Formal Ontology in Information Systems
F. Neuhaus and B. Brodaric (Eds.)

doi:10.3233/FAIA210375

An Infrastructure for Collaborative Ontology Development

Lessons Learned from Developing the Financial Industry Business Ontology (FIBO)

Dean ALLEMANG[a,d], Pawel GARBACZ[b,d,e], Przemysław GRĄDZKI[d,e], Elisa KENDALL[c] and Robert TRYPUZ[b,d,e,1]

[a] *Working Ontologist LLC*
[b] *The John Paul II Catholic University of Lublin*
[c] *Thematix Partners LLC*
[d] *EDM Council*
[e] *MakoLab SA*

Abstract. Collaborative development of a shared or standardized ontology presents unique issues in workflow, version control, testing, and quality control. These challenges are similar to challenges faced in large-scale collaborative software development. We have taken this idea as the basis of a collaborative ontology development platform based on familiar software tools, including Continuous Integration platforms, version control systems, testing platforms, and review workflows.

We have implemented these using open-source versions of each of these tools, and packaged them into a full-service collaborative platform for collaborative ontology development. This platform has been used in the development of FIBO, the Financial Industry Business Ontology, an ongoing collaborative effort that has been developing and maintaining a set of ontologies for over a decade.

The platform is open-source and is being used in other projects beyond FIBO. We hope to continue this trend and improve the state of practice of collaborative ontology design in many more industries.

Keywords. ontology development tooling, continuous integration, hygiene test, collaborative ontology development, FIBO

Introduction

The development of a standard is inherently a collaborative process, regardless of the nature of the standard or the domain to which it applies. Good standards require input from a wide variety of subject matter experts from multiple organizations, working together in a highly distributed environment. The tools and process for developing a standard have to support collaboration smoothly and transparently. In this paper we focus on the situation in which one of the goals of the standard is to publish a shared data model or shared

[1]Corresponding Author: Robert Trypuz; E-mail:fibo@edmcouncil.org

reference data in the form of a formal ontology. This situation poses particular challenges for collaboration process and infrastructure, and is the focus of this paper. We draw on our experience in developing FIBO, the Finance Industry Business Ontology, to understand the challenges that face such a standards development effort, and outline the design of the infrastructure we have used for several years to manage those challenges.

FIBO evolved out of concerns that arose during the 2008 financial crisis among individuals who worked together in data governance and management. The most pressing issue at the time was the need for a *shared*, common vocabulary, focused on financial contracts and related concepts, that could be used for analysis and regulatory reporting purposes. From that time, FIBO has been sponsored and hosted by the Enterprise Data Management Council (EDM Council)[2], a global association for data management professionals, initially focused on financial services that has since expanded to other domains. Some selected modules of FIBO have been standardized by the Object Management Group (OMG)[3], and a new baseline standard is in work. As of the latest release[4], the production subset of FIBO includes roughly 1750 classes, 750 relationships and attributes, and over 14000 individuals (nominals, reference data and examples). The development version, which ranges in maturity from “almost releasable” to “really, really rough”, is more than 40 percent larger.

There are currently three FIBO primary content development teams working in parallel on different but related topics. In order to coordinate continuous integration of new and revised material, facilitate collaboration across topic teams, and ensure continuous quality improvement, leadership and process teams were put in place several years ago. One of the products of their work is a development framework created to automate aspects of ontology “unit-level testing", to guarantee a minimum level of quality. The individual tests are not necessarily novel. Many of them have been derived from, or inspired by, earlier work on ontology evaluation such as Chimaera [1], OntoClean [2], and OOPS! [3]. What is new, however, is a portable, open-source infrastructure that automatically runs these tests as an integral part of the ontology integration and publication process. This framework is designed for either a single development environment, or cross-organizationally, for example, for FIBO, and as planned for other ontology standards efforts at the EDM Council, OMG, and the Industrial Ontology Foundry (IOF)[5]. To date, in addition to its use for FIBO, the framework has been successfully deployed at Rensselaer Polytechnic Institute (RPI)’s Tetherless World Constellation[6], for use on several projects.

In this paper we present the infrastructure described above. The approach follows well-established principles and leverages tools commonly used in software engineering, treating *ontologies as source code*. Section 1 describes the FIBO development process, which is focused on use cases developed within working groups. Section 2 covers our approach to ontologies as managed source code, and how the infrastructure supports the development effort, with examples at various steps in the process. Finally, section 3 compares our approach with related attempts at supporting collaborative ontology development.

[2]See:https://edmcouncil.org
[3]See: https://www.omg.org
[4]See: https://github.com/edmcouncil/fibo/releases/tag/master_2021Q1
[5]See: http://www.industrialontologies.org
[6]See:https://tw.rpi.edu

1. Ontology Development Approach

The original motivation for FIBO was the failure of financial institutions and regulatory agencies to clearly exchange and integrate data about financial contracts and their counterparties, as demonstrated by the industry's failure to roll up risk with respect to those contracts. The initial FIBO use case was to provide an industry glossary that financial institutions and other market participants can use to meet regulatory requirements such as Dodd-Frank[7] in the U.S. and the MiFID II[8] framework in the EU for regulating financial markets. That use case was extended to cover additional requirements for data governance, data management, and enterprise glossaries mandated in the EU by the Basel Committee on Banking Supervision (BCBS) for risk data aggregation and reporting (BCBS 239[9]). Over the last few years, we have refined our approach as recommended in [4] to create instrument- or topic-specific use cases that add incremental value, resulting in significant progress by each of the working groups. The use cases include several usage scenarios and a number of competency questions per scenario, which are used to test the efficacy of the ontology as the work progresses.

The FIBO effort is organized into working groups, each consisting of at least one ontologist and some number of subject matter experts, which meet weekly to (1) review the use cases, (2) find areas in the ontologies where gaps remain, (3) refine and extend the ontologies to address those gaps and other issues raised by users, and (4) develop examples that answer the competency questions based on the revisions to the ontologies. Given an issue, use case, or partial use case, such as one scenario, the development process is roughly as follows:

1. In the context of a working group teleconference, review the existing ontology to determine what aspects of the ontology can be used to answer the question(s)
2. Identify the specific gap(s) and raise an issue to address the gap
3. Identify any missing concepts and work together to develop definitions and other annotations for those concepts and any important relationships based on a combination of appropriate resources (online financial dictionaries, offline financial dictionaries, ISO and other financial standards, etc.) and record our findings, discussion, and references in our minutes in the working group wiki
4. Create a branch in GitHub for the issue
5. Identify the ontology(ies) that need to be revised, where in the class hierarchy the concept(s) belong, and, importantly, whether or not there are existing patterns we can leverage in order to integrate the material
6. Integrate the new content into the relevant ontology(ies), reusing existing classes and properties as much as possible, and extending them as needed
7. Run at least one reasoner and perform SPARQL queries to ensure that the semantics seem reasonable and that the ontology(ies) remain logically consistent
8. Check the changes into GitHub and push them to a remote branch so that other members of the working group can review the results, automatically invoking the RDF serializer described below that ensures consistent serialization of the resulting RDF/XML via a custom Git hook

[7] See: https://www.govinfo.gov/content/pkg/PLAW-111publ203/html/PLAW-111publ203.htm
[8] See: https://www.esma.europa.eu/policy-rules/mifid-ii-and-mifir/
[9] See: https://www.bis.org/publ/bcbs239.pdf

9. Create example individuals (or update existing individuals) and test whether or not the competency question(s) can now be answered by the ontology (as appropriate), and check-in any examples that might be used as guidance for FIBO users
10. Once the working group members are comfortable with the revisions, perform a pull request in GitHub to get broader review, which automatically kicks off the infrastructure presented below; address any issues uncovered as a consequence
11. Once the pull request passes all of the stages in the publication cycle, at least two qualified reviewers must sign off (currently active members of at least one of the working groups plus other process team members have this privilege)
12. Finally, one of the process team will merge the pull request after it has been approved.

We iterate through steps 6-9, as needed, depending on the complexity of the issue and until we reach consensus on the resulting ontologies. Additional information regarding the methodology, minimal criteria for metadata and ontology content, and unit-level hygiene testing is outlined in our ontology guide[10].

Note that the development steps outlined above do not describe aspects of our methodology with respect to pairwise, pattern-driven development, as presented in [5]. A 'pair programming' approach that is increasingly pattern driven is applied regularly by the FIBO content teams for ontology revisions. Neither does it cover our test and integration process, which includes build out of example content, use of multiple reasoners and rule engines to ensure consistency and correctness, or validation of results against competency questions. The focus presented herein is on the automated infrastructure that is used to support the development process.

2. Ontology as Source Code

The long-term success of the process outlined in the previous section hinges on the ability to manage incremental change in the ontology (ontologies) over both short and long development cycles. This dynamic is not unique to ontology development, and in fact is quite familiar from collaborative software development methods, in which *an ontology is managed as a piece of program source code*. In our approach, we take this idea literally, and treat ontology components as source code, and organize the development using tools and techniques familiar from software engineering. In particular, we focus on four familiar areas of software development: Modularity, Version Control, Continuous Integration (CI), and Testing.

2.1. Modularity and Maturity Levels

Just as is the case with any large software project, components of a large ontology like FIBO have different governance requirements. These include ownership, speed of update, and dependencies. Just as is the case for conventional software, *modularity* is a powerful tool for managing governance. As an example from the point of view of the FIBO audience, modularization allows us to express the maturity level of different parts of the ontology,

[10]See: https://github.com/edmcouncil/fibo/blob/master/ONTOLOGY_GUIDE.md

allowing the community to understand which FIBO ontologies can be used “as they are” and which ones are still under development and likely to be more volatile.

FIBO is structured as a collection of relatively small ontologies; currently, there are about two hundred ontologies. Each ontology has its own namespace and is recorded (as source code) as a single OWL-compliant file, serialized as RDF/XML. Furthermore, these ontologies are organized into a hierarchical structure that is reflected in a file folder structure stored on GitHub[11].

At the top level, FIBO currently has ten modules called *domains* (represented in the file system as high-level directories). Within these domain areas are one or two levels of *subdomains*. In the smallest subdomains (bottom level directories in GitHub), are the two hundred ontology files. Thus, any ontology has a unique place in the domain/module hierarchy.

For example, for the Business Entities domain and the Legal Entities module, we have the following structure:

- **(FIBO domain)** Business Entities
  - **(FIBO module)** Corporations
  - **(FIBO module)** Functional Entities
  - **(FIBO module)** Government Entities
  - **(FIBO module)** Legal Entities
    - **(FIBO ontology)** Corporate Bodies Ontology
    - **(FIBO ontology)** Formal Business Organizations Ontology
    - **(FIBO ontology)** Legal Entity Identifier (LEI) Entities Ontology
    - **(FIBO ontology)** Legal Persons Ontology
  - **(FIBO module)** Ownership and Control
  - **(FIBO module)** Partnerships
  - **(FIBO module)** Private Limited Companies
  - **(FIBO module)** Sole Proprietorships
  - **(FIBO module)** Trusts

Each ontology in FIBO is described by one of three maturity levels[12]: *release*, *provisional*, or *informative*. Ontologies marked as *release* are considered to be stable and mature, and ready for use by stakeholders. Ontologies marked as *provisional* are considered to be under development, so they are less stable, and one can expect changes occurring in their content more frequently. Ontologies marked as *informative* have been deprecated, but are still included for informational purposes because some provisional concept references them.

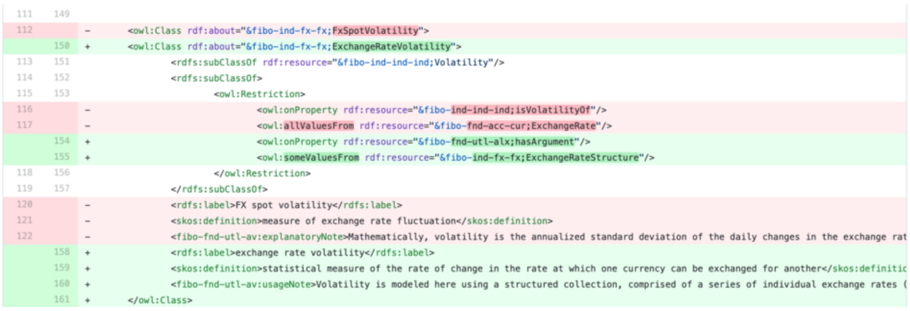
2.2. Collaborative Version Control

One of the advantages of viewing ontologies as source code is that we can take advantage of decades of experience with managing collaboration in code production. The state of the art in software version control is embodied in a service called GitHub, which has become the default infrastructure for source code collaboration.

[11] See: https://github.com/edmcouncil/fibo

[12] Maturity levels in FIBO are assigned only to FIBO ontologies (so not to FIBO domains or modules).

GitHub's operation is based on the idea that it is possible to compare two versions of the same file and display the differences in a simple form, as shown in Figure 1. This works very well for program code, since a typical change is done using a text editor or an IDE that make changes directly to the code; any changes show up as simple differences from one version to the next. This means that if two contributors make changes to the same file, the changes from each of them can be detected, and then displayed, processed or even automatically merged.



```
-    <owl:Class rdf:about="&fibo-ind-fx-fx;FxSpotVolatility">
+    <owl:Class rdf:about="&fibo-ind-fx-fx;ExchangeRateVolatility">
             <rdfs:subClassOf rdf:resource="&fibo-ind-ind-ind;Volatility"/>
             <rdfs:subClassOf>
                     <owl:Restriction>
-                            <owl:onProperty rdf:resource="&fibo-ind-ind-ind;isVolatilityOf"/>
-                            <owl:allValuesFrom rdf:resource="&fibo-fnd-acc-cur;ExchangeRate"/>
+                            <owl:onProperty rdf:resource="&fibo-fnd-utl-alx;hasArgument"/>
+                            <owl:someValuesFrom rdf:resource="&fibo-ind-fx-fx;ExchangeRateStructure"/>
                     </owl:Restriction>
             </rdfs:subClassOf>
-            <rdfs:label>FX spot volatility</rdfs:label>
-            <skos:definition>measure of exchange rate fluctuation</skos:definition>
-            <fibo-fnd-utl-av:explanatoryNote>Mathematically, volatility is the annualized standard deviation of the daily changes in the exchange rat
+            <rdfs:label>exchange rate volatility</rdfs:label>
+            <skos:definition>statistical measure of the rate of change in the rate at which one currency can be exchanged for another</skos:definitic
+            <fibo-fnd-utl-av:usageNote>Volatility is modeled here using a structured collection, comprised of a series of individual exchange rates (
+    </owl:Class>
```

Figure 1. GitHub 'diff' shows the changes between two snapshots in unified diff format.

GitHub falls short in a situation in which code is not directly edited as text files, as is often the case when editing ontologies. Typically, a contributor to an ontology will want to view and edit the ontology with a user interface tuned specifically to ontology development. The actual text files that serve as the 'golden copy' of the ontology are the output from such tools.

The RDF standard includes a number of serialization options (e.g., Turtle [6] and RDF/XML [7]); each of these specifies a syntax for writing down triples in a text file. But the relationship between triples and a file is not one-to-one; any set of triples can be written down in a wide variety of ways. This is not an unusual situation; even conventional programming languages are agnostic about things like variable names, the order in which variables and subroutines are declared, etc. The difference is that for most programming languages, a human decides these things when they edit the text file; when an ontology is edited through a graphical user interface, these decisions are made by the ontology management application. Each of these many tools [13] makes different serialization decisions. This means that two versions of the same file that do not differ at all could be saved in files that are vastly different - thwarting the basis on which GitHub works.

There are three basic approaches to this problem:

- Standardization on a single tool. If this tool is consistent in the way it serializes triples, then similar ontology versions will be written in similar files, and GitHub can work appropriately. Most modern ontology editors satisfy this condition.
- Use a tool with version control built in. An example of such a system (working with RDF triples) is MOBI [14].

[13]https://www.w3.org/2001/sw/wiki/Tools
[14]https://mobi.inovexcorp.com/

- Post-process files with a stable serializer. A hybrid approach is to allow each contributor to use whatever tool they like, but process each ontology file after it is written and before it is committed to version control. This process must preserve the content of the file (same triples in as out), but serialize it in a consistent way.

Each of these approaches has its advantages and disadvantages. Using a single tool has the advantage that contributors can collaborate easily, but has the disadvantage that it does not allow contributors to use whatever tool they choose. Using a tool with version control in it simplifies the process quite a lot, but also limits tool choices. These approaches are sometimes appropriate in an enterprise setting, where there are many motivations for controlling software use. Using a stable post-processor allows contributors to use whatever tool they like (and even encourages the development of new tools for specific uses). A disadvantage of this approach is that it requires extra infrastructure for each collaborator; they must install the serializer before they can contribute. This approach is more appropriate in a wide collaboration setting, where different collaborators come from different organizations. Not surprisingly, this latter approach is the approach that FIBO has taken. The EDM Council has provided an open-source serializer[15]. This allows ordinary text comparison tools to operate on the OWL files that represent the FIBO ontologies, and, in turn, allows FIBO developers to follow any workflow based on GitHub. In the case of FIBO, we have integrated GitHub with a continuous integration platform (see section 2.3 below) that runs a number of services and tests over each committed change.

2.3. Continuous Integration

FIBO uses a popular continuous integration (CI) platform called Jenkins; but any of several platforms perform similar functions. The job of Jenkins is to coordinate actions that will be automatically taken whenever a change is committed to GitHub.

When a change to is committed, it triggers a “chain of actions” (which is called a “pipeline”), in Jenkins, as shown in Figure 2.

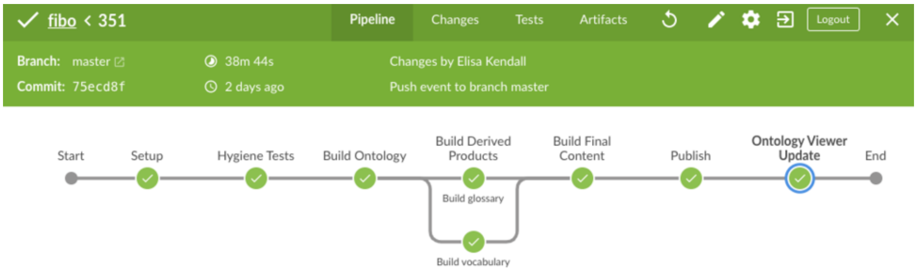


Figure 2. FIBO’s continuous integration and continuous delivery pipeline system.

The sequence of stages in the pipeline is as follows:

- “Setup” prepares the infrastructure for the next stages,
- “Hygiene Tests” checks whether FIBO follows all the principles from the FIBO ontology guide

[15]See: https://github.com/edmcouncil/fibo/blob/master/CONTRIBUTING.md#fibo-serialization-tools

- "Build Ontology" is responsible for creating ontology files in different serializations,
- "Build Derived Products" creates FIBO derived products such as SKOS version of FIBO or FIBO glossary (a tabular version of FIBO containing labels and definitions, see below)
- "Build Final Content" combines the results of the building of all the products,
- "Publish" places files on the webserver,
- "Ontology Viewer Update" sends an "update" message to the Ontology Viewer

The many services used by Jenkins to manage FIBO updates are included in a software module called the ontology-publisher[16]. The ontology-publisher is deployed as a docker image, built on the basis of the Dockerfile[17] and made available on the public EDMC Docker Hub account [18]. The workhorse of this image is the "publish.sh" script[19], which enables the execution of all steps described in the caption of figure 2. The "ontology-publisher" uses the following components: Python libraries RDFLib[20] and PyLD[21], EDMC's rdf-toolkit[22], Apache Jena[23] and other. It is built in a modular way, which allows for the flexible addition of new steps of the process and their easy integration into the Jenkins pipeline.

2.4. Testing

In conventional software development, testing is a large component of any development activity. FIBO uses Jenkins to perform a wide range of automated tests.

FIBO is developed by a heterogeneous community. In order to ensure consistency in contributions from a variety of community members, the FIBO team has developed a set of *hygiene tests* that are run automatically each time a FIBO change is committed to GitHub. FIBO organized hygiene tests into three categories:

1. errors
2. production errors
3. warnings

All three categories of tests run against the full FIBO ontology with the proposed change. If a test fails, Jenkins will take an action depending on the category of the test. If a *warning* test fails, a warning is issued, and the developer and the FIBO team are made aware of the transgression. In this case, ontology processing continues. If an *error* test fails, then processing stops, and no more steps are taken. The change is considered a fatal error, and it cannot be accepted. In the case of a *production error*, the same test is run over the *production* version of FIBO and then separately over the *development* version of FIBO; a failure in the production version of FIBO is treated as fatal (just like a "vanilla" error), whereas a failure in the development version of FIBO is treated as a warning. If no fatal error occurs, the change moves on to a review phase.

[16] https://github.com/edmcouncil/ontology-publisher
[17] https://github.com/edmcouncil/ontology-publisher/blob/master/Dockerfile
[18] https://hub.docker.com/repository/docker/edmcouncil/ontology-publisher
[19] https://github.com/edmcouncil/ontology-publisher/blob/master/publisher/publish.sh
[20] https://rdflib.dev/
[21] https://github.com/digitalbazaar/pyld
[22] https://github.com/edmcouncil/rdf-toolkit
[23] https://jena.apache.org/

The hygiene tests are implemented as SPARQL queries that are called by Jenkins using the Jena ARQ processor.[24] An example of such test can be found in Listing 1.

Once a commit has passed the tests, it is eligible to become a pull request. This is a concept that is common in git-based software development, whereby a contributor proposes a change to the ontology, and others approve it and merge it into the main, published branch. In the case of FIBO, a contributor proposing a pull request asserts that they have read and understood the FIBO workflow, including all the tests listed above, and signs the Developer Certificate of Origin (DCO) certifying that he/she has the right to submit the proposed change under the MIT license. The Continuous Integration system validates that these tests have passed.

```
# banner Definitions shouldn't be circular – this finds direct circularities therein.

SELECT DISTINCT ?error ?definition ?label
WHERE
{
  ?s rdfs:label ?label .
  ?s skos:definition ?definition .
  FILTER NOT EXISTS {?s a owl:NamedIndividual} .
  FILTER (REGEX(?definition, "\\W"+?label+"\\W"))
  FILTER (CONTAINS(str(?s), "edmcouncil"))

  BIND(concat("PRODERROR: Definition of ",str(?s)," is immediately circular ") AS ?error)
}
```

Listing 1: Hygiene test example

Once the pull request has been made, FIBO has a board of reviewers, two of which must approve the change, at which point (after a minimum review period has passed), it is merged into the main branch. This assures that changes make it in to FIBO in a timely fashion after undergoing peer review. This sort of workflow is typical of large, open-source code projects, in which the code undergoes some sort of unit and integration tests before it is accepted into the main branch. In the case of ontologies, the process is the same, but the tests are tailored to ontology development in OWL.

As of the last run our hygiene test finds zero errors or warnings in the production subset of FIBO and around 1000 warnings in the development subset. The latter number, although significant, is constantly decreasing over time.

2.5. Ontology Derived Products

The canonical version of FIBO is rendered in the OWL 2 DL language. Use of combinations of exact qualified cardinalities, intersections, and existential quantification to differentiate financial instruments from similar variants requires greater expressivity than is available in OWL 2 RL. For example, a fixed-float interest rate swap is a swap that has exactly two swap legs, one of which is a fixed-rate debt instrument and the other is a floating-rate debt instrument, as shown in Figure 3.

[24] All tests can be found on https://github.com/edmcouncil/fibo/tree/master/etc/testing/hygiene.

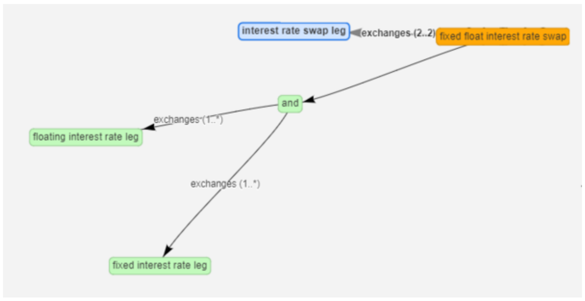


Figure 3. fixed-float interest rate swap expressed in OWL.

On the other hand, many data management applications don't require this level of detail; they really just want to know that a fixed-float interest rate swap has a fixed-rate leg and a floating-rate leg, more like the simpler diagram shown in Figure 4.

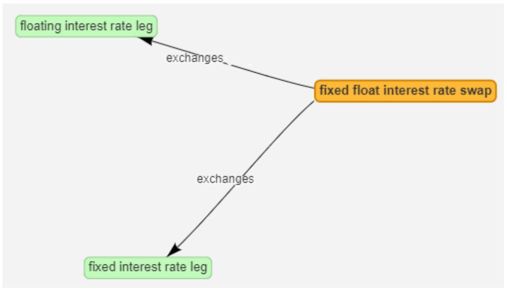


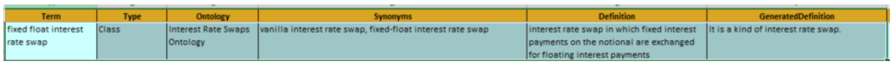
Figure 4. Fixed-float interest rate swap, in simplified form.

For this reason, FIBO is published in alternate forms that are faithfully generated from FIBO, but that include less information than the prescriptive model. For example, the structure in Figure 4 is automatically derived from the logical ontology in Figure 3; it isn't as detailed, but approximates the information. Specifically, we provide two derivative version of FIBO:

1. FIBO as a SKOS vocabulary
2. FIBO as a data dictionary

SKOS FIBO vocabulary expresses all FIBO classes as individuals in the spirit of SKOS. These individuals are related by the respective object properties. This is the structure that is shown in Figure 4.

In many cases, data managers are accustomed to seeing data elements in a spreadsheet, with names, synonyms and definitions in a tabular form. This typically expresses much less information than is available in an OWL ontology, but is amenable for human review, and for use in glossary applications. Figure 5 shows an example of this sort of display.



Term	Type	Ontology	Synonyms	Definition	GeneratedDefinition
fixed float interest rate swap	Class	Interest Rate Swaps Ontology	vanilla interest rate swap, fixed-float interest rate swap	interest rate swap in which fixed interest payments on the notional are exchanged for floating interest payments	It is a kind of interest rate swap.

Figure 5. Data dictionary entry for the fixed-float interest rate swap.

FIBO data dictionary is a single table that includes all FIBO classes, properties, and describes the basic metadata features of each one.

2.6. Ontology Viewer

Finally, in order to enhance the collaborative development of FIBO, we provide a web application to browse through its contents. The Ontology Viewer is a REST API-based web application written in Java that allows accessing the FIBO structure and its content in a user-friendly way. It is an open-source community project hosted by EDMC[25].

The Ontology Viewer provides an interactive experience that reflects all of the features we have described here: the modularity of FIBO is reflected in the navigation functionality of the viewer. The maturity of each ontology is shown. For components of the ontologies (classes and properties), they are uniformly documented with labels, synonyms and logical relationships. These are expressed in words (describing the logical relationships) as well as diagrams. Every mention of a resource in FIBO is a hyperlink that leads to more information about that entity. In addition to the human-readable display, the full URI of each resource can be copied with a single click, for use by developers when composing SPARQL queries.

The Ontology Viewer has some unique advanced features; in addition to a full-text search, it also includes a time machine. Every proposed pull request is registered in the Ontology Viewer; a reviewer can select which version of FIBO is to be viewed (including new, proposed versions as well as old, released versions). This feature is invaluable for reviewers who want to understand the ramifications of a proposed change, or to research how FIBO has changed over the years.

Ontology Viewer always presents up-to-date content; it is kept current as part of the continuous integration logic described in section 2.3. The FIBO viewer is available to the public at https://spec.edmcouncil.org/fibo/ontology.

3. Related Work

Open-source tools for collaborative ontology modeling, such as WebProtégé[8], are widely used in the knowledge representation community for specifying ontologies in the Web Ontology Language (OWL)[9]. There are fewer common processes and downstream collaboration environments, however, for development involving multiple, independently evolving ontologies, that must be kept in sync and that ensure the resulting ontologies meet the quality levels necessary for publication of an international standard.

As far as we are aware there is no publicly available framework for continuous ontology development with the same level of support for distributed, social development as the infrastructure described above. There are however a couple of web applications that provide some of the capabilities we offer – in some cases surpassing ours in terms of flexibility or scope[26].

There are a number of frameworks available today for collaborative ontology development. Most of these did not exist, however, when our work on the FIBO

[25] See https://github.com/edmcouncil/onto-viewer

[26] We will not discuss ontology editors here although some of them, like VocBench [10] or WebProtege [8], do provide support for collaborative ontology development.

infrastructure was initiated. The Ontology Development Tookit (ODK)[27] was developed in parallel with our work to assist the OBO Foundry community[28] in standardizing their collaborative ontology development workflows using GitHub. It supports methods of extracting various subsets of an ontology, including some axiom stripping and relaxation of axioms which we do not do in order to produce versions of a baseline ontology that can interoperate with other ontologies for certain purposes. It is possible that we will take advantage of some of their insights for FIBO users over time. ODK does not, however, automatically publish revisions covering the alternative representations, including the 'flattened' SKOS Vocabulary and data dictionary that the FIBO user community requires[29] OBO Foundry community members and others also use the ROBOT[30] semantic diff tool as a part of their process. While such a tool is quite useful, and some FIBO users also use ROBOT as part of their workflow, ROBOT does not serialize an ontology to support the file-oriented diff process that the RDF Toolkit provides, which has proven to be essential for visual comparison in GitHub for FIBO users. Another such framework is OnToology, a web-based tool designed to automate part of the ontology development process in a collaborative environment, *i.e.,* in GitHub. When one registers an ontology with OnToology, the tool will monitor for changes and upon each (committed) change it will generate a new pull request that contains: (i) the ontology documentation (with several proposals for diagram representation), its evaluation, and publication of the ontology in the user's repository[11]. VoCol is an integrated environment to support version-controlled vocabulary development. It supports a round-trip model of vocabulary development: modeling, population, and testing. To facilitate modeling VoCol allows users to formulate queries which represent competency questions for testing the expressivity and applicability of a vocabulary. To support testing, it provides the automatic detection of "bad smells" in the vocabulary design by employing SPARQL patterns. For modeling purposes, VoCol integrates a number of techniques facilitating conceptual work: automatically generated documentation and visualizations provide different views on the vocabulary as well as an evolution timeline supporting traceability[12].

There also exist various tools that automatically check the level of compliance of the ontology development against a set of logical and conceptual requirements. OOPS! is a web app that scans ontologies looking for potential pitfalls that could lead to modelling errors.[31] OOPS! is intended to be used by ontology developers during the ontology validation activity, particularly during the diagnosis task. OOPS! currently catalogs 41 checks, of which 33 are automated. Some of these automated checks overlap with the FIBO checks: P01-P03, P08, P32, P34, and P35.[13][32]. RDFUnit is a debugging framework that can run automatically generated (based on a schema) and manually generated test cases against an RDF – either inputted directly or via an endpoint[33]. The test case definition language is SPARQL, which proved to be convenient to directly query for identifying violations. For rapid test case instantiation, a pattern-based SPARQL-template

[27] See https://github.com/INCATools/ontology-development-kit

[28] See https://www.obofoundry.org

[29] As of this writing, we have not had the resources to test whether or not ODK supports the level of analysis over the tens of ontologies in the imports closure of some changes as a part of the testing and publication process that the FIBO infrastructure does.

[30] See http://robot.obolibrary.org/

[31] http://oops.linkeddata.es/

[32] Note that OnToology uses OOPS! for its evaluation service.

[33] https://aksw.org/Projects/RDFUnit.html

engine, running over a library of common patterns is supported where variables can be easily bound into patterns.[14]. RDFUnit thus corresponds our hygiene test component, in fact it provides more flexible functionality due to the automatic test generation. OntoSeer is a recent Protege plugin that provides automatic recommendations while ontology development [15]. These recommendations concern, among other things, IRIs for classes and properties, subsumption hierarchy structure, and recommendations based on the repository of ontology design patterns (http://ontologydesignpatterns.org/). We may consider adapting it for use with the patterns we use in FIBO in the future.

Finally, there is an emerging stream of research focused on ontology evolution, including visualization of changes in ontology design. ChImp, a Protege plugin, is an instance of this approach. It visualizes the diff between the current and the previous state of the ontology, its impact on logical consistency, and the customized metrics of on how the diff changed such aspects as the property to class ratio or the annotation richness[16].

4. Future Work

The number and nature of the SPARQL queries we run has increased considerably over the last 18 months, but more tests can be added. FIBO has evolved to be increasingly pattern oriented, as mentioned above, with recent focus on situational patterns such as ownership and control. The patterns are complicated, and it would be helpful to add tests that look for issues in their application, potentially leveraging the approach taken by OntoSeer. The same is true for other patterns, such as those involving time, payment, dividend, and other schedules, and new patterns that emerge in time. Our hope is for the framework to be widely used, so organizing the tests to allow other implementations to pick and choose which ones they care about and ignore others would be useful. We also want to make it easy for others to contribute new tests and ultimately catalog them in such a way that makes the tests searchable. Finally, we plan to integrate feedback from other implementations to evolve the framework to facilitate usage outside of FIBO.

5. Conclusion

Our experience managing FIBO, a collaborative, standardized ontology development effort, has given us some insight into how to manage such projects. The effort is daunting, and we don’t claim to have got every decision right over the decade of FIBO development and maintenance; but we have learned from our mistakes, and have incorporated them into an open ontology development environment. Along the way, we realized that many of the challenges we face are common to any collaborative software effort, and so we have drawn on the experience of hundreds of open software projects to adapt their best practices to the unique requirements of ontology development.

But we also found that some challenges are unique to ontology development. Unlike most software development, many phases of ontology development are done using graphical tools, so that the ontology files themselves are not directly edited by the developers, This places special requirements on the infrastructure to enable collaboration. Ontology testing is possible, but unlike most software, there isn’t a notion of "running" an ontology; you can draw inferences, and run queries over it, but there isn’t an "input/output" relationship that specifies the desired behavior.

Despite these differences, we have found the parallel between code management and ontology management to be very productive, and that the workflows (e.g., commits and pull requests) and tools (version control, continuous integration) from software management do in fact apply well to ontology development.

It is our hope that we can interest more ontology development projects in using and contributing to this effort, providing much needed support across many communities and industries.

Acknowledgments

Many people have contributed FIBO and to the development, deployment, and maintenance of the infrastructure over the last decade. We would like to thank Mike Atkin and Mike Bennett for the dedication and hard work that launched FIBO, including the work they did to bring countless skeptical bankers, vendors, and consultants together to create the content at the heart of the ontologies. We would also like to recognize Dennis Wisnosky for his vision, establishing the original structure and teams that enabled scalable development of the FIBO content, and for developing the overall Build-Test-Maintain-Deploy process that led to the creation of the infrastructure. We also want to thank Anthony Coates and Omar Khan for their work on the RDF Serializer, and Jacobus Geluk and Pete Rivett for their contributions to the infrastructure.

References

[1] McGuinness DL, Fikes R, Rice J, Wilder S. The Chimaera Ontology Environment. In: AAAI/IAAI. AAAI Press / The MIT Press; 2000. p. 1123-4. Available from: http://dblp.uni-trier.de/db/conf/aaai/aaai2000.html#McGuinnessFRW00.

[2] Guarino N, Welty C. A Formal Ontology of Properties. In: Dieng R, Corby O, editors. Knowledge Engineering and Knowledge Management Methods, Models, and Tools. Springer, Berlin, Heidelberg; 2000. p. 97-112. Available from: https://www.researchgate.net/publication/2362508_A_Formal_Ontology_of_Properties.

[3] Poveda-Villalón M, Gómez-Pérez A, Suárez-Figueroa MC. OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation. International Journal on Semantic Web and Information Systems (IJSWIS). 2014;10(2):7-34.

[4] Kendall EF, McGuinness DL. Ontology Engineering. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers; 2019. Available from: https://doi.org/10.2200/S00834ED1V01Y201802WBE018.

[5] Shimizu C, Hirt Q, Hitzler P. MODL: A Modular Ontology Design Library. In: Janowicz K, Krisnadhi AA, Villalón MP, Hammar K, Shimizu C, editors. Proceedings of the 10th Workshop on Ontology Design and Patterns (WOP 2019) co-located with 18th International Semantic Web Conference (ISWC 2019); 2019. p. 47-58.

[6] Carothers G, Prud'hommeaux E. RDF 1.1 Turtle. W3C; 2014. Available from: http://www.w3.org/TR/2014/REC-turtle-20140225/.

[7] Schreiber G, Gandon F. RDF 1.1 XML Syntax. W3C; 2014. Available from: http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.

[8] Horridge M, Gonçalves RS, Nyulas CI, Tudorache T, Musen MA. WebProtégé: A Cloud-Based Ontology Editor. In: Companion Proceedings of The 2019 World Wide Web Conference. New York, NY, USA: Association for Computing Machinery; 2019. p. 686–689.

[9] Bao J, Kendall EF, McGuinness DL, Patel-Schneider PF. OWL 2 Web Ontology Language Quick Reference Guide (Second Edition). W3C; 2012. Available from: https://www.w3.org/TR/owl2-quick-reference/.

[10] Stellato A, Fiorelli M, Turbati A, Lorenzetti T, Gemert W, Dechandon D, et al. VocBench 3: A collaborative Semantic Web editor for ontologies, thesauri and lexicons. Semantic Web. 2020 05;11:1-27.

[11] Alobaid A, Garijo D, Poveda-Villalón M, Santana-Perez I, Fernández-Izquierdo A, Corcho O. Automating ontology engineering support activities with OnToology. Journal of Web Semantics. 2019;57:100472.

[12] Halilaj L, Petersen N, Grangel-González I, Lange C, Auer S, Coskun G, et al. Vocol: An integrated environment to support version-controlled vocabulary development. In: European Knowledge Acquisition Workshop. Springer; 2016. p. 303-19.

[13] Gómez-Pérez A. Did You Validate Your Ontology? OOPS! The Semantic Web: ESWC 2012 Satellite Events: ESWC 2012 Satellite Events, Heraklion, Crete, Greece, May 27-31, 2012 Revised Selected Papers. 2015;7540:402-7.

[14] Dimou A, Kontokostas D, Freudenberg M, Verborgh R, Leghmann J, Mannens E, et al. Test-driven Assessment of [R2] RML Mappings to Improve Dataset Quality. In: Proceedings of the 14th International Semantic Web Conference: Posters and Demos; 2015. p. 747-58.

[15] Bhattacharyya P, Mutharaju R. OntoSeer: A Tool to Ease the Ontology Development Process. In: 8th ACM IKDD CODS and 26th COMAD. ACM; 2021. p. 428-8.

[16] Pernischova R, Serbak M, Dell'Aglio D, Bernstein A. ChImp: Visualizing Ontology Changes and their Impact in Protégé. In: Proceedings of the Fifth International Workshop on Visualization and Interaction for Ontologies and Linked Data co-located with the 19th International Semantic Web Conference (ISWC 2020). Ceur – Workshop Proceedings; 2020. p. 47-60.