RESEARCH

Integration of Structured Biological Data Sources using Biological Expression Language

Charles Tapley Hoyt[1,2*], Daniel Domingo-Fernández[1,2], Sarah Mubeen[1,2], Josep Marín-Lláo[1], Andrej Konotopez[1], Christian Ebeling[1], Colin Birkenbihl[1,2], Özlem Muslu[1,2], Bradley English[1,2], Simon Müller[1,2], Mauricio Pio de Lacerda[2], Mehdi Ali[3,4], Scott Colby[5], Denés Türei[6,7,8], Nicolàs Palacio-Escat[7,8] and Martin Hofmann-Apitius[1,2]

*Correspondence: charles.hoyt@scai.fraunhofer.de
[1] Department of Bioinformatics, Fraunhofer Institute for Algorithms and Scientific Computing (SCAI), Konrad Adenauer Strasse, 53754 Sankt Augustin, Germany
Full list of author information is available at the end of the article

Abstract

Background: The integration of heterogeneous, multi-scale, and multi-modal knowledge and data has become a common prerequisite for joint analysis to unravel the mechanisms and aetiologies of complex diseases. Because of its unique ability to capture this variety, Biological Expression Language (BEL) is well suited to be further used as a platform for semantic integration and harmonization in networks and systems biology.

Results: We have developed numerous independent packages capable of downloading, structuring, and serializing various biological data sources to BEL. Each Bio2BEL package is implemented in the Python programming language and distributed through GitHub (`https://github.com/bio2bel`) and PyPI.

Conclusions: The philosophy of Bio2BEL encourages reproducibility, accessibility, and democratization of biological databases. We present several applications of Bio2BEL packages including their ability to support the curation of pathway mappings, integration of pathway databases, and machine learning applications.

Keywords: Data Integration; Semantic Web; Biological Expression Language; Knowledge graphs

Background

The integration of heterogeneous, multi-scale, and multi-modal biomedical data has become a cornerstone of modern computational investigation of the mechanisms and aetiologies underlying complex diseases (Iyappan et al., 2014; van Dam et al. 2014; Wanichthanarak et al., 2015; Himmelstein et al., 2017; Fan et al., 2019). An overarching strategy was proposed by Davidson *et al.* more than two decades ago that outlined the transformation of data into a common model, semantic alignment of related objects, integration of schemata, and federation of data (Davidson et al., 1995). However, integration remains a challenging task that requires the identification and deep understanding of biological data sources and their respective formats, conversion, harmonization, and unification.

Initial interest in the semantic web and linked open data along with the adoption of RDF (Resource Description Framework) in the biomedical community led to the Bio2RDF project, in which pipelines for converting and serializing several biological data sources to RDF were developed (Belleau et al., 2008). Several updates have been issued since its deployment such as the inclusion of chemical information systems (Chen et al., 2010). Further, it has also influenced in and has been

adopted by subsequent projects such as Open PHACTS (Williams et al., 2012). While RDF is highly expressive and each of these projects have developed and enforced well-defined schemata, the format is often not well-suited for downstream analyses and must first be queried with languages like SPARQL (SPARQL Query Language for RDF) and subsequently be transformed into appropriate formats with general-purpose programming languages. Alternatives to RDF/SPARQL such as property graphs (e.g., Neo4j, OrientDB) are comparable (Alocci et al., 2015) but also necessitate similar post-processing.

Conversely, there have been several biologically meaningful integration efforts (e.g., STRING; Warde-Farley, et al. 2010, GeneMANIA; Szklarczyk et al., 2015, GeneCards; Stelzer et al., 2016). However, most suffer from a lack of defined schemata or standardized data format that impede biological database interoperability. As interoperability itself is a multifaceted concept, we would like to highlight three of its facets: first, data sources should refer to named entities using high-quality, publicly accessible terminologies as prescribed by the Minimal Information Requested in the Annotation of Biochemical Models standard (Laibe and Le Novère, 2007). Second, data sources should additionally denote the ontological classes of named entities (e.g., gene, transcript, protein, pathway, disease) along with their reference using controlled vocabularies such as the Systems Biology Ontology (Courtot et al., 2011). Some identifiers, such as those for genes, are often used to refer not only to the physical region of DNA within the genome, but also the corresponding RNA transcript(s) or protein product(s). Unfortunately, many biological databases do not explicitly distinguish between these entity classes. For example, the STRING database lists gene-centric homology relationships, transcript-centric co-expression relationships, and protein-centric protein-protein interactions using gene-centric nomenclature. While it may be possible to identify the classes of entities based on their incident relationships, doing so requires specific knowledge of the database including the semantics of its relationships. Third, resources should, at a minimum, map their relationships to controlled vocabularies such as the Relation Ontology, or further use standardized data formats with defined semantics (e.g., PSI MI-TAB) to minimize both the interpretation and implementation effort when combining them with other resources.

OmniPath (Türei et al., 2016) began to address these facets when it combined several signaling pathway and transcriptional regulation databases. It achieved interoperability between several databases by normalizing the identifiers and relationships between entities from several databases describing the same phenomena (e.g., microRNA-target interactions, protein-protein interactions, etc.) and creating a unified network. However, because it did not use a standard format or schema as mentioned in the third facet for interoperability, OmniPath itself cannot readily be directly integrated with other biological data sources. Pathway Commons (Cerami et al., 2011) addressed this concern when combining several molecular pathway and interaction databases by translating the source databases into the BioPAX standard (Demir et al., 2010) using automated pipelines. However, it suffers from low granularity and low recovery of information from some of its primary biological data sources which may be due to prioritization of software development time, data usage restrictions, or shortcomings in the BioPAX standard. While BioPAX is well-suited

for representing biological reactions and transformations, it is limited in its ability to represent correlative and associative relationships across multi-scale biology (e.g., at the levels of processes, phenotypes, and clinical observations).

As an alternative, we propose the use of Biological Expression Language (BEL; Slater, 2014) as an integration schema in order to overcome the limits faced by previous efforts and to simultaneously address all three facets of interoperability. BEL has begun to prove itself as a robust format in the curation and integration of previously isolated biological data sources of high granular information on genetic variation (Naz et al., 2016), epigenetics (Irin et al., 2015), chemogenomics (Emon et al., 2017), and clinical biomarkers (Iyappan et al., 2017). Its syntax and semantics are also appropriate for representing, for example, disease-disease similarities, disease-protein associations, chemical space networks, genome-wide association studies, and phenome-wide association studies.

With the same focus on reproducibility as Bio2RDF, OmniPath, and Pathway Commons as well as deference to software maintainability and the ease of development and inclusion of new biological data sources, we have developed a growing list of Bio2BEL packages, each capable of downloading, structuring, and serializing various biological data sources to BEL (Table 2). Each can be found in the Bio2BEL GitHub organization (`https://github.com/bio2bel`) as an independent open-source Python package that can readily be installed with pip. We have also developed and freely provided a framework (`https://github.com/bio2bel/bio2bel`) in the Python programming language to enable code reuse and the fast generation of additional Bio2BEL packages. Notably, the list of Bio2BEL packages includes one for OmniPath as a proof of concept that authors of other resources can implement their own Bio2BEL packages. In this article, we present the philosophy and implementation of Bio2BEL packages, a summary of past and future Bio2BEL packages, and finally, several case studies including the utility of Bio2BEL packages during curation of pathway mappings, in the analysis of cancer genome data, and for machine learning applications.

Implementation

Bio2BEL comprises numerous independent open-source Python packages that each enable reproducible access to a given biological data source (Figure 1). Each Bio2BEL package contains five components: 1) a definition of the source database or knowledge base, 2) an automated downloader for the data, 3) a parser for the data, 4) a storage and querying system for the data, and 5) a protocol for serializing the data to BEL (Figure 2). In this section, we outline the components of a Bio2BEL package and their implementation details.

Components of a Bio2BEL package

As this section outlines the core components and philosophy of a Bio2BEL package, it illustrates the tasks and thought process of a scientific software developer as they implement a new Bio2BEL package.

Definition of data

The first step in generating a Bio2BEL package is to understand the source data. This requires determining if the data are publicly accessible, if they are versioned

(and how the location changes with versions), and if they are available under a permissive license. Bio2BEL packages do not contain data themselves and only refer to the locations of the original data sources. For those that are versioned, providers commonly generate symlinks to the most recent version (e.g., InterPro; `ftp://ftp.ebi.ac.uk/pub/databases/interpro`). These characteristics help minimize licensing issues while enabling the resulting packages to update their content without changing code. Then, the developer implements custom code that makes the appropriate interpretations to convert the source data to BEL. Below, three types of data that can be readily integrated in BEL are described along with accompanying Table 1.

Taxonomies, hierarchies, and ontologies The Medical Subject Headings (MeSH; Rogers, 1963) multi-hierarchy can be converted to BEL by generating an isA relationship between each MeSH descriptor and all of its corresponding parents in the associated MeSH tree. Nomenclatures like the Complex Portal (Meldal et al., (2015) also define partOf relations between protein complexes and their substituents. The multi-hierarchy in Gene Ontology (GO; Carbon et al., 2017) can be converted similarly, which contains both isA relations and partOf relations.

Tabular and Relational Data Enzyme inhibitors from ChEMBL and PubChem can be encoded like a(X) directlyDecreases act(p(Y), ma(kin)), and disease-specific differential gene expression can be encoded like path(X) positiveCorrelation r(Y) or path(X) negativeCorrelation r(Y), or path(X) causeNoChange r(Y) depending on whether the gene's expression is up-regulated, down-regulated, or not regulated, respectively. Further, BEL relationships can be extended include metadata (i.e., annotations) describing their quantitative aspects. For example, IC50, EC50, or other kinetic assay measurements as well as provenance and biological contextual information (e.g., original publication, cell line, assay type) can be included with the enzyme inhibition relationships from ChEMBL. Similarly, the log2 fold change and p-values can be included with relationships about differential gene expression.

Graphs Wet-laboratory experimentation can be used to generate networks of directly observed phenomena (e.g., protein-protein interaction networks) and indirectly observed phenomena (e.g., gene co-expression networks). Graphs are often distributed as tabular data to include additional information about their constituent nodes and edges and there is often overlap with the previous data type describing tabular and relational data. In silico experimentation can also be used to derive edges from experimental data sets or even other graphs. For instance, bipartite graphs can be projected to homogeneous graphs consisting of a single entity and edge type as suggested by Sun et al. (2014). Menche et al. (2015) used this strategy and computed a homogenous graph of disease-disease associations from a bipartite graph of diseases and their associated genes.

Downloader

The Bio2BEL framework follows a functional programming paradigm to provide an abstraction of the acquisition of data over common internet protocols like HTTP,

HTTPS, and FTP. With only the URL of the data set as an input, Bio2BEL generates a download function that wraps Python's built-in urllib module and a simple caching mechanism in the local filesystem that avoids unnecessary network usage and duplication of potentially large files. However, some data sources, such as DrugBank (Wishart et al., 2018), are not available without authentication and cannot make use of this abstraction. In those cases, developers can substitute the standard code provided in the Bio2BEL framework with custom implementations. We have taken this route for several of the packages presented in the Results section of this paper for repositories including DrugBank and MSigDB (Liberzon et al., 2015).

Parser

There are several common file formats used by biological data sources (e.g., CSV, TSV, XML, RDF, JSON, KGML, Stockholm, OBO, OWL). Data may also (and sometimes only) be accessible through public application programming interfaces (APIs) such as the data from KEGG (Kanehisa et al., 2017), Reactome (Fabregat et al., 2018), and BioThings (Xin et al., 2016). Alternatively, data may be available through software packages usage such the Affymetrix R package (Gautier et al., 2004) and HaploReg (Ward and Kellis, 2012). After each Bio2BEL package's downloader generates a local copy of the data, the developer can either use one of the pre-defined parser functions from the Bio2BEL framework or implement a custom parser. For the most simple formats (i.e., CSV and TSV), the Bio2BEL framework automatically generates a parser that uses the pandas package (McKinney, 2010; https://github.com/pandas-dev/pandas). Formats like XML, JSON, and Stockholm have corresponding parsers built into the Python language or standard biology-focused packages, but the information contained within often needs custom logic for restructuring such as in the case of KGML, BioPAX, or PSI MI-XML. The remaining custom formats all require custom parsers and logic. We have already implemented Bio2BEL that used CSV and TSV data (e.g., InterPro, ExCAPE-DB), XML (e.g., DrugBank), RDF (e.g., WikiPathways), JSON and KGML (e.g., KEGG), Stockholm (e.g., miRBase), and OBO and OWL (e.g., GO, DOID).

In the case of tabular data, the developer has the opportunity to annotate the column headers and their corresponding data types, which are not always included in the data and may be sought from various readme files or by exploring the corresponding website. Further, the contained data might be more useful after normalization or augmentation with information from other biological data sources. Because some databases provide identifiers with redundant information, such as the duplication of the namespace in the identifier, they must be normalized. For example, each identifier in the Disease Ontology (Schriml et al., 2018) is prefixed by its namespace, DOID, as can be seen in the Compact URI for the entry for restless legs syndrome, DOID:DOID:0050425. In the corresponding Bio2BEL DOID package, as well as those for others (e.g., HGNC, Gene Ontology) we normalized these identifiers to remove the redundant information. Because the main Entrez Gene database does not contain crucial information for genes, such as their chromosomal coordinates in various genomic builds, we augmented the data in the Bio2BEL Entrez package for each gene with information from RefSeq so that the genomic positions and corresponding genome build for each gene were readily accessible. Additionally, several

databases that reference genes only use their HGNC gene symbols and not stable identifiers, and therefore require this additional normalization step.

Storage
Though this step may be considered optional after parsing the data, it is helpful for future reuse to choose a database type and develop a schema with which the data can be stored. Often, relational databases that can be queried with SQL are an appropriate choice. The Bio2BEL framework provides a full harness for generating an object-relational mapping (ORM) using the SQLAlchemy (https://www.sqlalchemy.org) Python package that handles generation of the SQL schema and storage of the data in a SQL database. Corresponding entity-relation diagrams can be found in the supplementary data repository at https://github.com/bio2bel/bio2bel-manuscript-supplement. While all Bio2BEL packages have, until now, used SQL databases with the SQLAlchemy ORM, there exists alternatives such as graph databases built on RDF or property graphs like Neo4J or OrientDB with a corresponding object-graph mapper that have been successfully employed in downstream applications using biological knowledge graphs (Himmelstein et al., 2017; Saqi et al., 2018).

Serializer
The final aspect of a Bio2BEL package is either to serialize the parsed data as BEL or to export the accompanying database as BEL. Entities in the SQL database that correspond to nodes and edges in BEL graphs can be converted by extending their respective ORM classes with Python functions using the internal domain-specific language provided by PyBEL (Hoyt et al., 2018a). It can then be output in several formats provided by PyBEL and its growing ecosystem of plugins as well as it shields Bio2BEL packages from changes to the BEL language. Additionally, some Bio2BEL packages wrap standard nomenclature resources such as HGNC (Yates et al., 2017) and are able to generate BEL namespace files that are a necessary in both manual and automated curation of content in BEL (Figure 2). This step is deeply connected with the prior step related to the definition of the data.

Implementation Details

The Bio2BEL framework and Bio2BEL packages are implemented in Python with accessibility and readability in mind. The framework provides an abstract class bio2bel.Manager whose functionality all Bio2BEL packages must completely implement. Using these definitions, the framework automatically generates a uniform command line interface (CLI) that includes functions for populating the database, clearing the database, reloading data from the source, generating a web application with a view over the contents of the database, and serializing to BEL.

The Bio2BEL framework and Bio2BEL packages use flake8 (`https://github.com/PyCQA/flake8`) to enforce code quality, a setup.cfg file to describe the package, setuptools (https://github.com/pypa/setuptools) to build distributions, pyroma (https://github.com/regebro/pyroma) to enforce package metadata standards, sphinx (https://github.com/sphinx-doc/sphinx) to build documentation, Read the Docs (https://readthedocs.org) to host documentation, pytest (https://github.com/pytest-dev/pytest) as a testing framework, coverage (https://github.com/nedbat/coveragepy)

and Codecov (https://codecov.io) to monitor testing coverage, and Travis-CI (https://travis-ci.com) as a continuous integration service. Further, we provide a template for Cookiecutter (https://github.com/audreyr/cookiecutter) at https://github.com/bio2bel/bio2bel-cookiecutter such that the structure of new packages can be quickly generated containing all of the configuration for each of these tools.

Implications of the Bio2BEL Philosophy

Because all Bio2BEL packages are uniform in their implementation and CLI usage, it is trivial to provide a Dockerfile and Docker-Compose configuration for quick deployments. In the future, we plan to automatically generate RESTful APIs, which may be more useful to deploy internally than to use publicly available ones due to constraints like rate-limits. Because all Bio2BEL packages are independent, they avoid two major problems of monolithic codebases: they are more robust to breakages or failures in a single package and they can be installed as needed, which is pertinent as the data sources become larger, more heterogeneous, and more complex.

Further, Bio2BEL packages can be generated by any group, and registered with the Bio2BEL framework using Python entry points (`https://packaging.python.org/specifications/entry-points`) that can be defined in the installation configuration. While the Cookiecutter template allows new developers to quickly generate a package with the correct format, a full tutorial for implementing a uniform Bio2BEL package can be found at `https://bio2bel.readthedocs.io/en/latest/tutorial.html`.

Ethics approval and consent to participate
Not applicable

Consent for publication
Not applicable

Availability of data and materials
Each Bio2BEL package is listed on `https://github.com/bio2bel` and automatically acquires relevant data from their respective original biological data sources.

Competing interests
The authors declare that they have no competing interests.

Funding
This work was partially supported by the EU/EFPIA Innovative Medicines Initiative Joint Undertaking under AETIONOMY [grant number 115568], resources of which are composed of financial contribution from the European Union's Seventh Framework Programme (FP7/2007-2013) and EFPIA companies in kind contribution.
This work was also partially supported by the Fraunhofer Society's MAVO program.
The funding bodies did not play a role in the design of the study and collection, analysis, and interpretation of data, nor in writing the manuscript.

Author's contributions
CTH conceived and designed the study. CTH, DDF, and SM drafted the manuscript. MHA acquired funding and reviewed the manuscript. All authors performed data curation and developed computational pipelines for extraction, transformation, and loading of various biological data sources. All authors have read and approved the final manuscript.

Acknowledgements
We would like to thank the curators and maintainers of the several databases we have used, without whom none of this work would be possible.

Author details
[1] Department of Bioinformatics, Fraunhofer Institute for Algorithms and Scientific Computing (SCAI), Konrad Adenauer Strasse, 53754 Sankt Augustin, Germany. [2]Bonn-Aachen International Center for IT, Rheinische Friedrich-Wilhelms-Universität Bonn, 53115 Bonn, Germany. [3] Department of Enterprise Information Systems, Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS), Konrad Adenauer Strasse, 53754 Sankt Augustin, Germany. [4]Department of Computer Science, Rheinische Friedrich-Wilhelms-Universität Bonn, 53115 Bonn, Germany. [5]Department of Chemistry, Stanford University, 94305 Stanford, United States of America. [6]European Molecular Biology Laboratory (EMBL), Structural and Computational Biology Unit, Meyerhofstrasse 1, 69117 Heidelberg, Germany. [7]Faculty of Medicine, Joint Research Centre for Computational Biomedicine, RWTH Aachen University, MTZ Pauwelsstrasse 19, 52074 Aachen, Germany. [8]Faculty of Medicine and Heidelberg University Hospital, Institute of Computational Biomedicine, Heidelberg University, Bioquant Im Neuenheimer Feld 267, 69120 Heidelberg, Germany.

References
Figures

Figure 1 Sample figure title. A short description of the figure content should go here.

Figure 2 Sample figure title. Figure legend text.

Tables

Table 1 Sample table title. This is where the description of the table should go.

	B1	B2	B3
A1	0.1	0.2	0.3
A2	...	..	.
A3	..	.	.

Additional Files
Additional file 1 — Sample additional file title
Additional file descriptions text (including details of how to view the file, if it is in a non-standard format or the file extension). This might refer to a multi-page table or a figure.

Additional file 2 — Sample additional file title
Additional file descriptions text.