Maintaining the Drug Ontology: an Open-source, Structured Product Label API for the JVM

Roger A. Hall*, Josh Hanna, and William R. Hogan
Division of Biomedical Informatics, University of Arkansas for Medical Sciences, Little Rock, Arkansas, USA

ABSTRACT

Our use case for maintenance of the Drug Ontology includes a semi-automated, daily process capable of importing new, relevant information from a variety of linkable resources, using fast and flexible algorithms with full access to all data. Structured Product Labels contain linkable information regarding FDA approved drug products and the drug packages in which they are sold, as well as ingredients and metadata about the drugs. We created an Application Programming Interface for SPLs using Scala, which will run on any implementation of the Java Virtual Machine (JVM) and is freely available through an open-source license for any non-commercial use.

1 INTRODUCTION

We are using Structured Product Labels (SPL) from the Food and Drug Administration (FDA) to capture new drug and related entities for representation in the Drug Ontology (DrOn). The reason is that manufacturers must submit an SPL for all new drugs approved in the United States, and thus this information is "directly from the source". It is also much richer than what is available in drug terminologies such as RxNorm. This paper describes our processes and the software we are developing to support them for extracting information from SPLs to update DrOn as new drug products come into existence.

SPLs are machine-readable files, submitted-to and released-by the FDA, containing prescription drug labeling and product metadata, such as National Drug Codes (NDCs) and drug product ingredients. They are available as full "current revision" releases, and monthly, weekly, or daily updates from the DailyMed website at http://dailymed.nlm.nih.gov. Each archive release contains individual archives. Each individual archive contains one XML file and may contain zero or more ".jpg" image files, which are referenced by the XML file.

SPLs have been found useful linking active ingredients and chemical entities (Hassanzadeh *et al*., 2013), extracting indication information (Fung *et al*., 2013), and improving detection of drug-intolerance issues (Schadow, 2009) and can be enhanced with current literature for greater safety, efficacy, and effectiveness (Boyce *et al*., 2013).

While a non-proprietary SPL parsing web service called "LinkedSPLs" is available (Hassanzadeh *et al*., 2013), we discuss below the lack of fitness for our use case.

Recent work (Hogan *et al*., 2013) has shown the benefit of ontological realism for avoiding scientific inaccuracy with the creation of the Drug Ontology (DrOn). In addition to modeling drug products, ingredients, and their respective dispositions, DrOn includes an extensive historical collection of identifiers such as NDCs. To keep DrOn updated as new drug products come into existence, we have designed a system for automated staging of data from three initial sources: RxNorm, ChEBI, and SPLs (Fig 1).

In addition to the need to drive DrOn maintenance, we are aware that SPL files are created by a large and diverse user base in industry and submitted to the FDA, so we have also included the capability to write SPL documents. Although

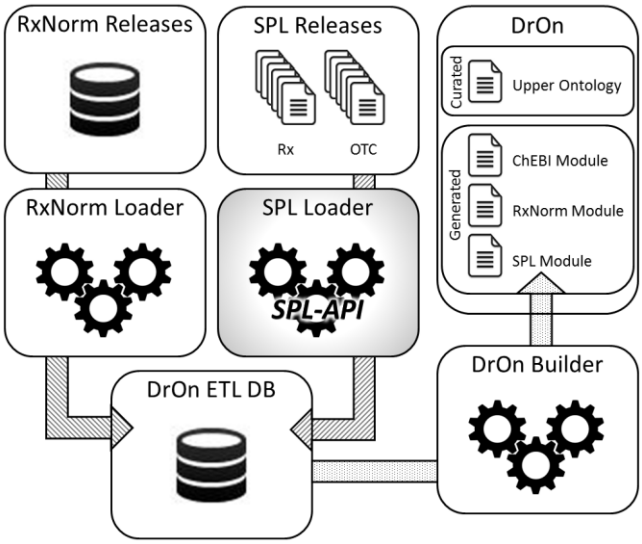

Figure 1: SPL-API is a part of the DrOn ETL process for automatically regenerating DrOn modules holding externally sourced information on drugs, ingredients, and dispositions.

one free tool for creating and editing SPL files exists ("SPL XForms"), we discuss here our motivations for creating a new tool.

Our implementation is an Application Programming Interface (API) for SPLs (SPL-API) which we use to update DrOn each day using the "daily updates" from Dailymed.

* To whom correspondence should be addressed: rahall2@uams.edu

SPL-API also includes features for downloading and parsing full releases and monthly and weekly updates.

Thus, the goal of this work is to create a generally useful open-source parser and writer for SPL XML files, and to use it within a larger system to update DrOn with applicable classes and their relationships, enabling additional data to be linked by other processes. As a result DrOn will be continuously updated with new drug products as they are approved.

2 METHODS

The DrOn support system must Extract, Transform, and Load (ETL) data from a variety of sources in order to automatically rebuild the "lower ontology" containing specific drug products from the latest sources. A DrOn Builder has been previously implemented (Hanna *et al*., 2013) which produces OWL 2.0 ontology modules from the initial DrOn database. Here, we describe the methods completed and planned to add SPL support to DrOn Builder.

2.1 Analysis of Existing SPL Tools

Our use case requires full access to all data in available resources and flexible server-side processing, preferring a local library over a connected service for both processing speed and algorithmic flexibility.

LinkedSPL provides SPL content for prescription and over-the-counter drugs, and is updated weekly. It can be accessed through a SPARQL endpoint to acquire the free-text contained in any "section" of the SPL file, which is defined by the "<section>" tag-set. Although the LinkedSPL software artifacts are freely available, and may be used locally, they are unable to report included label image files or a link to them. LinkedSPL also only parses prescription and OTC files, leaving out the "Remainder" labels (which include data on vaccines and some medical devices) and the "Animal" labels. Additionally, DrOn is not currently using RDF technologies (other than serialization of OWL into RDF), so we seek to avoid the complexity of adding an intermediate representation to the system.

A browser-based editor ("SPL XForms") for SPL format XML files is also available (Pragmatic, 2010). Developed in collaboration with the FDA, it can be used to view, create, edit, and validate the XML data once a Java-helper is allowed to load. Although useful to our study of individual XML files, it is not freely available as a local library, and thus could not be part of future system integrations.

2.2 Analysis of SPL Labels

Software was written to survey all XML elements and their attributes and relationships. Survey data will be available online (see section 3). An analysis was conducted on 45,182 SPL submissions in the Human Prescription, Human Over The Counter, Medical Device, and Animal label sets available as of April 22, 2013. The survey revealed elements that were primarily classes, those that were primarily attributes, and those that were unnecessarily verbose "wrapper" elements. Additionally, elements which are found in collections were identified using a "max and mean" algorithm.

SPL Documents have a fairly simple structure (Fig 2), combining a metadata-filled header and a body (contained in element <structuredBody>). The body contains a list of "section" elements. Section elements contain other section elements. While 90% of all files had 24 levels of nesting or less, some runaways include 40 levels. We note that every element deeper than 18 levels is related to a nested <containerPackagedProduct> element, which creates significant ambiguity for parsing drug products.

Each section is "typed" by the *loinc_code* attribute according to LOINC codes (e.g. "34067-9") that identify the common sections of SPLs (e.g. "Indications and Usage"). There are 87 codes allowed per ucm162057.htm (FDA, 2013), but only 84 were observed. Most documents include an SPL PRODUCT DATA ELEMENTS SECTION, an INDICATIONS & USAGE SECTION, and a WARNINGS SECTION. There was a mean of 1.48 PACKAGE LABEL.PRINCIPAL DISPLAY PANEL sections per document. All other codes were observed in less than half of the documents (and most were observed in less than 20% of the documents), while a full 33% of all SPL sections were coded as SPL UNCLASSIFIED SECTION, suggesting significant limitations in the standard.

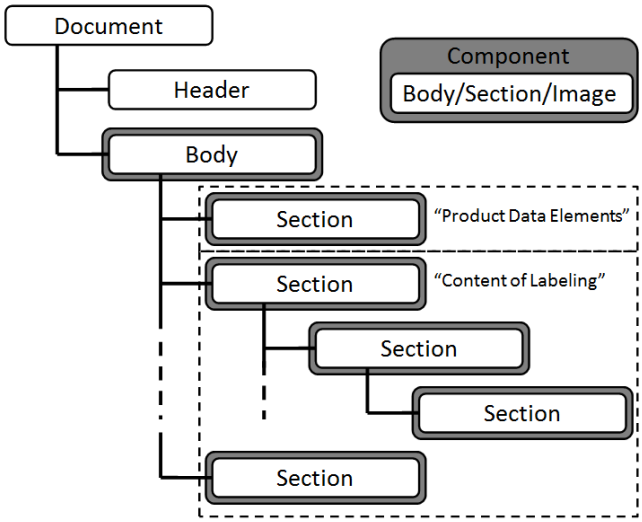

Figure 2: The SPL Document structure includes "wrapper classes" like <Component /> and "infinitely nest-able" <Sections />.

2.3 Technical Specifications

Our implementation is in Scala (version 2.10.1), which runs on any Java Virtual Machine (JVM) implementation and can be used within custom Java or Scala programs.

2.4 The DrOn Relational Schema

DrOn is influenced by RxNorm, and contains OWL 2.0 classes that model ingredients, semantic clinical drug forms,

semantic clinical drugs, and semantic branded drugs (Hogan *et al*., 2013). The staged data from all external resources used to build the "lower ontology" in DrOn are stored in a relational database whose schema follows the RxNorm file format closely. We added additional tables to the core schema for annotations regarding provenance, including a system-standard field for an "external link" to a resource-specific table. The external link can be used as an ID to load a resource-specific helper module as described below. At a minimum, the ID enables provenance for the external resource file. A "module" of resource-specific tables may be added to capture desired data. Although persistence of the all SPL information is unnecessary for our current integration with DrOn, our implementation represents all XML classes and attributes (except for the lowest level classes that represent HTML *formatting* of product labels). An applicable prefix for the table-set (e.g. "*spl_*") helps separate the tables visually when added to the same database.

The database currently holds on ~10^6 entries; the authors are experienced with databases containing ~10^9 entries. In the short term, expansion of ingredients and dispositions will increase the database more quickly than new products.

2.5 XML to JVM Classes with Code Generation

Code generation (cogen) has been shown useful for creating packages with numerous classes from OWL ontologies (Kalyanpur *et al*., 2004). It has also been useful in generating SOAP clients from WSDL files (Simpkins, 2008).

We developed a custom code generation utility to generate Scala classes for the SPL XML Format using the referenced XML Schema Definition (XSD), which validates the SPL format, and the survey results (see section 2.2). Classes were identified as elements (and their wrappers) which contain a number of attributes and zero or more collections. Attributes and elements of collections may both be typed as other classes. Collections were implemented to hold lists of child node types when necessary. Node types that never contain other node types, such as <id />, <name>, and <code />, are created as typed attributes of the classes that represent the containing node types. Accessors were generated for attributes; iterators were generated for collections.

Instead of attempting to create classes at run-time through the Java-beans paradigm (Kalyanpur *et al*., 2004), we chose to keep code generation in a separate utility, and import the resulting "top 34" classes into the dron.spl package.

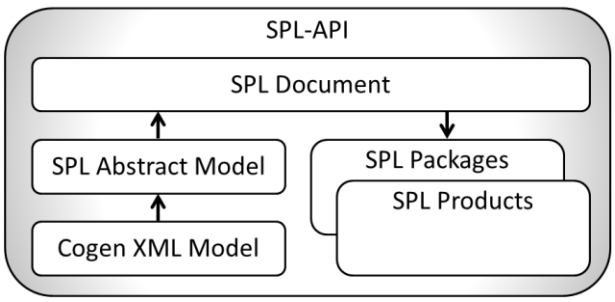

Figure 3: The SPL-API uses a set of 34 cogen classes that model the XML precisely within a set of classes that model an SPL document abstractly. The figure shows data flow for parsing.

2.6 The SPL-API

The parser and writer implementation is contained in the package *dron.spl*. Three sets of classes are included; classes that model the SPL XML format, classes that model the products and packages represented in the SPL XML files, and utilities necessary to manage Dailymed releases and updates (Fig 3 does not show utilities).

The root Scala class is "SPLDocument". At least one SPLDocument instance is created for each SPL submission file parsed (see section 2.8).

2.6.1 SPL XML Format

SPLDocument exposes classes and methods that represent an abstract SPL document, which in turn utilize the cogen classes that model the XML more exactly (Fig 3). A recursive printing algorithm built into the cogen classes enables the SPL-API to write SPL files.

When an XML element is always wrapped by another element, and the parent element never contains another element, then only one cogen class is represented. There are fourteen wrapped classes in SPL-API.

As an example of the layered class design, consider the "ComponentStructuredBody" cogen class that represents the XML elements for the SPL document body. (This class is the "structuredBody" element wrapped with a "component" element.) With this class, you can create a collection of "ComponentSection" cogen classes. However, the better approach would normally be to use the "section methods" of SPLDocument (e.g. *SPLDocument.addSection()*) to manage the sections of a document.

2.6.2 SPL Drug Classes

Classes for Drug Products, Ingredients, and Drug Label Data were created, along with a base class for Drug Packages. Ingredients are maintained as a collection in the Products class since each product may contain multiple ingredients, and each ingredient has "active" or "inactive" status attribute. Additional attributes are planned, such as the "strength" of the Ingredient within the Drug Product.

The primary subject of an SPL file—a drug package—is implemented as two classes that are sub-classed from the base class Package; SimplePackage and ComplexPackage. Every instance of Package must be related to at least one instance of Product. SimplePackage relates to exactly one NDC, while a ComplexPackages contains a collection of SimplePackage(s) along with its own metadata.

Parsing one SPL file produces a list of Package instances, which will have one or more elements of Content of Labeling Data. Lists of Label Data are maintained as a collection in the appropriate Package instance.

2.6.3 SPL-API File Utilities

We provide utilities for downloading full release and periodic updates, unzipping downloaded archives, unzipping all archives in a given directory, and unzipping individual submission archives. Additional data lookup utilities will be added, for example to translate *loinc_code*(s) to text labels, which is hoped to also assist users in minimizing the future share of SPL UNCLASSIFIED sections.

2.7 Matching NDCs

For our use case, a key step in correctly identifying all of the real drug products represented by the XML submission file is to identify all NDCs, but NDCs are not encoded in the XML scheme, and are only found in the free text of Product Data Elements sections. They generally contain the text string "NDC", and they always conform to the NDC 10-digit format (5-4-1, 4-4-2, or 5-3-2). We use pattern matching to identify multiple NDCs per text section. The NDCs that are found are checked against the National Drug Code Directory (FDA, 2013). The ability to correctly match all NDC's affects the quality of the results of the Drug Package listing (see section 2.6.2).

2.8 Core Document References

Of the 45,182 SPL submissions surveyed, 220 used the XML tag-set "<relatedDocument>". This tag includes the "SetID" of a "Core Document Reference" (FDA, 2012) (CDR), from which all sections are inherited by the containing document. When parsing a submission XML, the SPL-API will load a related document if it is available within the same directory, and create a separate instance of SPLDocument to hold the related document data. Documents that are "parents" can still have a <relatedDocument> tag, so the loading scheme is recursive, and is currently dependent on a small level of nesting.

When using the Scala classes that represent the XML model (see section 2.6.1), each SPL section is contained within its proper document, and each related document is accessible by the *SPLDocument.getRelatedDocument()* property accessor.

When using the Scala classes that represent Drug Packages (see section 2.6.2), all related documents are "flattened", and each section is included from all documents. Inheritance rules are unclear, so all sections are currently collected by section type. All identified Drug Products and Drug Packages will be included in the list.

2.9 ETL and External Resource Helpers

In addition to the SPL-API, a "helper" will be developed to load the parsed SPL data into the DrOn relational schema (represented as gears in Fig 1). A plugin system added to the DrOn builder will be able to identify the proper resource-specific plugin and pass the initialization necessary to complete loading for the next update.

3 CONCLUSION

We have developed an open-source API for processing SPLs in a Java Virtual Machine. A developer's release will be made available at the start of VDOS 2013, and will be available at: https://bitbucket.org/rogerhall68/spl-api.

Ongoing work includes loading processed data into the Drug Ontology to keep it current as new drug products are released.

ACKNOWLEDGEMENTS

This work was supported by award number UL1TR000039 from the National Center for Advancing Translational Sciences, award R01GM101151 from the National Institute for General Medical Science, and the Arkansas Biosciences Institute, the major research component of the Arkansas Tobacco Settlement Proceeds Act of 2000. This paper does not represent the views of NCATS, NIGMS, or NIH.

REFERENCES

Dailymed, http://dailymed.nlm.nih.gov/dailymed/downloadLabels.cfm

Hassanzadeh, O., Zhu, Q., Freimuth, R., & Boyce, R. (2013) Extending the "Web of Drug Identity" with Knowledge Extracted from United States Product Labels. *Proceedings of the 2013 AMIA Summit on Translational Bioinformatics*

Fung K.W., Jao C.S., & Demner-Fushman D. (2013) Extracting drug indication information from structured product labels using natural language processing. J Am Med Inform Assoc. 2013 May 1;20(3):482-8

Schadow, G. (2009) Structured Product Labeling Improves Detection of Drug-Intolerance Issues. *J. Am Med Inform Assoc*, **16**, 211–219.

Boyce, R., Horn J.R., Hassanzadeh O., de Waard A., Schneider J., Luciano J.S., Rastegar-Mojarad M., and Liakata M. (2013) Dynamic enhancement of drug product labels to support drug safety, efficacy, and effectiveness. *J. Am Med Inform Assoc*, **16**, 211–219.

Hogan, W.R., Hanna, J., Joseph, E., and Brochausen, M. (2013). Towards a Consistent and Scientifically Accurate Drug Ontology, This Volume.

Hanna J., Brochausen M., & Hogan W. R. (2013) Building a Realist Drug Ontology using RxNorm and Other Sources. This Volume.

FDA (2013) ucm162057.htm http://www.fda.gov/ForIndustry/DataStandards/StructuredProductLabeling/ucm162057.htm

Pragmatic Data (2010) SPL XForms, http://pragmaticdata.com/spl/form/

Kalyanpur, A., Pastor, D. J., Battle, S., & Padget, J. (2004, June). Automatic mapping of OWL ontologies into Java. In *SEKE* (Vol. 4, pp. 98-103).

LinkedSPL, http://dbmi-icode-01.dbmi.pitt.edu/linkedSPLs/

SPL XForms, http://pragmaticdata.com/spl/form/

Simpkins N., Generating a client from WSDL, http://www.eclipse.org/webtools/community/education/web/t320/Generating_a_client_from_WSDL.pdf

FDA (2013) National Drug Code Directory http://www.fda.gov/drugs/informationondrugs/ucm142438.htm

FDA (2012) Structured Product Labeling (SPL) Implementation Guide with Validation Procedures http://www.fda.gov/downloads/ForIndustry/DataStandards/StructuredProductLabeling/UCM321876.pdf