identifier>P51587</dc:identifier>
        <dc:source>P51587</dc:source>
        <isAssociatedTo
rdf:resource="http://bioinformatics.ua.pt/coeus/pdb_1N0W"/>
        <hasConcept
rdf:resource="http://bioinformatics.ua.pt/coeus/concept_UniProt"/>
        <rdfs:label>item_P51587</rdfs:label>
        <isAssociatedTo
rdf:resource="http://bioinformatics.ua.pt/coeus/hgnc_BRCA2"/>
        <isAssociatedTo
rdf:resource="http://bioinformatics.ua.pt/coeus/mesh_D001943"/>
        <dc:title>P51587</dc:title>
        <isAssociatedTo
rdf:resource="http://bioinformatics.ua.pt/coeus/mesh_D010190"/>
        <isAssociatedTo
rdf:resource="http://bioinformatics.ua.pt/coeus/mesh_D005910"/>
      </rdf:Description>
    </rdf:RDF>
```

When these data are delivered through the web interface, users can explore LinkedData innate connections, which allow users to jump from object to object within the same seed, in an external seed or accessible through a normalized URI.

With the LinkedData interface, COEUS completes the interoperability features required to enhance modern service composition ecosystems. This ability to make the integrated and enriched data available in the Linked Open Cloud without complex configuration tasks or tricky deployment processes is a defining feature for COEUS, taking it further in semantic web for life sciences innovation.

Advanced User Interactions

Modern application development relies on advanced browser-based capabilities to deliver more compelling user interactions. The latest versions of all major browsers include powerful JavaScript processing engines, like Google's V8[18] or Mozilla's JagerMonkey[19], with

[18] http://code.google.com/p/v8/

outstanding performances on the client-side. This triggered an evolution on web-based application, making them able to deal with larger amounts of data and increasing their processing capabilities.

In addition, JavaScript development frameworks such as jQuery[20], MooTools[21] or SproutCore[22], include methods to further rival server-side data handling and computational capabilities. This allows for the development of increasingly interactive web applications, reducing the thin line that separates them from desktop-based applications.

It is important for COEUS to also take part in this emerging and fast-growing application trend. Therefore, COEUS includes a JavaScript library (available under *assets/js/sparqler.js*) that enables direct connections to each seed's SPARQL endpoint. With this, it is possible to ask queries to and process data directly from the knowledge base with a powerful querying language. Data are retrieved as a JSON object easily handled in JavaScript. This library further increases rapid application prototyping and interface development in COEUS.

Case Studies

Exploring Collected Data

A trivial case study can be setup to test the various elements composing COEUS APIs. For this matter, knowledge regarding the **breast cancer type 2 susceptibility protein** (UniProt accession number P51587) will be collected from COEUS sample dataset.

These results are obtained from the graph of relationships where a representation of this individual, mapped in COEUS' sample knowledge base as **coeus:uniprot_P51587**, is an active subject. The methods for accessing these data are detailed next.

- **Java.** To obtain these data in Java, the ***getTriple()*** API method must be invoked, defining what filter to use and the desired XML output format.

```
/* Invoke getTriple("coeus:uniprot_P51587", "p", "o", "xml"); */
pt.ua.bioinformatics.API.getTriple(…);
```

- **REST.** The desired protein data can be obtained, in CSV format for example, through a direct GET request to the public REST interface at http://bioinformatics.ua.pt/coeus/api/triple/coeus:uniprot_P51587/p/o/csv.

[19] https://wiki.mozilla.org/JaegerMonkey

[20] http://jquery.com/

[21] http://mootools.net/

[22] http://sproutcore.com/

- → **SPARQL.** UniProt P51587 data can be queried from COEUS' SPARQL endpoint, available at http://bioinformatics.ua.pt/coeus/sparql, with the following query.

```
# SPARQL query to issue
PREFIX coeus: <http://bioinformatics.ua.pt/coeus/>
SELECT ?p ?o {coeus:uniprot_P51587 ?p ?o}
```

  Any query can be tested at http://bioinformatics.ua.pt/coeus/sparqler/.

- → **LinkedData.** The requested protein data can be explored through a LinkedData browser pointed to http://bioinformatics.ua.pt/coeus/resource/uniprot_P51587. Additionally, the same address provides a summary view for regular web browsers.

Promoting a Federated Knowledge Ecosystem

The execution of federated SPARQL queries enables access to data across multiple sources in a single transaction. Whether data are locally stored or in a remote location, the query engine uses the **SERVICE** property to acknowledge where a specific question should be asked.

Every COEUS seed includes a SPARQL endpoint by default. With multiple seeds in place, it is fairly easy to perform queries across the various COEUS instances, inferring results on the fly. This virtual distributed knowledge network, the aforementioned knowledge garden, opens up immense data integration and interoperability possibilities. In modern national health information systems scenarios, launching multiple seeds with similar data models and targeted at regional subsets, originates a federated knowledge ecosystem. Applications can access each seed individually, cross data between two or more seeds, or have an holistic perspective over the entire knowledge garden.

A case study for COEUS' federation support regards the answers for following scientific question: *What are the PDB identifiers for the protein structures and the MeSH term identifiers associated with the human BRCA2 gene?*

To answer the proposed question, the federated query shown next links four distinct services, i.e. SPARQL endpoints. COEUS' default SPARQL is replicated three time to virtually simulate the query distribution. The query is processed in real time through the SPARQL endpoint, with the following steps:

1. The Diseasome SPARQL endpoint is queried to obtain the label for the human BRCA2 gene (**?label**).
2. The **?label** variable is passed to the first COEUS seed, acting as the selection clause for the gene and enabling access to a set of UniProt proteins associated with it (**?uniprot**).

3. The **?uniprot** variable is shared with the third and fourth SPARQL endpoints, where data regarding PDB identifiers (**?pdb**) and MeSH term identifiers (**?mesh**) is selected. This last request could be executed in a single query, but is divided to further demonstrate COEUS' federation capabilities.

```
   # Federated SPARQL query
   PREFIX dc: <http://purl.org/dc/elements/1.1/>
   PREFIX                    diseasome:                    <http://www4.wiwiss.fu-
berlin.de/diseasome/resource/diseasome/>
   PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
   PREFIX coeus: <http://bioinformatics.ua.pt/coeus/>

   SELECT ?pdb ?mesh
   WHERE{
       {
           SERVICE <http://www4.wiwiss.fu-berlin.de/diseasome/sparql>
           {
                   <http://www4.wiwiss.fu-
berlin.de/diseasome/resource/genes/BRCA2> rdfs:label ?label
           }
       }
       {
           SERVICE <http://bioinformatics.ua.pt/coeus/sparql>
           {
                   _:gene dc:title ?label.
                   _:gene coeus:isAssociatedTo ?uniprot
           }
       }
       {
           SERVICE <http://bioinformatics.ua.pt/coeus/sparql>
           {
                   ?uniprot coeus:isAssociatedTo ?pdb.
                   ?pdb coeus:hasConcept coeus:concept_PDB
           }
       }
       {
           SERVICE <http://bioinformatics.ua.pt/coeus/sparql>
           {
                   ?uniprot coeus:isAssociatedTo ?mesh.
                   ?mesh coeus:hasConcept coeus:concept_MeSH
           }
       }
   }
```

This, and any other federated queries, can be tested online at http://bioinformatics.ua.pt/coeus/sparqler/. Additionally, more complex queries can be built combining these data with any other SPARQL endpoint.

Despite being targeted at life sciences developers, COEUS can be used in various other real world settings. In either the corporate domain or TV networks, data are generated in large quantities and with complex innate relationships. While we do not envisage COEUS replacing already setup infrastructures in these areas, the framework is suitable for quickly deploying ad-hoc knowledge bases.

For example, a news channel web application can be built to aggregate information on a selected topic, from various media sources, in a single environment. With COEUS, content from Twitter, Facebook or any modern news site (using RSS/Atom feeds) can be quickly pulled into a new repository, enabling the creation of semantically richer applications for both web and mobile environments. Furthermore, the resulting dataset can be also used to improve existing applications. A new semantic layer can be incorporated in the client-side of modern web applications by querying and loading data in JSON format

Pharmaceutical companies can also use COEUS. A virtual scenario uses the COEUS platform with a well-designed ontology to create a Semantic Web-powered infrastructure to manage specific in-house datasets. Managing marketing results or large clinical trials data can be improved by establishing COEUS seeds, each with its own goals and needs, and allowing for future connections amongst these initially disparate data through COEUS' knowledge federation features.

Features and Usability

Rapid Application Development

COEUS' "Semantic Web in a box" approach streamlines the creation of new Semantic Web applications. The development of new semantic systems is highly associated with a steep learning curve and a myriad of technologies and tools to chose from. Although this variety is beneficial, it is also a characteristic of a still immature deployment environment. Unlike traditional relational database applications where the "technology path" is clearly outlined, with semantic web applications the adequate set of technologies and strategies continues to be chosen in *ad hoc* fashion.

COEUS provides the means for semantic web rapid application deployment by offering a single package comprising the set of tools required to develop a new application from scratch. Moreover, the application backend, the knowledge base, can be populated through advanced data integration wrappers that use flexible configuration ontology.

Incorporating the interoperability API with the integration features results in a framework that highly reduces the application "time-to-market". It is easy to get the data in the system. Likewise, it is easy to get the data out the system. This facilitates the creation of independent application platforms, supported by a comprehensive backend knowledge base that enables deploying to desktop, web or mobile systems. The API also permits coding client-side applications in any programming environment and using any framework, further improving COEUS flexibility and robustness.

Data Integration Platform

Data integration is the initial cornerstone for the COEUS framework. Its powerful resource integration capabilities enable the creation of customized niche-based data warehouses, powered by a semantic knowledge base. Distributed and heterogeneous data can be replicated or linked, taking advantage of semantic web's advanced data modelling capabilities to overcome schema mappings and internal wrappers for general data retrieval.

Data in CSV, SQL, XML or SPARQL formats are easily configured for integration, smoothing the transition from traditional data storage approaches to a modern semantic web reality. This migration is further improved through the advanced extract-transform-load warehousing features, providing a simple strategy for generating triple sets from any kind of data type. Moreover, custom plugins can be developed to match scenarios that do not fit COEUS' capabilities yet.

COEUS aids in the publishing of semantic web-powered knowledge bases, moving one step further to the envisaged view of the Internet as a semantically rich distributed knowledge network.

LinkedData & Semantic Services

Once data are integrated into a seed's knowledge base they are promptly available through various APIs. Firstly, the internal Java API layer hides away all complexities regarding semantic triple stores and data structures, offering a set of methods to retrieve data directly as an iterable result set.

Next, the REST services API encompasses simple GET-based methods to access data. The *triple* service offers a quick way to iteratively load all data into any application development environment.

The SPARQL endpoint is the most powerful interoperability feature. Besides supporting the LinkedData infrastructure and the client-side JavaScript library, it makes all data available through a standardized and efficient query engine. Complex queries can be asked and processed in command-line tools, web clients or desktop applications, further increasing the wide scope of COEUS' use. The SPARQL endpoint is also the underlying entry point for knowledge reasoning and inference features.

The LinkedData interface empowers the availability of integrated data in the most advanced data interoperability scenarios. With URLs uniquely and precisely identifying data descriptions numerous possibilities for service composition arise, taking the most out of connected, i.e. linked, data.

At last, the included JavaScript library enables creating best-of-breed user interactions, handling all data access and processing on the client-side. From the web application development perspective, this is the most interesting feature, as it enables the development of more responsive interactions in desktop- or mobile-based systems, taking web applications to the next usability level.

Knowledge Federation Framework

The challenge of federating knowledge scattered through multiple independent databases is also tackled in this framework. New seeds automatically launch SPARQL endpoints and LinkedData views, endowing developers with multiple ways to access and federate data.

Whether we are dealing with SPARQL-based federation or virtual LinkedData networks, data are inherently distributed and connected. With these technologies, anyone can launch his own customized and focused application ecosystem. In a COEUS' knowledge garden, the holistic view over all data empowers the sharing of knowledge amongst a scalable of numbers of peers, improving the federation of and facilitating access to data.

Future Developments

COEUS is an active project, published as open source with the purpose of captivating interest in new developments, thus creating a community surrounding the framework. Foreseen developments are focused on three main areas: improve the transition from monolithic systems to a semantic web environment, simplify the configuration of new seeds and provide new methods to input and output data from a seed.

Firstly, a **migration assistant** tool will be developed to smooth the transition from relational databases, CSV or XML structures into the semantic web paradigm. Leveraging on tools such as D2R we aim to create algorithms that read database structures and generate COEUS configuration models dynamically. For instance, automated processes to discover **Entity-Concept** organizations or internal data/object properties and import content on the fly will ease the creation of new seeds. Consequently, it will be much easier for bioinformaticians to transform their platforms and access all COEUS' integration and interoperability features.

The migration assistant will also feature **simplified configuration interfaces**. While now developers need to configure new seeds in Protégé or text-editor, a GUI-based setup & installation tool will be available in the future. This is aimed at non-expert bioinformatics developers that would rather fill in forms and click buttons than edit configuration settings *by hand*. Another step towards the simplification of seeds creation is the creation of a

COEUS virtual machine image pre-build with all required tools. In this case, the goal is to use a solution like TurnKey[23] to offer a disk image with the required application server, database and COEUS seed ready for deployment in a real-world scenario. Furthermore, this will also empower future COEUS integration with cloud-based developments.

Next, the API will also be augmented with **new applications and tools**. OS-specific applications and command-line tools for accessing COEUS' endpoints will be created. These will be an even better fit for a bioinformaticians research workflow. For instance, old-school biologists traditionally use basic shell scripts to perform data filtering and enriching operations. These can be enhanced with access to a COEUS knowledge base allowing the integration of state-of-the-art integrative datasets with legacy tools. Another opportunity concerns COEUS service composition. REST services will be improved and new ones developed to ease the process of combining COEUS services in Taverna workflows. With Taverna as the *de facto* workflow platform, it is advisable to foster COEUS use in this service composition environment.

Discussion

Ad-hoc Software Solutions VS Rapid Application Development

Developing tailored *ad hoc* solutions is the current practice in bioinformatics. Solutions like the ones highlighted in chapters 4 and 5 (EU-ADR Web Platform and WAVe) play a fundamental role in the evolution of the way bioinformatics software is developed. In spite of the recent turn of events in the innovative technologies side, where previously built packages are preferred over deployment from scratch, we must realize that bioinformaticians are not "regular" developers. The traditional bioinformatician's background usually lacks computer science skills, such as database management, modelling or object-oriented programming. Since most stakeholders fit this profile, it is easy to understand the biased focus on building new systems from scratch, paying little to no consideration to existing platforms, frameworks or programming libraries.

On the other end of the spectrum is the use of rapid application development strategies. By considering RAD ideals in a very broad sense, we observe that its practices are already being used in the majority of innovative technological platforms. Reusable assets are being used more often whether in the form of fully-fledged application frameworks, user interface bootstrapping packages or simple external libraries. Over time, the inclusion of

[23] http://www.turnkeylinux.org/

these components in new systems became easier, empowering the creation of new tools and disseminating the adoption of RAD ideals. The created strategies that empower COEUS build on this growing use of RAD principles, aiming at its use to create innovative biomedical applications.

Enhancing Rapid Application Development

The overall concept of RAD strategies for bioinformatics is still in its infancy. Molgenis is one successful case in the area, with a robust framework for launching new bioinformatics applications very quickly. The room for improvements over general RAD and bioinformatics-specific RAD is tied to two domains: integration & interoperability research and the semantic web paradigm.

RAD frameworks are not prone to facilitating the integration of data from external resources. Whereas the ability to quickly deploy data stores is omnipresent, RAD frameworks lack the features required to easily populate those data stores. The multiple challenges associated with the integration of data in any field, detailed along this thesis, are cumbersome for bioinformatics developers. Hence, the inclusion of integration features is deemed vital for bioinformatics RAD frameworks. Collecting and transforming data from CSV, SQL or XML files into a centralized knowledge base is a must-have feature in a field riddled with data heterogeneity and distribution. COEUS achieves this through a flexible integration engine, allowing the mapping of existing content into any ontology, and storing generated triples in a centralized knowledge base. Continuing DynamicFlow and WAVe's pursuit of the best service description strategy for data integration, COEUS uses an adaptive ontology to organize and configure a set of integrated resources.

As previously mentioned, the semantic web paradigm adoption and acceptance by the life sciences community is growing and it emerges as a viable alternative to lead biomedical software to a new level with tighter integration and better interoperability. The applicability of semantic web's ideals fits perfectly the complex life sciences challenges set. However, the steep learning curve associated with semantic web technologies is drawing users away from this new world. As such, the opportunity arises for the inclusion of semantic web technologies and features within a rapid application development package. For this matter, bioinformaticians must think about triplestores instead of relational databases, about SPARQL endpoints instead of SQL hosts or about LinkedData instead of SOAP-based data exchanges. The semantic web empowers a new services layer that allows the creation of truly federated intelligent data networks. Combining LinkedData with

SPARQL endpoints we can connect and exploit the wealth of data from miscellaneous data stores. As stated in the initial requirements, COEUS includes, by default, a SPARQL endpoint and support for LinkedData views, enabling truly semantic access to data collected in a single seed or federated from multiple COEUS instances.

A Framework for Semantic Bioinformatics Software

The next-generation of bioinformatics software will be empowered by the combination of two grand innovations that are diluting the boundaries between computer and life sciences.

On the computer science standpoint, the adoption of agile strategies to develop new applications is pushing forward the adoption of generic rapid application development methods, from reusable programming packages to user interface prototype building. For life sciences, enhanced biomedical semantics are the cornerstone for a better understanding of our human condition. While it will not solve all problems in bioinformatics, the semantic web emerges as the most viable alternative to build the next-generation of biomedical knowledge.

The COEUS framework is our approach to tackle these challenges and produce a next-generation semantic web rapid application development framework. The innovative "Semantic Web in a Box" approach encloses four major pioneering roles.

- The adoption of **rapid application development** strategies in COEUS endows developers with the tools to quickly build new application ecosystems targeting any deployment environment.
- COEUS is a **semantic data integration platform** enabling the acquisition and translation of heterogeneous data from distributed resources intro a centralized knowledge base.
- COEUS provides **Semantic Web and LinkedData services** by default. This ensures the interoperability of integrated data with any external system through open standard methods. Moreover, with a semantic knowledge base in place, support for reasoning and inference strategies is facilitated.
- COEUS enables the **federation** of gathered knowledge through comprehensive APIs. The SPARQL endpoint and LinkedData interfaces empower querying and reasoning over multiple COEUS instances.

The COEUS framework is an open-source project. Documentation and code samples are available online at http://bioinformatics.ua.pt/coeus/.

A COEUS INSTANCE

Many bioinformatics platforms are emerging within the constantly evolving life sciences field that satisfy most integration and interoperability requirements. The main advantage of this evolution is that developers do not need to rebuild entire knowledge systems and data infrastructures from scratch. It is possible to reuse and recombine existing components to form entirely new software systems as an answer to the latest challenges. Furthermore, with the COEUS framework in place, we have the tools required to launch a new semantic web application with minimal effort.

Continuing our research within the individualized healthcare field, we tackle the study of rare diseases with the development of a new version for the Diseasecard platform. The personal health implications behind rare diseases are seldom considered in widespread medical care. The low incidence rate and complex treatment process makes rare disease research an underrated field in the life sciences. Diseasecard, an online portal containing thousands of pointers to rare disease resources, was developed to aid rare disease investigators. However, the uncontrollable evolution of data and services in the field, united with an aging legacy code, triggers the need for a new release.

Not only was Diseasecard's server-side code dated, the user interface was also in the need of a facelift to better suit the current generation of web applications. Hence, Diseasecard appeared as the perfect benchmark for COEUS' developments. An initial prototype of the new Diseasecard, COEUS first public instance, is available at http://bioinformatics.ua.pt/dc4/.

In this chapter we introduce the new Diseasecard platform and discuss the details behind its development. Starting with a brief analysis of the legacy Diseasecard portal, we cover the easy process of creating a new COEUS seed, from the construction of a rare disease knowledge base to the details of Diseasecard's new user interface.

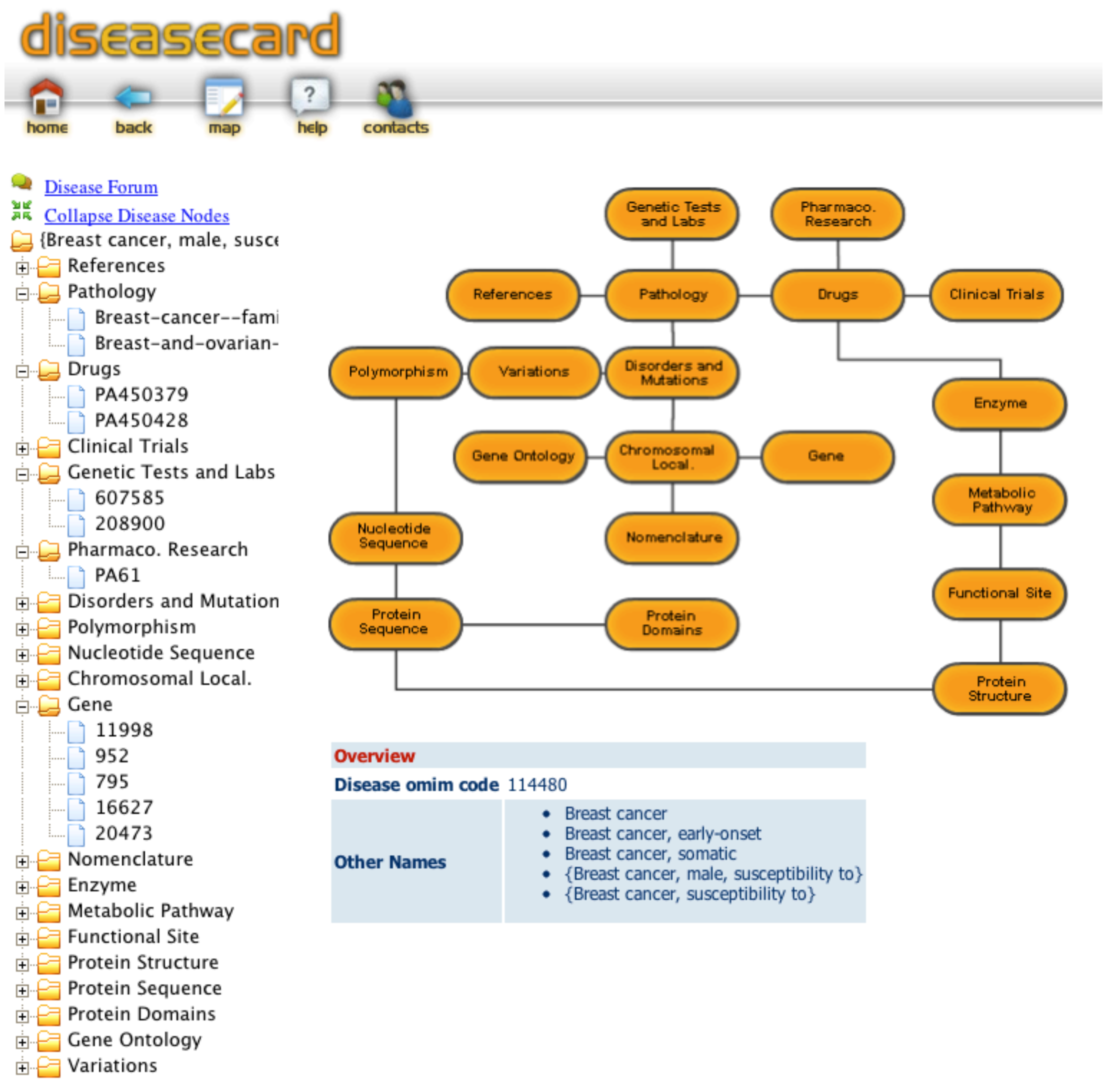
Improving Rare Diseases Research

Rare diseases' particular conditions hold the strongest relations between genotypes and phenotypes. Understanding gene-disease associations is a fundamental goal for bioinformatics research, especially at rare disease level, where the genotype-phenotype connections are limited to a small set of genes. Rare diseases are particular conditions that affect at most 1 in 2000 patients. The European Organization for Rare Diseases (EURORDIS) estimates that there are approximately 6000 to 8000 rare diseases, affecting about 6% to 8% of the population. Within these, about 80% are caused by genetic disorders. Due to the reduced incidence of each individual disease, it is difficult for patients to find support, both at clinical and psychological levels. Some of these chronic diseases hinder the patients' quality of life and cause serious damage or disability in social terms. The existence of a small number of patients for each rare disease also delays the creation of adequate research studies, as it is difficult to identify and coordinate a relevant cohort. Despite the low statistic impact regarding these diseases, the combined amount of patients suffering from one of these rare diseases is considerably high.

Diseasecard's Legacy

Diseasecard was developed to improve rare disease research and education. It is a web portal that uses link integration strategies to establish connections to a myriad of external resources. The goal is to provide a central workspace where users can explore available connections to assess rare diseases underlying genotype, associated proteins or pathways, known drugs, ongoing clinical trials or relevant literature (Figure 0-1).

Initial Diseasecard developments date back to 2004. At that stage, the data acquisition strategy relied on web crawling to discover links for the various data types integrated in the database, and static HTML pages contained most of the Internet content. The idea of providing services to access data was not mature enough yet and most data was still published in CSV files or similar text-based tabular formats. Diseasecard's platform uses a link integration engine, pre-configured with a navigation map that teaches the system what links to collect and what links to follow for further crawling. Whilst this strategy worked for the 2004 timeline, it is currently totally inadequate.



Disease Forum
Collapse Disease Nodes
{Breast cancer, male, susc
- References
- Pathology
  - Breast-cancer--fami
  - Breast-and-ovarian-
- Drugs
  - PA450379
  - PA450428
- Clinical Trials
- Genetic Tests and Labs
  - 607585
  - 208900
- Pharmaco. Research
  - PA61
- Disorders and Mutation
- Polymorphism
- Nucleotide Sequence
- Chromosomal Local.
- Gene
  - 11998
  - 952
  - 795
  - 16627
  - 20473
- Nomenclature
- Enzyme
- Metabolic Pathway
- Functional Site
- Protein Structure
- Protein Sequence
- Protein Domains
- Gene Ontology
- Variations

Overview

Disease omim code	114480
Other Names	• Breast cancer • Breast cancer, early-onset • Breast cancer, somatic • {Breast cancer, male, susceptibility to} • {Breast cancer, susceptibility to}

Figure 0-1. Diseasecard workspace for Alzheimer's Disease, OMIM code 104300 (from legacy Diseasecard[24]).

Collecting Rare Disease Information

Much like the human variome scenario, rare disease information is scattered through multiple non-exchangeable data sources. In a sense, Diseasecard development is a common "by the books" service composition problem. With heterogeneous rare disease data fragmented through multiple independent resources, new strategies must be devised to collect it and make it available for other tools.

[24] http://bioinformatics.ua.pt/diseasecard/evaluateCard.do?diseaseid=104300

In a world with personalized medicine and individual healthcare as primary research topics, advanced integration and interoperability tools are essential. The huge amounts of data are meaningless unless they are interconnected with rich relationships. Moreover, data integration in bioinformatics has been mostly focused on genotype data. Nowadays, the goal is to balance the scale. We need to access the increasing quantity of clinical phenotype data and combine it with existing rich genotype resources, empowering a new knowledge reasoning level.

Despite dated, Diseasecard's initial approach already preconceived this much-needed integration from genotypes to phenotypes. However, with the appearance of WAVe we already provide an alternative geared towards genotype data. Hence, the new Diseasecard is much more directed towards phenotype information. This demands establishing a new rare disease relationship network that in spite of being based on the original Diseasecard, further specializes it with another filtering layer.

The New Diseasecard

Developing a new Diseasecard version was an entirely different task from developing a new application from scratch. The complex requirements analysis or data modelling tasks were already executed and documented for the original portal. Likewise, mock-ups were not required for the interface design as the idea was to improve on existing interactions and to make the lower number of changes as possible, always without disrupting the tree-based and map-based navigation metaphors.

Supported by previous requirements' comprehension, we only needed to update Diseasecard's internal data model to fit the COEUS seed configuration. This requires organizing data in the **Entity-Concept-Item** tree structure and defining the integration properties. Whereas the original Diseasecard used a map-based navigation model to crawl for identifiers, parsing HTML content from webpages dynamically, the new COEUS-based integration engine uses web services and databases to load data and generate a similar, yet richer, rare disease knowledge network.

In addition, this rare disease knowledge network is available for reasoning and inference. The new data relationships allow denser knowledge connections, further enabling the success and availability of reasoning features, which may result in deeper rare diseases insights. Through these new connections we can also infer new knowledge. As such, by semantically integrating data from miscellaneous heterogeneous resources, we are empowering the discovery of new direct links from genotypes to phenotypes in rare diseases.

Regarding the web application, we tried to maintain the user interactions already present in the legacy Diseasecard. Using the same metaphor, the new Diseasecard delivers an improved user experience. The navigation tree is smoother and more complete and leaf links trigger the **Live View** feature. The latter promotes accreditation and ownership of original work, loading the external resource directly within Diseasecard's workspace, just like in WAVe.

At last, the legacy Diseasecard included a navigation map. In this map, users could identify which data types were available for each specific disease. In the new Diseasecard, this key static interaction component was replaced by a dynamic identifier map. Besides being a more interactive tool, the new navigation map enables accessing the external resources directly, without additional navigation tree mouse clicks.

Application Setup

Data Model

Following COEUS "reuse instead of rewrite" motto, the new Diseasecard's data model reuses existing schemas internally. Using COEUS seed configuration and taking advantage of existing ontologies and models for internal usage is enough to organize collected data.

For each individual item, such as a UniProt protein or an OMIM disease, we only need to store its identifier. Hence, we can reuse the *identifier* term from the Dublin Core ontology. In Diseasecard, each Item individual has a **dc:identifier** data property, matching a string with the external identifier. COEUS enables reusing any kind of property, liberating our knowledge base from strict data models. Another example is the **rdfs:label** property, obtained from the RDF schema ontology that is used to label each individual object in Diseasecard, whether it is an Entity, an Item or a Resource.

This "reuse instead of rewrite" fits most required properties. Nevertheless, to further enhance user interactions new relationships were required. Diseases may have multiple names and OMIM's internal structure makes distinctions from phenotype and genotype identifiers. To this end, new data and object predicates were created, as listed in Table 0-1. With the set of integrated resources in place and the new model designed, Diseasecard was ready to be launched as a new COEUS seed.

Table 0-1. List of new predicates in Diseasecard ontology.

PREDICATE	RELATIONSHIP	DESCRIPTION
Object Properties		
hasGenotype	Disease to Disease	Connects a Disease phenotype entry with its associated genotypes.
hasPhenotype	Disease to Disease	Connects a Disease genotype entry with its associated phenotypes.
Data Properties		
chromossomalLocation	to String	Chromossomal location information (read from MorbidMap).
genotype	to Boolean	True if Disease is a genotype.
name	to String	Disease name.
omim	to String	Disease OMIM accession number.
phenotype	to Boolean	True if Disease is a phenotype.

A New COEUS Seed

As mentioned, Diseasecard is the first COEUS seed. To launch a new COEUS the initial step is to download or clone COEUS' source code into a new development workspace. Java, an Apache Tomcat server and a MySQL database must be in place to set up the new system. As mentioned in the previous chapter, the configuration involves three files:

- The Diseasecard model is transposed to a new ontology[25], including the new data and object properties. This file can be created and managed using Protégé.
- ***Config.js***, the seed configuration file, contains the details for the new Diseasecard application properties. These basic properties define where

[25] http://bioinformatics.ua.pt/dc4/diseasecard.owl

further configurations, such as the seed ontology, the setup files or MySQL database connections are stored.

```
# Diseasecard application properties file
{
    "config": {
        "name": "Diseasecard",
        "description": "Diseasecard v4",
        "keyprefix":"coeus",
        "version": "4.0",
        "ontology":
    "http://bioinformatics.ua.pt/dc4/diseasecard.owl",
        "setup": "dc4_setup.rdf",
        "sdb":"dc4_sdb.ttl",
        "predicates":"dc4_predicates.csv",
        "built": true,
        "debug": false,
        "environment": "testing"
    }, … }
```

→ The seed setup file, ***dc4_setup.rdf***, includes the internal data structure and resource configurations, defining how to connect to and exploit resources in Diseasecard's network.

Additionally, three files must be updated with knowledge base connection properties: one for **Jena**, a second for **Joseki** and a third for **pubby**. **Jena** and **Joseki** definitions are similar and include the seed's MySQL database connection properties. The third file, for **pubby**, includes the LinkedData configurations such as the system SPARQL endpoint and internal ontology base URIs.

Resource Configuration

COEUS allows collecting data in miscellaneous formats from local or remote data sources. For Diseasecard's seed, we are looking to build a semantically powerful data network. Hence, we need to obtain a huge amount of identifier mappings. These mappings are usually available as CSV or XML files in some sort of FTP file server.

Figure 0-2 shows Diseasecard's data integration graph, detailing how COEUS integration engine moves from one resource to the next. The starting resource uses a custom connector plugin, processing OMIM's morbid and gene maps.

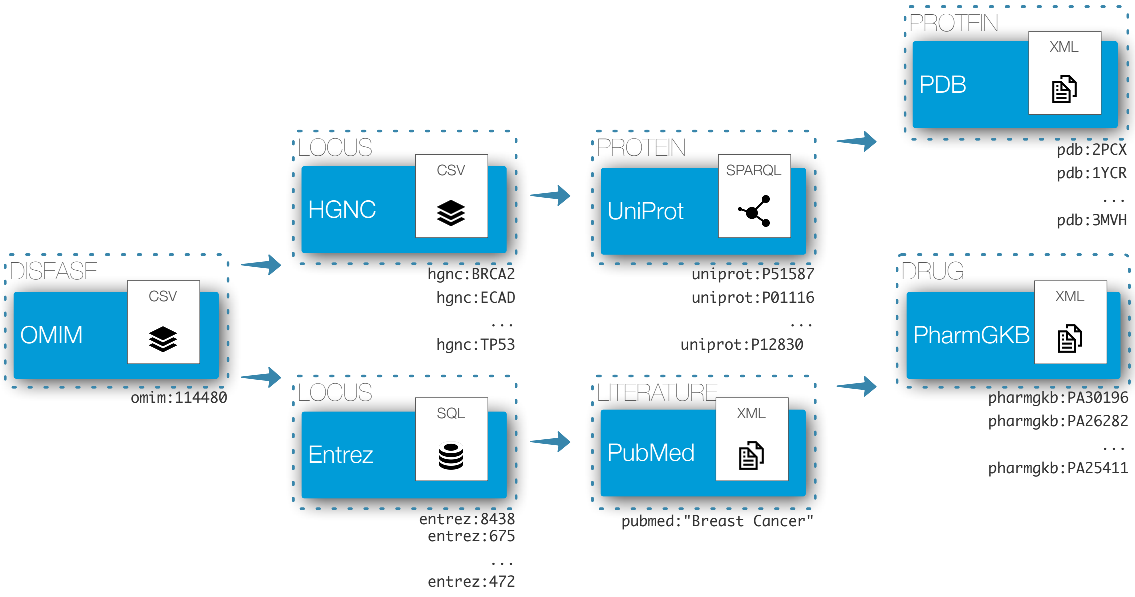


Figure 0-2. Subset of Diseasecard's integration graph. This seed uses COEUS's flexible integration engine to acquire data from heterogeneous and distributed CSV, XML, SQL and SPARQL resources. The integration process generates a rich data network. For example, starting with Breast cancer in OMIM (114480) we obtain multiple genes from HGNC database (BRCA2, TP53…), which are used individually next to obtain a list of UniProt identifiers (P51587, P12830…). From UniProt data we also extract PharmGKB (PA30196, PA26282…) and PDB (2PCX, 1YCR…) identifiers, among others. This process continues until data are fully integrated for all resources in Diseasecard's configuration.

Each resource is configured individually in the local setup file. Once again, Protégé use is advised to build this file, making it fairly easy to edit COEUS setup. In this case, each concept corresponds to an external resource, being it a database or application. The following simplified code snippets highlight the *Protein* **Entity**, the *UniProt* **Concept** and its respective **Resource**.

```
<!-- Protein Entity configuration -->
<owl:NamedIndividual
rdf:about="http://bioinformatics.ua.pt/coeus/entity_Protein">
   <rdf:type rdf:resource="http://bioinformatics.ua.pt/coeus/Entity"/>
   <rdfs:label rdf:datatype="&xsd;string">entity_protein</rdfs:label>
   <dc:title rdf:datatype="&xsd;string">Protein</dc:title>
   <isEntityOf
rdf:resource="http://bioinformatics.ua.pt/coeus/concept_InterPro"/>
   <isEntityOf
rdf:resource="http://bioinformatics.ua.pt/coeus/concept_PDB"/>
   <isEntityOf
rdf:resource="http://bioinformatics.ua.pt/coeus/concept_PROSITE"/>
   <isEntityOf
rdf:resource="http://bioinformatics.ua.pt/coeus/concept_UniProt"/>
   <isIncludedIn
rdf:resource="http://bioinformatics.ua.pt/coeus/seed_Diseasecard4"/>
</owl:NamedIndividual>

<!-- UniProt Concept configuration -->
<owl:NamedIndividual
rdf:about="http://bioinformatics.ua.pt/coeus/concept_UniProt">
```

```
   <rdf:type rdf:resource="http://bioinformatics.ua.pt/coeus/Concept"/>
   <rdfs:label rdf:datatype="&xsd;string">concept_uniprot</rdfs:label>
   <dc:title rdf:datatype="&xsd;string">UniProt</dc:title>
   <hasEntity
rdf:resource="http://bioinformatics.ua.pt/coeus/entity_Protein"/>
   <isExtendedBy
rdf:resource="http://bioinformatics.ua.pt/coeus/resource_DrugBank"/>
   <isExtendedBy
rdf:resource="http://bioinformatics.ua.pt/coeus/resource_InterPro"/>
   <isExtendedBy
rdf:resource="http://bioinformatics.ua.pt/coeus/resource_MeSH"/>
   <isExtendedBy
rdf:resource="http://bioinformatics.ua.pt/coeus/resource_PDB"/>
   <isExtendedBy
rdf:resource="http://bioinformatics.ua.pt/coeus/resource_PROSITE"/>
   <hasResource
rdf:resource="http://bioinformatics.ua.pt/coeus/resource_UniProt"/>
</owl:NamedIndividual>

<!-- UniProt Resource configuration -->
<owl:NamedIndividual
rdf:about="http://bioinformatics.ua.pt/coeus/resource_UniProt">
   <rdf:type rdf:resource="http://bioinformatics.ua.pt/coeus/Resource"/>
   <rdfs:label>resource_uniprot</rdfs:label>
   <order rdf:datatype="&xsd;integer">2</order>
   <dc:title rdf:datatype="&xsd;string">UniProt</dc:title>
   <method rdf:datatype="&xsd;string">cache</method>
   <dc:publisher rdf:datatype="&xsd;string">csv</dc:publisher>
   <endpoint       rdf:datatype="&xsd;string">http://www.genenames.org/cgi-
bin/hgnc_downloads.cgi?title=HGNC+output+data&amp;hgnc_dbtag=on&amp;col=md_
prot_id&amp;status=Approved&amp;status=Entry+Withdrawn&amp;status_opt=2&amp
;level=pri&amp;where=gd_app_sym+LIKE+%27#replace#%27&amp;order_by=gd_app_sy
m_sort&amp;limit=&amp;format=text&amp;submit=submit&amp;.cgifields=&amp;.cg
ifields=level&amp;.cgifields=chr&amp;.cgifields=status&amp;.cgifields=hgnc_
dbtag</endpoint>
    <extends
rdf:resource="http://bioinformatics.ua.pt/coeus/concept_HGNC"/>
    <isResourceOf
rdf:resource="http://bioinformatics.ua.pt/coeus/concept_UniProt"/>
    <hasKey
rdf:resource="http://bioinformatics.ua.pt/coeus/csv_UniProt_id"/>
    <loadsFrom
rdf:resource="http://bioinformatics.ua.pt/coeus/csv_UniProt_id"/>
</owl:NamedIndividual>
```

Once all resources are configured correctly, the actual integration process starts, populating Diseasecard's knowledge base. This is envisaged as a spiralled iterative process, where each iteration fine-tunes the previous one.

From OMIM's MorbidMap to 5 Million Triples

Diseasecard adopts a targeted warehousing strategy. This means that data are integrated once and stays static until the following build process. Accordingly, the data import and translation process gathers all data from external resources in a single centralized Diseasecard knowledge base. In Diseasecard, this process starts with a custom *connector* plugin to process OMIM’s data, traversing the dependency graph for all configured resources iteratively.

During this data import process triples are generated from external data. Adding a new semantic layer on top of existing data results in an augmented dataset. COEUS adds several metadata relationships to each item along with the configured resource properties. Moreover, connections are established from items to concepts, from concepts to items and amongst items. These rich relationships are what make semantic knowledge bases so powerful. Whereas in a CSV file we have a set of columns with text, with the move to a semantic environment all data are interconnected, generating a richer dataset. The same is true for SQL databases, where foreign key relationships or table/column names are mapped to new properties, resulting in more metadata, more relationships and more triples.

OMIM’s morbid map has around 5600 entries related to a gene map with about 12800 entries. From these maps, the graph proceeds to link multiple entities and concepts, increasing the amount of data exponentially. The current Diseasecard build accounts for almost 5 million triples. Leveraging on the big data network and the additional metadata, this number grows constantly as each new resource is integrated. Despite the 5 million triples, the knowledge base only stores around 1.5 million distinct individuals. This is further proof that collected data are deeply intertwined, resulting in a very dense graph.

Building Diseasecard’s knowledge base highlighted some issues with COEUS building process. The performance is severely hindered by the repetitive connections to external web services or by complex SQL queries. Executing the build process as a single task in a single thread takes to long to be acceptable. This fostered the development of a basic multithreaded integration solution. The multithreaded strategy involves processing the various resources at different levels based on the configured dependency graph - Figure 0-3. Diseasecard’s multithread integration solution was promptly integrated within COEUS. Nevertheless, foreseen developments will focus on improving the code implementation for this feature.

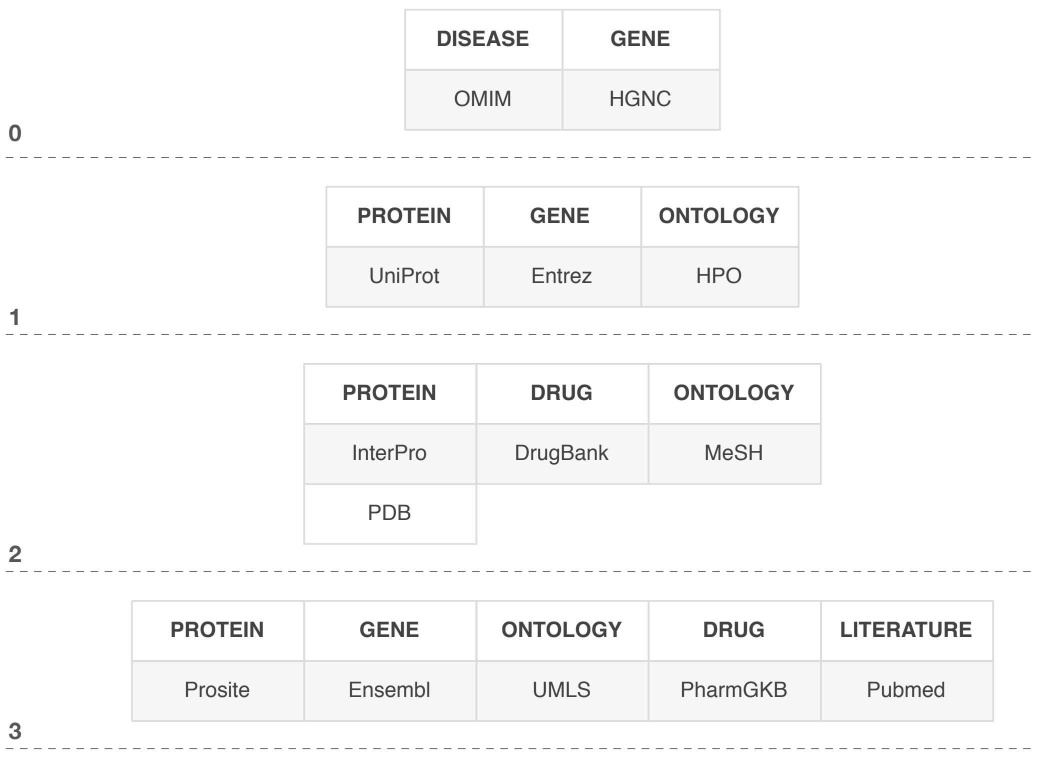
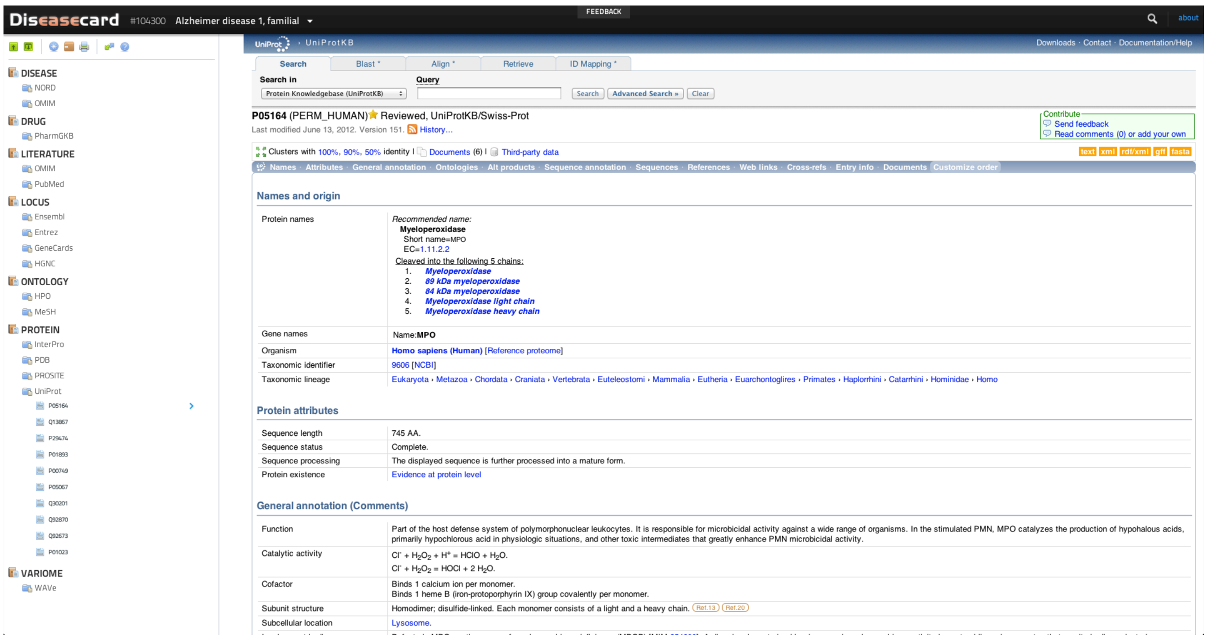


Figure 0-3. Diseasecard build process levels. The multiple stages use a multithreaded approach to load data from external resources, significantly improving the process efficiency and performance.

Building the New Diseasecard's User Interface

With Diseasecard's triplestore populated, interoperability services are enabled. This means that the default API methods (Java, SPARQL, REST, LinkedData, JavaScript) are ready for us. Miscellaneous services were created to access and retrieve data required by the client-side application (Figure 0-4).

Modern web applications employ new user interaction approaches that require flexible server-side code and intelligent client-side code. On the server-side, the application controller must offer easy access points to all data and, if possible, in custom formats ready for use in the web application. The client-side should handle most of the payload for processing data. This does not mean that browsers will perform intensive data processing or transformation activities, but should rely more on asynchronous data exchanges.



P05164 (PERM_HUMAN) Reviewed, UniProtKB/Swiss-Prot
Last modified June 13, 2012. Version 151. History...

Clusters with 100%, 90%, 50% identity | Documents (6) | Third-party data

Names and origin

Protein names	*Recommended name:* **Myeloperoxidase** Short name=MPO EC=1.11.2.2 Cleaved into the following 5 chains: 1. ***Myeloperoxidase*** 2. ***89 kDa myeloperoxidase*** 3. ***84 kDa myeloperoxidase*** 4. ***Myeloperoxidase light chain*** 5. ***Myeloperoxidase heavy chain***
Gene names	Name:**MPO**
Organism	**Homo sapiens (Human)** [Reference proteome]
Taxonomic identifier	9606 [NCBI]
Taxonomic lineage	Eukaryota › Metazoa › Chordata › Craniata › Vertebrata › Euteleostomi › Mammalia › Eutheria › Euarchontoglires › Primates › Haplorrhini › Catarrhini › Hominidae › Homo

Protein attributes

Sequence length	745 AA.
Sequence status	Complete.
Sequence processing	The displayed sequence is further processed into a mature form.
Protein existence	Evidence at protein level

General annotation (Comments)

Function	Part of the host defense system of polymorphonuclear leukocytes. It is responsible for microbicidal activity against a wide range of organisms. In the stimulated PMN, MPO catalyzes the production of hypohalous acids, primarily hypochlorous acid in physiologic situations, and other toxic intermediates that greatly enhance PMN microbicidal activity.
Catalytic activity	$Cl^- + H_2O_2 + H^+ = HClO + H_2O$. $Cl^- + H_2O_2 = HOCl + 2 H_2O$.
Cofactor	Binds 1 calcium ion per monomer. Binds 1 heme B (iron-protoporphyrin IX) group covalently per monomer.
Subunit structure	Homodimer; disulfide-linked. Each monomer consists of a light and a heavy chain. Ref.13 Ref.20
Subcellular location	Lysosome.

Figure 0-4. The new Diseasecard workspace for Alzheimer's Disease, OMIM code 104300[26].

Diseasecard implementation uses COEUS API to apply these modern data access paradigms. Relying on internal Java methods, Diseasecard includes multiple actions to mediate data access from the client application to the seed's knowledge base. For instance, a method for retrieving a data network associated with a single OMIM code (104300, Alzheimer's Disease) is available at http://bioinformatics.ua.pt/dc4/content/104300.js. This returns a JSON object with the disease information that is used to generate, on the browser, both the sidebar navigation tree and central navigation map. Similarly, data requests to the REST *triple* service are used to load disease synonyms (http://bioinformatics.ua.pt/dc4/api/triple/coeus:omim_104300/dc:description/obj/js).

These interactions use COEUS API and enable a faster web application, with smaller data requests and improved responsiveness. Comparing the sidebar navigation tree in the legacy Diseasecard with the new one, users had to wait for the entire page to be processed on the server and then sent to the browser before the webpage actually appears. In the new version, the page loads completely and provides adequate feedback to users while the navigation tree and map are being loaded.

[26] http://bioinformatics.ua.pt/dc4/disease/104300

Another welcome addition is the inclusion of an easier bookmarking tool. In the current version, when **Live View** is triggered, the page URL is updated in the browser, enabling the creation of bookmarks pointing directly to an external resource within a disease context in Diseasecard (similar to WAVe's **UniversalAccess**).

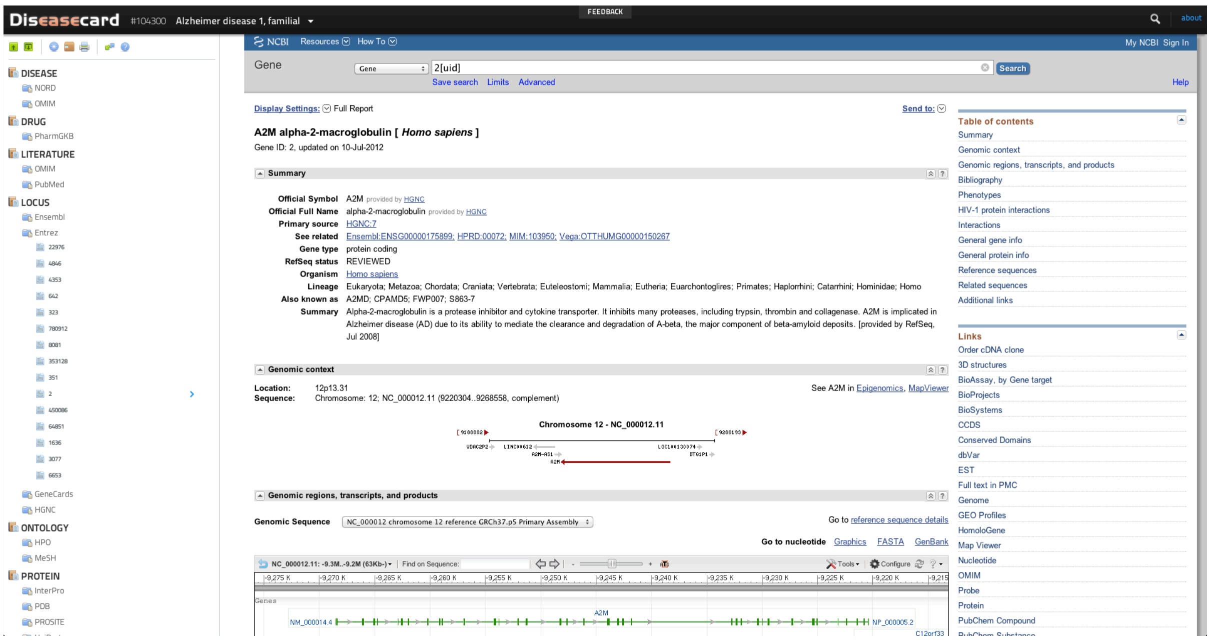
With a re-engineered server side it was also essential to revamp Diseasecard's interface. With a sleeker design, the new Diseasecard has improved usability and delivers a more fulfilling experience to end-users.

Features and Usability

Context-based Navigation

Diseasecard is a unique alternative for browsing biomedical rare disease information in a centralized environment. The context-based navigation approach enables exploring a variety of resources associated to a single disease and also browsing disease synonyms. With these two complimentary perspectives all significantly relevant resources associated to one or more diseases are a couple clicks away.

The workspace includes two disease data network navigation alternatives. The left sidebar includes a tree to quickly access all links with a familiar metaphor. The central area displays a circular navigation map, pointing to all individual identifiers. This map is an outstanding improvement from what was previously available, making it one of Diseasecard's key features. Both the navigation tree and map trigger the **Live View** feature (Figure 0-5).


A2M alpha-2-macroglobulin [*Homo sapiens*]

Gene ID: 2, updated on 10-Jul-2012

Summary

Official Symbol	A2M provided by HGNC
Official Full Name	alpha-2-macroglobulin provided by HGNC
Primary source	HGNC:7
See related	Ensembl:ENSG00000175899; HPRD:00072; MIM:103950; Vega:OTTHUMG00000150267
Gene type	protein coding
RefSeq status	REVIEWED
Organism	Homo sapiens
Lineage	Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia; Eutheria; Euarchontoglires; Primates; Haplorrhini; Catarrhini; Hominidae; Homo
Also known as	A2MD; CPAMD5; FWP007; S863-7
Summary	Alpha-2-macroglobulin is a protease inhibitor and cytokine transporter. It inhibits many proteases, including trypsin, thrombin and collagenase. A2M is implicated in Alzheimer disease (AD) due to its ability to mediate the clearance and degradation of A-beta, the major component of beta-amyloid deposits. [provided by RefSeq, Jul 2008]

Genomic context

Location: 12p13.31

Sequence: Chromosome: 12; NC_000012.11 (9220304..9268558, complement)

See A2M in Epigenomics, MapViewer

Genomic regions, transcripts, and products

Genomic Sequence: NC_000012 chromosome 12 reference GRCh37.p5 Primary Assembly

Go to reference sequence details

Go to nucleotide Graphics FASTA GenBank

Figure 0-5. Diseasecard's workspace for Alzheimer's Disease, OMIM code 104300, highlighting Entrez Gene entry A2M for *Homo sapiens*[27] in LiveView**.**

This approach was used initially in the legacy Diseasecard and was enhanced for WAVe. The newest Diseasecard version further improves WAVe's approach, making **Live View** more interactive, responsive and usable.

Resources Relationship Graph

To correctly explore Diseasecard's huge amount of data and relationships we could not rely on a static navigation system or a non-scalable navigation tree. This rich rare disease resource relationship graph provides a unique wealth of direct and indirect connections. Hence, a suitable approach for displaying these relationships was required. Our choice set on the JavaScript InfoVis Toolkit[28] framework (JIT). This framework combines the power of client-side data handling with a collection of visualization approaches based on JavaScript JSON objects and manipulations on the DOM canvas. Figure 0-6 shows Diseasecard using JIT to expose a disease map to users in a simple aesthetically pleasing way.

[27] http://bioinformatics.ua.pt/dc4/disease/104300#entrez:2

[28] http://thejit.org/

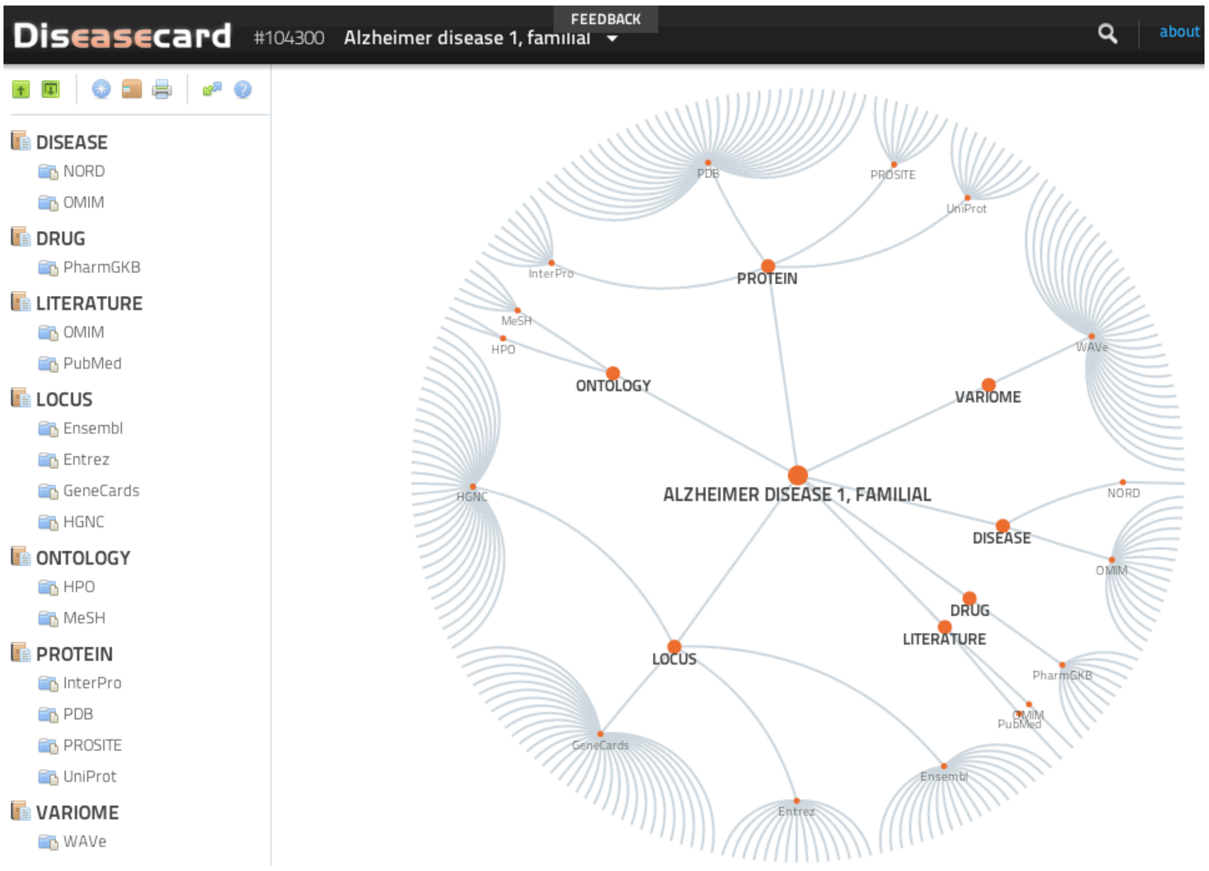


Figure 0-6. Diseasecard's entry navigation graph for Alzheimer's Disease, OMIM code 104300. A circular navigation map was created, using the JIT visualization library, to facilitate the access to the huge amount of linked resources.

The navigation map starts with the selected disease and connections to the set of entities in the knowledge base. Clicking each entity name, *Protein* for example, centres the map on the *Protein* node, highlighting its connections to its various internal concepts. Likewise, clicking on a concept, *UniProt* for instance, centres the node and shows links to each individual concept Item.

Rich Data

Selecting the adequate set of resources for phenotype-oriented information was a crucial step towards the new Diseasecard. As expected, each resource features its own domain, architecture and interface standards, i.e., resources are heterogeneous and distributed. Table 7-2 list the resources and pointers integrated in the new Diseasecard.

Table 0-2. Diseasecard integrated resources.

NODE	RESOURCE	DESCRIPTION
Disease	NORD	http://www.rarediseases.org/

NODE	RESOURCE	DESCRIPTION
	OMIM	http://www.omim.org/
Drug	PharmGKB	http://www.pharmgkb.org/
Literature	Pubmed	http://www.ncbi.nlm.nih.gov/pubmed/
Locus	Ensembl	http://www.ensembl.org/
	Entrez	http://www.ncbi.nlm.nih.gov/gene/
	GeneCards	http://www.genecards.org/
	HGNC	http://www.genenames.org/
Ontology	GO	http://amigo.geneontology.org/
Protein	UniProt/SwissProt	http://www.uniprot.org/
	UniProt/TrEMBL	http://www.uniprot.org/
	PDB	http://www.pdb.org/
	Expasy	http://expasy.org/
	InterPro	http://www.ebi.ac.uk/interpro/
Variome	WAVe	http://bioinformatics.ua.pt/WAVe

One of COEUS major features is the prompt availability of interoperability services. In Diseasecard, the services required for accessing data are enabled by default.

The wealth of data collected during the data integration process is available through REST services, a SPARQL endpoint and a LinkedData view. Furthermore, the REST services and SPARQL endpoint are already used within Diseasecard client-side application.

Considering the constructed knowledge base a single platform without Diseasecard's web application, it is, *per se,* a single unique resource for the rare disease community. Life sciences developers can exploit this data collection to build or extend existing applications.

From a modern application perspective, Diseasecard can also be seen as platform with multiple applications. While for now only a web information system is available, COEUS robustness permits deploying applications targeted at the desktop or mobile devices, using the same set of APIs and accessing the original knowledge base.

The rare diseases dataset build using COEUS flexible integration engine results in a strikingly rich semantic knowledge base. This opens the room for further exploratory endeavours, namely using reasoning and inference. Whilst these features are not yet a part of the new Diseasecard, they will be made available through innovative user interactions in Diseasecard's web workspace.

Discussion

A Suitable Software Infrastructure for each Bioinformatician

As the miscellaneous "omics" fields branch new domains and research specializations, the technological needs for each field revolve around a common set of problems. Managing data, accessing and integrating information from other laboratories, or providing recently discovered knowledge to others are essential steps in the path of making science.

While it is nearly impossible to satisfy the requirements from all life sciences research fields, we can promote the use of technologies and tools that facilitate accomplishing all of the project's software-related goals. In a broad sense, this is our main objective with the COEUS framework.

The COEUS platform is a powerful development environment. Its scalability and flexibility make it ideal for highly heterogeneous scenarios and apt for the challenging requirements associated with particular "omics" fields. In fact, COEUS targets the improvement of niche fields, empowering amateur and professional developers with the tools to quickly model, integrate and publish data. Furthermore, the semantic web ideals span through all framework's components, from the integration of data to the interoperability with other tools. This provides limitless resource integration architectures with advanced data exploration features. With the former we are able to triplify almost all existing data into a new richer knowledge base and, complementarily, with the latter, we are able to access collected data by multiple means. This enables the creation of custom application ecosystems with comprehensive architectures. Bioinformaticians can program interfaces in any language and easily target web, desktop or mobile environments. Furthermore, by publishing data through the SPARQL and LinkedData interfaces, new systems will be part of the global web of knowledge. With the widespread use of COEUS and other similar tools, we expect that each bioinformatician can create its own technological infrastructure that goes beyond the boundaries of project-specific internal use.

Whereas the strategies adopted in the EU-ADR Web Platform and WAVe consisted in the adoption of traditional relational databases, COEUS empowers the creation of semantic knowledge bases. This enables a whole new level of knowledge exploration through the aforementioned SPARQL and LinkedData interfaces that allow the creation of complex knowledge reasoning and inference features. Diseasecard's dataset has innate semantics; collected data has an undisclosed meaning that can be explored to obtain new vital connections between genes and diseases. Likewise, COEUS brings this semantic web layer, and its underlying semantic features, to all bioinformaticians in any life sciences field.

Applying COEUS to the Rare Diseases Research Field

Research on rare diseases is of growing importance in the last couple of years. Uncovering the underlying genetic causes of rare diseases is the first step towards a better comprehension of our health, making us one step closer of the individualized healthcare panacea. Moreover, the funding and interest in large-scale rare disease projects has been renewed, namely within the European Union.

Since 2004, Diseasecard has been contributing to this research field by providing a portal with information regarding rare diseases and connections to a myriad of resources contextualized to each disease. Despite its quality, the legacy Diseasecard is an out-dated system, with an architecture that is no longer efficient for the current bioinformatics landscape. With COEUS, we have the opportunity to overcome the original Diseasecard's caveats, deploying a richer application, with a reengineered architecture and re-designed user interface. As previously mentioned, the combination of rapid application development with biomedical semantics is a key enabler of the next-generation of bioinformatics and the new Diseasecard is the first step towards this bright future.

The new Diseasecard, powered by COEUS, improves on the legacy version in three distinct aspects, discussed next.

→ With a **semantic knowledge base** supporting the application, collected and connected data are richer and more meaningful. Whilst these capabilities are not yet fully explored, the open possibilities are immense. Establishing new relationships between OMIM's disease data and multiple ontologies or external resources generates a comprehensive rare disease dataset, enabling the inference of unique connections that would not be possible to obtain otherwise.

→ Data in the knowledge base are now **interoperable** using COEUS default API. REST services, a SPARQL endpoint and a LinkedData interface are available for others to explore the dense rare disease data graph compiled in the new Diseasecard.

→ The new Diseasecard **web workspace** is a significant improvement over the legacy version. Disease synonyms, the navigation tree and the new navigation graph present a more interactive and usable interface.

Diseasecard is the first COEUS instance and represents our initial endeavour towards the future of web-based biomedical applications. The new Diseasecard is available online at http://bioinformatics.ua.pt/dc4/.