LexTerm Manager: Design for an Integrated Lexicography & Terminology System

Logan Kearsley, Joshua Elliot, Jason Housley, Alan Melby
Brigham Young University Department of Linguistics & English Language
4064 JFSB
Provo, UT 84602 USA
Email: chronosurfer@gmail.com, joshuacelliott@gmail.com, housleyjk@gmail.com, alan.melby@gmail.com

Abstract

We present a design for a multi-modal database system for lexical information that can be accessed in either lexicographical or terminological views. The use of a single merged data model makes it easy to transfer common information between termbases and dictionaries, thus facilitating information sharing and re-use. Our combined model is based on the LMF and TMF metamodels for lexicographical and terminological databases and is compatible with both, thus allowing for the import of information from existing dictionaries and termbases, which may be transferred to the complementary view and re-exported. We also present a new Linguistic Configuration Model, analogous to a TBX XCS file, which can be used to specify multiple language-specific schemata for validating and understanding lexical information in a single database. Linguistic configurations are mutable and can be refined and evolved over time as understanding of documentary needs improves. The system is designed with a client-server architecture using the HTTP protocol, allowing for the independent implementation of multiple clients for specific use cases and easy deployment over the web.

Keywords: TBX, terminology, lexicography

1. Introduction

Termbases and dictionaries have a great deal of overlap in the kind of information they contain. However, there is very little overlap in the tools used to build and interact with them or in the data formats used to store and transmit them. This means that the data assembled in the creation of a termbase or dictionary is rarely re-used to assist in creating the other kind of artefact. Since authoring dictionaries and termbases as well as ensuring their accuracy is a time consuming and labor intensive process, there is potentially a great deal to be gained from facilitating this kind of re-use of lexical information. Building on previous work by Melby and Wright (1999), on translating between lexicographical and teminological data formats, we have designed a system to merge lexicographical and terminological information into a single database. Combined entries can then be accessed in either lexicographical or terminological views depending on the needs of the user as a lexicographer or terminologist, with any data entered being immediately available in either format. We have chosen the name “LexTerm” to reflect this fusion of lexicographical and terminological functions. Hereafter, lexicographical and terminological data will be referred to as lexical data (i.e., dealing with lexical items) when considering the union of the two.

It is important to note that this work should not be confused with the similarly-named Linguoc LexTerm software (Oliver et al., 2007). While Linguoc LexTerm deals with the problem of automatic extraction of terms for inclusion in a termbase from a corpus of text, the current LexTerm manager project is concerned with the sharing of data between lexicographical and terminological contexts and curation of that data once it has been acquired.

1.1. Standards Compatibility

The designs for each half of the combined LexTerm data model were based on the existing Lexical Markup Framework (LMF) standard (ISO 24613, 2008) and the Terminological Markup Framework and TermBase Exchange (TMF and TBX) standards (ISO 16642, 2003; ISO 30042, 2008), and the combined model is compatible with both. Our model structure was derived by identifying data elements that are semantically identical or similar between the LMF and TMF metamodels followed by the selection of a canonical name and merger of matching elements into one in the combined model. While the LMF and TMF metamodels are both conceived of as describing hierarchical trees, the combined LexTerm data model is a more generic directed acyclic graph (DAG). Terminological or lexicographical views are derived by choosing to consider the appropriate nodes (concepts or languages, respectively) as roots and allowing the remainder of the graph to 'hang' from them (discarding nodes containing data elements irrelevant to the chosen view), thus restoring the appearance of the appropriate tree structure.

1.2. Inter-Standard Compatibility

There are many pairs of data elements between the LMF and TMF models that are not identical in meaning, but are similar enough that inclusion of both in the combined model would result in unnecessary duplication of information, requiring additional human effort to fill both fields and hindering the goal of automatically transferring as much information as possible between the two viewing modes. Additionally, in order to further the goal of sharing information from multiple sources, it’s desirable to maintain at least import compatibility with as many pre-existing termbase and dictionary formats as possible. The attempt to support compatibility with each additional

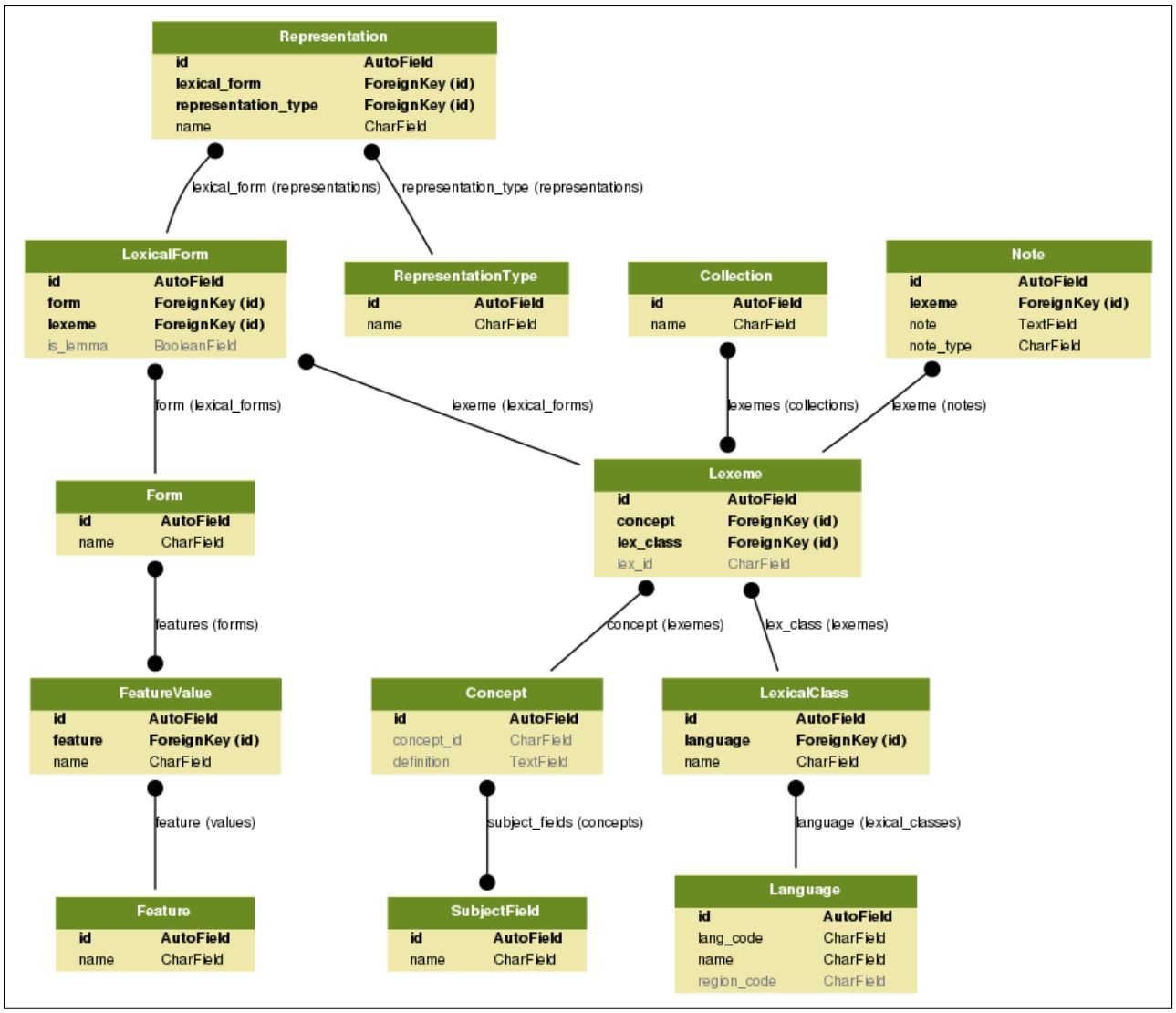

Figure 1: The Normalized Data Model

format results in a dramatic increase in the number of such pairs. Thus, depending on the richness of the data in question, export to a standard format such as TBX is not always as straightforward a process as traversing the database from the appropriate tree root and writing out the corresponding TBX elements for each node. Additional transformations are defined to handle the mapping between the elements of other lexicographical and terminological data models and the merged elements of LexTerm's internal data model. This can result in some loss of information when importing data from an external source format to the LexTerm database and re-exporting again to an arbitrary target format even when all of the relevant data elements from both the source and target formats are theoretically represented in the LexTerm model; in most cases, however, this is expected to be negligible. For example, many types of data elements are allowed to appear at multiple locations in the hierarchy of a TBX termbase, for maximal flexibility and backwards compatibility with other systems; in the LexTerm model, however, any type is allowed to occur in only one location relative to other data elements. As a result, the opinions encoded in LexTerm on what constitutes a single lexeme or what must grouped in a single concept entry may differ from the requirements of any particular import or export situation, but it is nearly always possible to perform the requisite rearrangements entirely automatically.

2. Data Model

2.1. Model Derivation

The key insight to merging lexicographical and terminological data is that a dictionary word sense is equivalent to a termbase concept (Melby & Wright, 1999). Linking these two data elements forms the initial bridge between lexicographical and terminological data models. After that connection is made, the models are further merged by deduplication of shared information in semantically similar elements- i.e., both termbases and dictionaries will contain definitions, and these need be stored only once for each combination of concept and language (in which a concept is instantiated as a lexical item or term), as long as a single definition can be accessed from or pointed to by both a terminological concept entry and a lexicographical word entry.

2.2. Model Normalization

The initial naively dedupli cated model frequently allows for the same information to be accessed by multiple paths; while this is desirable in the public API (and indeed, is necessary to fulfil the goal of presenting common information in either a lexicographical or terminological view), it results in the possibility of data inconsistencies where two paths that should point to the same data in fact lead elsewhere. The final database design thus uses the normalized model shown in Figure 1.

Achieving consistency guarantees through normalization comes at the cost of access efficiency. Reconstructing the non-normalized model to extract LMF or TMF halves requires multiple queries or complex table joins. For example, to obtain all of the representations (e.g., standard orthography, IPA pronunciation, etc.) for all of the terms for a particular concept in a term base, the Concept, Lexeme, LexicalForm, Representation, and RepresentationType tables must all be queried. This results in what can be a severe performance penalty when requesting hundreds or thousands of concepts and triggering thousands or hundreds of thousands of queries against the database.

To improve performance, we use a separate cache of pre-compiled lexeme entries from which concept entries can also be quickly reconstructed. Changes to each table are tracked and trigger updates to any related cached entries. For example, updating a concept for *cat* (animal) will result in identifying and retrieving the lexeme entries for "cat" in English and "ねこ" in Japanese. The cache can reduce latency by up to a factor of 100.

2.3. Model Flexibility

In addition to merging lexicographical and terminological information, we have also attempted to make the system as flexible as possible with regards to the structures of languages about which lexical data can be stored. For reference, TBX achieves the necessary flexibility via the use of XCS files that specify the allowed language names, grammatical numbers, grammatical gender markers, etc., that can occur in any given TBX file. This system as described in the TBX standard is, however, insufficient for our use because of the relatively greater complexity of information that may be required in lexicographical contexts compared to the terminological contexts TBX[1] was developed for, and because TBX provides no means of distinguishing which values are allowed on a per-language basis (such that one cannot specify that, for example, neuter gender is invalid on French words in a file containing entries for both French and German). In the lexicographic world, the Multi-Dictionary Formatter (MDF) format used by SIL Shoebox[2] (based on SIL Standard Format), while being developed as a fully specified machine-readable format (Coward & Grimes, 1995), achieves sufficient flexibility to record any conceivable language largely by virtue of allowing the user to extend the format by defining their own Standard Format data elements with no reference to comprehensibility by any other system. This results in a multitude of idiosyncratic "MDF-derived / MDF-based formats" (Drude & Nevskaya, 2010). The overly-flexible (and thus difficult to validate) design of standard MDF is also known to allow the possibility of data inconsistencies (Drude & Nevskaya, 2010).

While other existing lexicon-management systems such as COLDIC (Bel et al., 2006) are capable of handling essentially arbitrary LMF-compliant schemata, and thus of describing essentially any human language with full validation, they still have the minor drawback of requiring that a full schema be defined up-front. In other words, before you can begin documenting a language's lexicon, you must know beforehand what features will need to be documented. To further streamline the process of documenting new lexical data while maintaining the capacity for automatic validation, we have chosen to consider the language-specific schemata as a part of the mutable data model, thus allowing for new features to be added to a lexicon as they are deemed necessary and for different schemata to be used for data on different languages in the same database.

2.4. The Linguistic Configuration Model

Language specific schemata are defined by a meta-schema known as the Linguistic Configuration Model (LCM), whose basic structure is shown in Figure 2. Linguistic configurations are based loosely on and are analogous in function to XCS files from TBX. LCM data can be dynamically updated just like lexical data and used to validate the lexical data on a per-language, rather than per-file or per-database, basis. The ability to dynamically update language configurations and re-validate lexical information at any time allows the system to accept imports and merge data from multiple different sources that may have used different incompatible schemata by extending the relevant language descriptions as needed to accommodate incoming data. It also expands the system's usability in, e.g., field work, or other situations in which relevant linguistic features may not be known at all prior to data collection.

In order to maintain a balance between flexibility and comprehensibility, the LexTerm LCM initially assumes that:

1. all languages have one or more representational systems (e.g., native orthography, romanization, IPA transcription, etc.)
2. all languages have one or more lexical classes (i.e., parts of speech).
3. words belonging to a class may exhibit any number of morphological forms and any number of grammatical features with finite sets of values associated with that class
4. some form in each class will be preferred for citation (the lemma form).

The names and functions of representations, morphological forms, grammatical features, and values of features may be freely specified, with permissible values for each data element specified by a named enumeration;

[1] While it can be argued that lexicographical information is no more complex than terminological data in general, here we are concerned specifically with the structures that can be encoded in TBX specifically.

[2] http://www-01.sil.org/computing/shoebox/

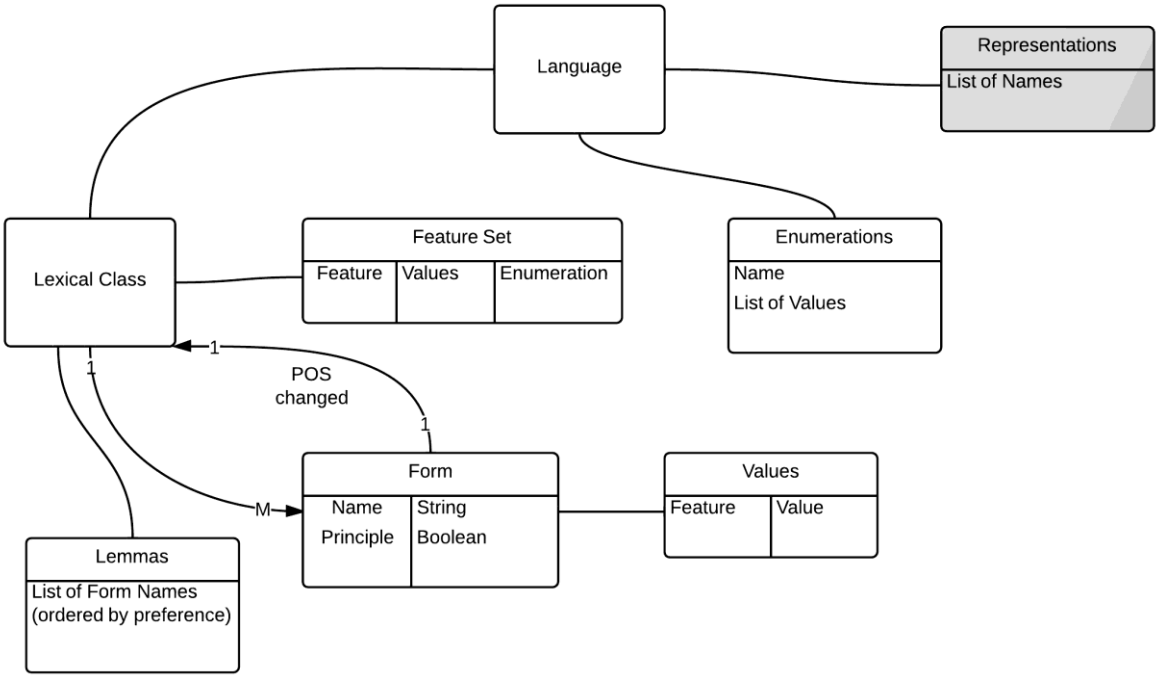

Figure 2: The Linguistic Configuration Model

however, it is intended that LexTerm interfaces will in some way highlight data elements that do not belong to standardized categories, such as those indexed in ISOcat (Kemps-Snijders et al., 2008), to encourage users of the system to adopt standard categories or to submit necessary new categories for standardization.

2.5. Data Validation

By moving schema information into the database itself, we are implicitly giving up any data validation capabilities that otherwise come built-in with the underlying database system (e.g., MySQL, PostgreSQL, or similar). We must therefore re-implement validation of lexical data against LCM schemata. We consider this, however, to be a worthwhile trade-off in exchange for greater end-user flexibility. The process is analogous to validation of an XML document against a arbitrary XSD file, or of a TBX document against an arbitrary XCS file.

In addition to internal validation, LCM data is intended to be used by LexTerm interfaces to determine what fields should be presented to a user for display and/or editing. When displaying an incomplete entry, LCM data indicates what should be present, thus allowing the interface to prompt the user for missing information.

3. Application Architecture

The LexTerm system employs a client-server architecture which allows re-implementation of either the client or the server in different programming languages or on different computing platforms independently of the other. Canonically, this allows a single LexTerm server, managing all of the stored information, to be accessed over the internet by any number of independent LexTerm clients. However, it is intended that a LexTerm server and client may be packaged into a single desktop application with the client and server simply being different processes residing on the same machine. In either case, the same communication protocol and division of responsibilities is used.

Our prototype system has a client implemented in HTML and JavaScript and a server implemented in Python with the Django web framework[3]. It is important to note, however, that the JavaScript client is in no way tied to the Django application. While the two could be hosted in a single location to provide a unified LexTerm web application, they are logically distinct. The client application can be served from any location, including the local filesystem, independent from and dynamically connected to a particular LexTerm server.

3.1. The LexTerm Server

The server is responsible for validating and storing data received from a client and retrieving data in either lexicographical or terminological views in response to requests. Additionally, the server performs mapping between separate client-facing lexicographical and terminological data models and the unified internal data model. In software engineering terms, it corresponds to the Model component in a Model-View-Controller (MVC) architecture (Deacon, 1995). The server provides only a machine-to-machine API, and presents no human-readable interface of its own.

[3] https://www.djangoproject.com/

3.2. The LexTerm Client

The client is responsible for displaying information and (optionally) providing an interface for editing. In MVC terms, it corresponds to the View and Controller components of the entire application. These functions can be realized in multiple ways; e.g., by an interactive graphical interface for browsing and editing entries, or by automated programs for importing from and exporting to files in various external formats. The client-server architecture thus makes it relatively easy to write plugins to handle interfacing with other systems, exporting human-readable PDF or HTML dictionaries, etc., simply by writing a client that can transform the intermediate format of the LexTerm client-server protocol into the desired format or vice-versa, with no changes required to the rest of the system.

To serve the common use case of importing from and exporting to various external data formats, we have investigated the possibility of defining a declarative templating language to automatically handle the requisite transformations, thus allowing new formats to be supported simply by writing a template interpreted by a common client. While we have yet to discover a fully satisfactory solution to this problem, there are several widely-used generic templating formats in existence (e.g., Mustache[4], which is capable of producing XML, LaTeX, JSON (JavaScript Object Notation), and any other text-based document type), and research in that area is essentially independent of core LexTerm functionality.

3.3. The Communication Protocol

In keeping with the idea of internet-based access, the client-server protocol is based on REST (Representational State Transfer) principles (Fielding, 2002), sending JSON-encoded objects (Bray, 2014) over HTTP. The widespread accessibility of libraries for processing HTTP requests and JSON serialization in most popular programming languages makes this a logical choice. JSON was chosen for object serialization due to its relative simplicity, but an alternate XML-based serialization mode is also possible, and may be preferred for ease of transformation into other XML-based formats (such as dialects of TBX).

One downside of the choice to use the HTTP protocol is that it is impossible for the server to notify any client of changes to the database made independently of that client, which complicates the usage of the LexTerm system as currently designed in multi-user concurrent real-time editing situations and necessitates work-arounds such as periodic long-polling[5] (Loreto et al., 2011; Stratmann, Ousterhout, and Madan, 2011). This is only a serious issue if two users are simultaneously editing fields of the same lexicographical or terminological entry; in that case, one user's changes will simply be overwritten. This situation should be rare enough, and the consequences suitably mild, that this is not considered an impedance to the practical use of the system; this situation is, however, an excellent candidate for optimistic concurrency control (OCC) methods which can detect and roll back conflicting transactions that would otherwise be lost and report back to the user for conflict resolution (Kung & Robinson, 1981). The OCC approach avoids the problem of a user acquiring a data lock and then never releasing it, which is especially problematic in stateless web-based systems without persistent connections that could be tied to data locking and unlocking. Completely resolving this issue also touches on the problem of properly merging duplicated entries and represents a potential area for future research.

In accordance with REST principles, the initial protocol design called for a series of HTTP end-points corresponding to each logical resource in the database (dictionary, termbase, lexeme, concept, definition, etc.). As data within either lexicographical or terminological views are inherently hierarchical, this resulted in hierarchically nested URLs, analogous to the usual directory-structured URL schemas used by most websites., but with multiple paths (using either a dictionary or termbase as the root node) to most resources. While conceptually elegant, this interface is, however, difficult to work with from both ends: on the server it requires translating hierarchical paths into the appropriate underlying database calls and often duplicating code paths to map multiple URLs onto a single object, while on the client it often requires jumping through numerous hoops to traverse the tree to the bits of data you actually want.

For ease of use, an alternative 'flattened' API was designed which corresponds much more closely to queries against the underlying database. Due to the previously mentioned caching layer, GET requests against this API can be handled by an Elasticsearch[6] document search server, often avoiding the need to touch the underlying normalized database at all.

4. Future Work

The implementation of a complete, commercially useful LexTerm system requires attention to several administrative concerns in addition to the simple capacity for storing, validating, and retrieving lexical information, some of which require annotations and additions to the basic data model. These include handling user accounts and permissions, term and lexeme life cycle management, and practical aids like duplicate entry detection, fact checking, and collaboration tools. More work also remains to be done on expanding the LexTerm data model to handle a wider range of potentially data elements.

Several expansions to the configuration model are also planned. First, it would be useful to allow internal names for data elements to be specified separately from

[4] http://mustache.github.io/

[3]Long-polling is a means of simulating server-initiated communication by sending a request that is not expected to be resolved immediately, and leaving the connection open until the server has data to respond with rather than timing out after a short interval.

[6] http://www.elasticsearch.org/

their correspondences to standard categories (as specified by ISOcat or another appropriate registry). This would significantly reduce the burden on the client software to identify data elements that lack a correspondence to a standardized definition. It would also represent the first step towards completely separating display names from internal data element identifiers, thus allowing for better localization. However, the process would introduce a non-trivial amount of additional complexity into the data model, as the valid values of user-facing data element names in the linguistic configuration data for the features of any particular language would depend on the other languages also defined in the configuration model and the valid representation types specified for those languages (unless interface language configurations are added separate from and independent of the existing configurations for documented languages).

5. Acknowledgements

The authors would like to acknowledge a generous grant from the Brigham Young University Office of Research & Creative Activities in furtherance of this research. Additional grant funding has been provided by LTAC Global[7]. We would also like to acknowledge the assistance of Kara Warburton and Hanne Smaadahl in providing feedback and guidance on developing terminological data models and typical user expectations.

6. References

Bel, N.; Espeja, S.; Marimon. M.; and Villegas, M. (2006) COLDIC, a Lexicographic Platform for LMF Compliant Lexica. In *Proceedings of LREC 2006, Genoa.*

Bray, T. (2014). The JavaScript Object Notation (JSON) Data Interchange Format. *Internet Engineering Task Force, Request for Comments, 7158*(2070-1721).

Deacon, J. (1995) Model-View-Controller (MVC) Architecture. Retrieved from http://www.jdl.co.uk/briefings/MVC.pdf

ISO 16642 (2003). Computer applications in terminology – Terminological markup framework (LMF). Geneva: International Organization for Standardization.

ISO 24613 (2008). Lexical resource management – Lexical markup framework (LMF). Geneva: International Organization for Standardization.

ISO 30042 (2008). Terminology and other language and content resources – Computer applications in terminology – TermBase eXchange Format Specification (TBX). Geneva: International Organization for Standardization.

Fielding, R. T.; Taylor, R. N. (2002), Principled Design of the Modern Web Architecture. In *ACM Transactions on Internet Technology (TOIT)*, 2(2). New York: Association for Computing Machinery, pp. 115–150

Kemps-Snijders, M.; Windhouwer, M.; Wittenburg, P.; and Wright, S. E. (2008). ISOcat: Corralling Data Categories in the Wild. In *Proceedings of LREC 2008, Marrakech*.

Kung, H. T.; Robinson, J. T. (1981). On optimistic methods for concurrency control. *ACM Transactions on Database Systems (TODS)*, 6(2), pp. 213–226.

Loreto, S.; Saint-Andre, P.; Salsano, S.; and Wilkins, G. (2011). Known issues and best practices for the use of long polling and streaming in bidirectional http. *Internet Engineering Task Force, Request for Comments, 6202* (2070-1721).

Melby, A.; Wright, S.E. (1999). Leveraging Terminological Data for Use in Conjunction with Lexicographical Resources. Proceedings of the Fifth International Congress on Terminology and Knowledge Engineering 23-27 August 1999, Innsbruck, Austria, 544-569. Vienna: TermNet.

Oliver, A.; Vàzquez, M.; and Moré, J. (2007). Linguoc lexterm: una herramienta de extracción automática de terminología gratuita. Translation Journal, 11(4).

Stratmann, E.; Ousterhout, J.; and Madan, S. (2011). Integrating long polling with an MVC framework. In *Proceedings of the 2nd USENIX conference on Web application development.* USENIX Association, p. 10.

7. Appendix

7.1. Reference Implementations

Source code for reference implementations of the LexTerm client and server is hosted publically on GitHub.

LexTerm Client:
https://github.com/LexTerm/LexTermClient

LexTerm Server:
https://github.com/LexTerm/LexTermServer

7.2. API Documentation

Detailed documentation on the hierarchical API, including specification of routes and examples of JSON-formatted request and response bodies can be found at http://docs.lexterm.apiary.io/ .

Interactive documentation of the 'flattened' API (which allows experimentation with raw HTTP requests and direct inspection of the resulting data structures) can be found at http://lexterm.gevterm.net/api/ .

[7] http://www.ltacglobal.org/