A Hypergraph Data Model for Building Multilingual Dictionary Applications

Louis Lecailliez, Mathieu Mangeot

A Hypergraph Data Model for Building Multilingual Dictionary Applications

Louis Lecailliez[1], Mathieu Mangeot[2]
[1] NLP Centre, Masaryk University, 602 00 Brno, Czech Republic
[2] IG, Université Savoie Mont Blanc, 38400 Saint Martin d'Hères, France
E-mail: louis.lecailliez@outlook.fr, mathieu.mangeot@imag.fr

Abstract

A non-negligible part of learners of an East Asian language have an interest for another tongue from East Asia that may share some common areal features such as the use of Chinese Characters and limited word morphology. It makes sense to build for this niche a dictionary application that provides multiple languages in one bundle and allow easy navigation between them and in the lexicon. This paper describes a data model, a generic dictionary application architecture and a prototype that fit this use case.

The task of merging lexical resources with vastly differing micro-structures and concerns is complex. Even more so is to update it to include new data types or languages after release. In this regard, lexical networks are appealing: they solve the problem by exploding the micro-structure into data nodes and explicitly linking them with edges that can be discovered and traversed automatically. One of these approach, The Linked Data, is gaining traction in lexicography. It is however plagued with issues within the Resource Description Framework (RDF) that backs it. Most notably, the lack of three-valent relationships make is harder than it should to handle the Chinese writing system.

We therefore came up with a simple and consistent hypergraph data model whose main features are: hyperlinks (links of arity greater than two), a flat type system (non-ontological lexical network) and annotations for both node and link instances. We propose a generic application architecture based on this model and illustrate it with a working mobile application. The user interface is constructed from independent components, allowing displaying of complex data while increasing further its updatability and maintainability. We use data from the Revised Mandarin Chinese Dictionary of the Ministry of Education of Taiwan, augmented with open-data Japanese readings to fed the prototype.

Keywords: graph dictionary; RDF; mobile lexicography

1. Introduction

It is quite common for students of East Asian Studies to have interests for another East Asian language: some universities actually provide double major degrees to meet this

demand. A lot of vocabularies of Sinitic and Sinoxenic[1] languages come from a common ancestor so they are related in meanings and pronunciations. It then makes sense to present learners with similar words of other languages he or she is learning while browsing an entry in an electronic dictionary. However, current mobile dictionaries are mostly distributed as independent applications for each language pair.

The task of integrating different dictionaries for a given language is not easy. Yet, creating a multilingual dictionary is even harder. This is because each one has its own microstructure that doesn't necessarily play well with others. Moreover, the transposition of paper to electronic dictionary resulted in digital resources such as XML or databases whose structure is still modeled on what is displayed to the end user (Polguère, 2012). Hence, the abstraction level needed to easily merge and use in these resources is not provided.

1.1. Graph-based Data Models

[il faudrait parler de polguère 2014? qui distingue deux types de réseaux lexicaux]

Researchers started to rethink the modeling of dictionary itself and came up with graph-based models. On the side of online dictionaries, the Semantic Web and the Linked Data — two related but distinct concepts — are frameworks that provide a graph model to piggyback on. This is a promising way of building dictionaries and there are ongoing research projects leveraging these technologies (Declerck, 2015). However, for the reason that Semantic Web relies on Internet connectivity to reach its full potential, it is by definition not suitable for an offline dictionary even if its technology could be used in an embedded way. In addition, using RDF creates its own share of problems because of its complexity.

This situation motivates us to create a data model that tries to capitalize on the essence of what makes the Semantic Web a potentially powerful technology — its underlying graph model — while not being limited in our implementation by the inherent complexity of RDF-based frameworks like Lemon (McCrae et al., 2011) for which support of East-Asian languages is lacking (Lecailliez, 2017a). We define a lightweight, simple and consistent graph model and use it to produce what looks like a classical dictionary entry from a graph instance.

1.2. Handling the Chinese Writing System

We aim to create a data model in the simplest way that allows us to develop an application that is visually similar to an existing Japanese-French (or Mandarin-English) dictionary application. It means we need to support the basic model for bilingual

[1] For a definition of Sinoxenic languages, see (Hashimoto, 1973).

dictionaries in these languages: triples containing the most common written form in the source language and its phonetic representation, and a translation.

This particular case arises from the Chinese script where the phonetic information is not presented in a clear way in the characters themselves. Moreover, "[o]ften a large number of pronunciations exists for the same sinogram […] In such cases, the variant pronunciations may indicate multiple meanings of the graph" (Mair, 1996) which leads to the need of triples. They include a phonetic aid instead of being minimal meaning/meaning pairs that are the basis of other language combinations such as English-French.

The problem actually exists in two instances: at the character level and at the word level (Lecailliez, 2017a). It is even more significant in Japanese where it is very common for a written form to have different readings associated to different meanings. This particular situation is also why the Chinese characters are a lexicographic object per se and needs to be referenced in dedicated bilingual dictionaries.

Owing to the fact that a given triple made of a character or a word, a pronunciation and a meaning cannot be decomposed in three pairs without losing information in the process, we need to deal with the situation with appropriate means. For example, the KanjiDict2 (2003) is an open source dictionary for kanji[2] that has a provision to encode different readings/meanings of groups for a given kanji entry.

1.3. Structure of the Article

This article is organized as follows: in the first section, we present the issues we faced with the relational model and with RDF as graph technology to model our intended data. Next, we expose the graph model we came up with to address the previous problems. In the following part, we explain in detail what test data were used and how we transformed it. The final section before conclusion is devoted to the architecture of a dictionary application built on top of the model. It is illustrated with screenshots from an actual prototype mobile application.

2. Issues with Existing Data Models & Frameworks

2.1. Issues with the Relational Model

The current major paradigm for organizing software data is the relational model. It is generally used in conjunction with a software layered in three main parts: the data layer which communicate with the database system to retrieve or store data, the business layer which contains the core logic of the software and the presentation layer that

[2] Chinese characters used in Japanese.

display content to the end-user[3]. Dictionary software do not escape to this paradigm and are typically implement this way too.

This model proved its robustness for implementing a wide array of programs but it causes frictions in our case: we want to implement an evolving dictionary which will include lexicographic data from different languages as they become available. Merging two dictionaries can already be a problem if their micro-structure is complex but as least they are known ahead of time.

To accomplish this vision, successive multiple changes to the data schema will be required. The main issue with the relational data model and the three-layered software approach is that each modification of the data schema requires a database migration, an update of the business logic and changes in the presentation layer. In short, every layer of the application is impacted. Also, the presentation layer can grow very large if there is a lot of data to display and user settings to take in account, in addition to the various null or empty checks. The work on the business layer is mostly wiring but can brings its own share of bugs.

Thus, we need a model which allow a dictionary to change his micro-structure frequently with the less possible impact to any software layer. In this article, we describe a data model that give us such agility. A change in the proposed model would (1) not affect the data layer at all nor (2) the business logic and (3) allow composable user interface to be written and maintained.

2.2. Issues with RDF

Data model based on the RDF framework have been proposed to write graph dictionaries. One of such model that gained traction is the Lemon model, which is continued by the OntoLex model (W3C, 2016a). While these models have issues of their own to encode Japanese dictionaries (Lecailliez, 2017a) they also contain every flaw of the RDF framework they are built upon. The main problem is that while RDF encoding of data result in a graph, it is not a general graph modeling tool. Very simple graphs such as A—[distance:50]—>B[4] cannot be represented directly in RDF because it imposes constraints such as the use of "concept" nodes represented by URIs and does not provide an edge annotation mechanism (Lopes et al., 2009). These are basic features of a graph and are provided by graph-focused product such as Neo4J (Robinson et al., 2013). It leads to a situation where models are created and modified to fit the RDF framework more than to solve the problem at hand.

---

[3] See https://msdn.microsoft.com/en-us/library/ee658109.aspx?f=255

[4] That is, an oriented graph with two nodes respectively labeled A and B, joined by a relationship with a property named "distance" that have the value 50.

2.2.1. Blank Nodes

The RDF framework on which most part of Linked Data movement is developed doesn't support out-of-the-box relationships of arity different than 2. In order to model 3-valent link between nodes, we need to rely on intermediate anonymous resource also known as blank node (W3C, 2006b). This is a problem for two reasons: first it increases complexity of modeling by introducing one additional node and two additional relationships for each triple relationship we want to implement in the graph. Secondly, it breaks the traversal mechanism uniformity: going from a node to another in dual relationships requires one link navigation while it needs two when more than two nodes are involved.

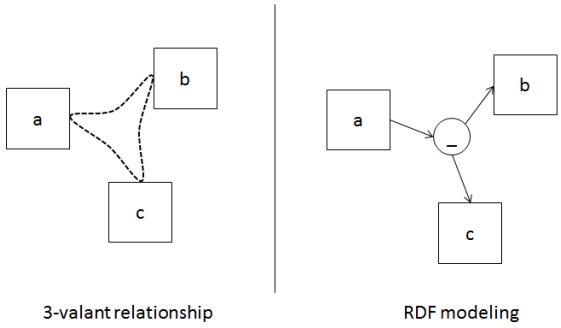

Figure 1: Hyper-Edge and RDF Links Comparison

The Figure 1 illustrates the difference in modeling directly a direct 3-valent relationship against its RDF representation, which feature a blank node labeled "_". Note that the relationship of arity 3 is non-oriented. Therefore, a more accurate RDF modeling actually involves 6 properties[5] instead of 3.

2.2.2. RFD Literals

RDF literals pose another problem: they are simpler to use and interpret than URI (that makes up resource nodes) and are used to store actual content but they are by essence different. In particular, a property cannot take a literal node as subject, in other words it cannot be the first element of a relationship. This cause the problem of not being able to reuse literal nodes as the source of a relationship. A typical solution is to make use of an intermediate blank resource node between the initial subject and the object, that acts as a proxy for the object. The intermediate node can then be the subject of newer relationships if needed. This cause another problem: because a blank node is anonymous, it is still hard to reuse to express other statements.

[5] An RDF triple is an ordered triple (subject, property, object) where the property is a link between two nodes.

The Lemon framework (McCrae et al., 2010) makes use of these kinds of blank nodes. Figure 2 illustrates the use of a blank node. In the Turtle serialization of RDF, a blank node can be instanced using a couple of square brackets. The `:nihongo` resource is linked to an anonymous (blank) node by the property `lemon:canonicalForm`.

```
:nihongo lemon:canonicalForm [
  lemon:writtenRep "日本語"@ja-Jpan ;
    isocat:transliteration "にほんご"@ja-Hira ;
    isocat:transliteration "nihongo"@ja-Latn ] .

isocat:transliteration rdfs:subPropertyOf lemon:representation .
```

Figure 2: Example 15 from the Lemon Cookbook

We remark that the leaf data still makes use of a literal, which leads to example 16 of the Lemon Cookbook where two sub-properties inheriting from the property *isocat:transliteration* are defined to handle the case of multiple transliterations. This is an example of how a data model must be extended to deal with the shortcomings of RDF instead of being needed to address the problem at hand.

2.2.3. Relationship Reification

Finally, the last problem we identified with RDF modeling is how complex it is to reify relationships. A reference book on RDF by (Powers, 2003) worded a section on the subject “Reification: The RDF Big Ugly”. Reified relationship appears in recent works on lexical graphs such as (Polguère, 2012). The ability to link a relationship instead of one of its constituent enables an interesting and powerful way to represent composed lexicographic objects such as Chinese proverbs, where each word that makes it up can be linked to its definition in context.

All the difficulties that were mentioned to model our core domain with RDF-based technologies motivate us to come with a simpler model that can be implemented with a few classes in an object-oriented language. The graph model that is described in details in (Lecailliez, 2016) is composed of three core notions: nodes, edges and annotations.

3. The Graph Model

This section describes the data model we propose to construct highly modifiable dictionary software. It is based on the notion of hypergraph. A similar system is described by Williams (2000, 2001) and is called the associative model of data (AMD). Despite their resemblances, this model is not directly built on AMD of as the work was initially unknown to the authors. Identical core concepts are the use of only two kinds of entities (item/node and links) and the ability of a relation to link other relations in addition to nodes. There are however key differences such as the arity of links (always three in

the AMD but unconstrained here) and the absence of relationship between types in the model we propose. Additional properties geared towards implementations which are not featured in the AMD exist on types presented below.

3.1. Type System

The data model we propose is built on the notion of hypergraph: it contains heterogeneous types of vertices (nodes) and edges (links/relationships). Each of these two kinds of graph objects is associated to a given type that serves as metadata. The basic properties that define types are listed in the Figure 3 below. There are no predefined types of vertex or edge.

Property Name	Property Usage
Name	Human readable name
Identifier	Unique machine identifier (GUID)
Description	Human readable description
Object Kind	Vertex or Edge
is_oriented (edge only)	Boolean value
is_direct_content (vertex only)	Boolean value

Figure 3: Type Properties and Example

The Figure 4 and Figure 5 below respectively illustrate a vertex type that represents a Chinese character and an edge type that can link two characters. The relationship expresses the fact that the destination vertex content is the traditional form of the origin vertex content.

Property Name	Value
Name	Chinese Character
Identifier	023d04df-fb49-4f21-95c8-057139e7e0a3
Description	Represent a single sinogram attested in a given language.
Object Kind	Vertex
is_oriented	*(not applicable)*
is_direct_content	True

Figure 4: Example of a Vertex Type that Model a Chinese Character

Property Name	Value
Name	Traditional Form
Identifier	8e260c3c-98c4-47f5-96cb-4c34880d0e24
Description	Indicate that the target vertex is the traditional form of the source vertex.
Object Kind	Edge
is_oriented	True

is_direct_content	*(not applicable)*

Figure 5: Example of an Edge Type

All the type properties are explained in detail in the next section (3.2). In addition to type properties, nodes also carry instance properties that are described in a dedicated section (3.4).

3.2. Type Properties

The name of a type is a non-null and non-empty string that contains at least a printable character. This information is destined to human lexicographers and developers; it thus should be readable, understandable and related to the node content. It also should be short as it may be used in various places of the dictionary editing workflow or the development of a client application. The name of a graph object must not be used for type checking by software because equal names may exist in a project. This is the purpose of the type's unique identifier (see below).

The description field contains a long description of the node purpose and content encoding hence solving ambiguities that may arise because of short type names. This is the appropriate place to put information such as the used transcription system, the encoding of multimedia content, the suitability of an expression, and so on. The description will not be used to do any automatic processing of the content of the instances of the type.

Each type is uniquely identified by a GUID. This field must be used by program testing if two types are identical. Such an identifier is the identity of a type: if modifications are made to a type (for example by changing the convention used to write a word) of a released project, the new modified type must use another identifier while the old one retains the existing GUID. Types are thus immutable.

The object kind property exists to allow an implementation to handle both kind of types with a unique class; this additional information may be used for differentiated treatment of edge and vertex types. Two additional properties exist as they come handy in the prototype implementation: is_oriented and is_direct_content. A vertex type with a is_direct_content property set to true indicate the node value can be displayed directly to the end user of a dictionary application. Otherwise some additional processing might be required (see section 3.4 for an example of such case).

Types can be shared across projects: a type defined in a project can be used in another if needed, easing data sharing between dictionaries using this data model. This stresses the importance of a relevant name and description for a type because they are targeted at current and future lexicographers that will work with the encoded data.

3.3. Relationships

Links between nodes is the second kind of objects populating the graph. They indicate that two or more graph objects are linked by a given relationship denoted by its type. Graph objects are node and edge instances. The model currently defines three kind of edge: simple (non-oriented) edge, oriented edge and hyper-edge. The difference between simple and oriented links is semantic: an oriented instance of a relationship means it has meaning in only one direction. If the reverse relationship exists and is meaningful the *Description* field should indicate so that the client implementer knows they can use it backwards.

Hyper-edge — a link between more than two objects — is mainly motivated to enable representation of information of East-Asian dictionaries where Chinese characters are used. In these languages the minimal bilingual entry is made of a word of the source language written in two forms and a word or expression of the target language. One of the graphical source forms contains Chinese Characters while the other is written in a script not ambiguous about its pronunciation. For example, a minimal triplet entry for the word “birthday” in Japanese is (誕生日, たんじょうび, birthday) or (誕生日, tanjōbi, birthday) if we use the Latin script to transliterate the word. The choice of a script or transcription system for the last part of the triple must be consistent and will be mainly motivated by the target audience of the dictionary.

Finally, any of these kinds of links can be established between any kinds of graph object: a node may be linked to an edge; or a link may be created between links. The mechanism of links between links allows representation of more complex phenomena such as Chinese proverbs where each of their constituent words is linked to the exact meaning it has in the context. However, it has the drawbacks of complexifying implementation: loading data from files need to be done in a recursive way and usage in code is also less developer friendly.

3.4. Node Instances & Atomicity

In addition of a type, node instances aggregate a bunch of values, mainly *Content* and *Language*. Internal value of a node that is stored in the *Content* property cannot be referenced by another node or by a relationship but is visible to display components. Internal content of a node cannot reference a node or edge instance. Nodes are hence atomic values in respect to the graph. That is an important property because (1) it allows a node to be safely removed from the graph without leaving dandling pointers and (2) any link between data must be explicitly declared with a relationship instance that allows its automatic discovery and processing.

Atomicity allows complex content to be stored in vertices. For example, information about vowel devocalization in a Japanese word written in kana[6] can be added with a binary mask. Each bit is associated with a kana, with the least significant one being matched with the first character of the string. Thus がくせい/2 indicates that く should be devocalized[7]. The Figure 6 illustrated the handling of this encoding: the kana containing a devocalized vowel is rendered in gray instead of white.

Figure 6: Rich Formatting Display of Japanese Vowel Devocalization

Node types whose content contains complex data that must be decoded by a dedicated component have their *is_directed_content* property set to *false* to prevent accidental unfriendly display to the user. This mechanism allows embedding of multimedia content (cf. 5.3).

3.5. Annotations

The last concept included in the model is annotations. Not every situation benefits from being modeled as a node or an edge. Typically, information like part of speech would have a huge number of linked nodes in a real dictionary and will not play nicely with an automatic display system such as the one presented in the next section. In addition, such very highly linked nodes would radically change the topology of the graph.

An annotation is a triplet (namespace, key, value) of strings that can be associated to any object graph instance (node or edge). It offers a great flexibility of modeling but because annotation have no associate type and cannot be annotated, this comes at the cost of a lesser automatic processing capabilities. The namespace part of an annotation allows a graph to be annotated with multiple properties with the same name. For example, two *freq* properties indicating frequency of a word but computed from two different corpora can annotated the same vertex; they are however kept distinct by the use of a different namespace.

Moreover, annotations provide a way to declare fined grained metadata: for instance, source or licensing can be specified at the node or edge level. It is also a mean to encode metadata for ordering nodes for display: (Lecailliez, 2017b) details such a technic and provides an implementation alongside illustrated examples.

---

[6] Hiragana and katakana, collectively known as kana, are the two Japanese syllabaries.

[7] 2 is 0010 in binary, so applied backward to the word, が = 0, く = 1, せ = 0, い = 0.

4. Dictionary Data

In addition to code, data is necessary to have a working prototype. We used various existing lexicographic data available on the web to build test data sets. One of the test set is built upon existing open source dictionaries. In this section, we describe these resources and how we transformed them to fit the graph data model. Other data sets may be featured in screenshots of section (5.0) but are not detailed in this paper.

4.1. Chinese

We use two existing open-source dictionaries to feed the application prototype. The first is the CFDict dictionary, an initiative launched by the website *Chine Informations* (2001). Content is available under Creative Commons 3.0 BY-SA since 2009. By 2016, the authors disappointed by license breaches and the lack of community contributions pulled back most of its content. From about a year only content added by third-party was distributed, until the end of October 2017 when more content was released again. We use as our Chinese dataset a subset of a file dating back from 2011 that was recovered from an earlier project instead of the file publicly available to download.

4.2. Japanese

We originally wanted to use the KanjiDict (2003) file which is a dictionary of Chinese characters and their usage in Japanese. In its format it defines a way to make triple relationships between a character, a reading and a meaning. This is the exact situation our model targeted but this way of structuring data is actually not used in the file. Instead, for each character a list of readings and a list of meanings are given, without any link between them. We checked if this situation occurs for a significant number of entries and there is indeed no entry that features at least two distinct groups of reading & meaning[8].

The most well-known and accessible dictionary resource for Japanese to European languages is JMDict (Breen, 2004) that is distributed as an XML file. Until data from the Jibiki.fr project (Mangeot, 2016) is released, it is overwhelmingly used by dictionary software and tools for the Japanese-French language pair as it is the only freely available resource for Japanese-French.

4.3. Ideographic Description Sequence

[TODO]

4.4. Modeling as Graph

[8] For some entries, the situation stem from having only one group of reading and meaning anyway.

We transform data extracted from the original XML files to a custom XML file. Before switching to a standard format such as GraphML (Brandes et al., 2013), a temporary format fitting the application needs was created first as a text format, and then evolved into a XML one. It bears heavy similarity with the GEXF format from the graph visualizer tool Gephi (Bastian et al., 2009).

Each element from the Chinese dictionary is represented by a triple of nodes linked by a relationship: *ChineseWord*, *ChineseWordSpelling* and *FrenchWord*. An additional type *Sinogram* was added; its instances are populated by splitting content from *ChineseWordContent*. This is an example of a data type not present explicitly in the original data that is created to increase the graph connectivity. The graph formed by the types are represented in Figure 7 (a). A relationship *Translation* establish triples made of a word from French and Chinese and a romanization of the Chinese word. The link *IsInCompound* connects a *ChineseWord* to a *Sinogram* used to write it.

The Japanese data was processed in a very similar way. The main difference is discarded data, as JMDict features richer micro-structure than CFDict. In addition, the Japanese language contains a lot of words multiples reading and meaning, and only one tripe was constructed here from each entry. It is visible from Figure 7 (b) that the structure is nearly the same of the graph from Chinese data.

Finally, data from the IDS were processed by the use of a single type *Sinogram*. This is a simplification, as some entries of the file does not encode a complete Chinese character. Sinograms are linked by an oriented *Compose* relationship.

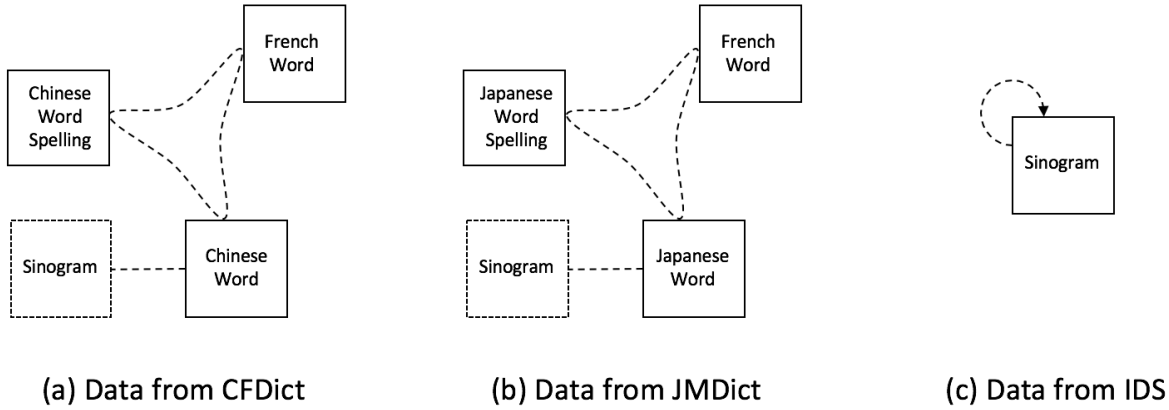

Figure 7: Graph Constructed from each Data Source

Abstractly, the final step is to merge each of the graph in a unified graph. The resulting schema is given in Figure 8. An additional *Use Same Sinograms* relationship is

introduced in the final graph: it links *ChineseWord* and *JapaneseWord* vertices that are written in the same way; for example 日本 (Japan) and 中国 (China) are written in the same way in two languages. This is an instance of a relationship not present in the original data that is added to increase the connectivity of the graph. Note that the real-world data were generated in a different way: once the CFDict was build, it was extended with IDS data and then expanded a third time with data from JMDict.

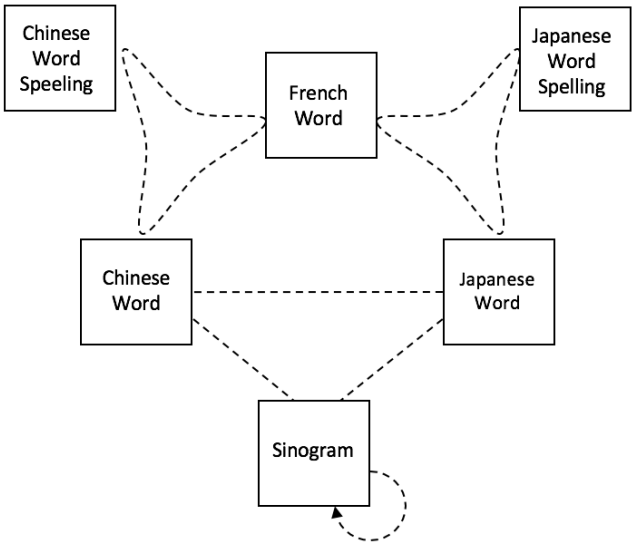

Figure 8: Graph Obtained from Merging the Three Data Sources

No annotation is used to simplify the example, but adding them is possible. The graph can be extended further be adding new types and new relationship. When only text is added, such modifications do not require change in an application structured as the one presented in the next section.

5. Application Architecture

To test the viability of the model an implementation was made in C#[9]. The core of the project is a C# library that encode the graph in memory and provide a de/serializer to process XML files. The library is used in multiples clients, the more advanced one being a Windows Phone application, from which originated the screenshot in the section. The architecture of this application is detailed here.

5.1. Application Bundle

[9] C# is an ECMA standardized object-oriented programming language with commercial and open-source implementations.

At the highest level of description shown in Figure 9, the application package is made of three parts, one of them being mandatory. The required part (2) is composed of (a) the domain model implementation, (b) the page composer which dynamically generates dictionary pages and display component for (c) vertex and (d) relationship. A file (e) or code section declares the associations between graph types and display components.

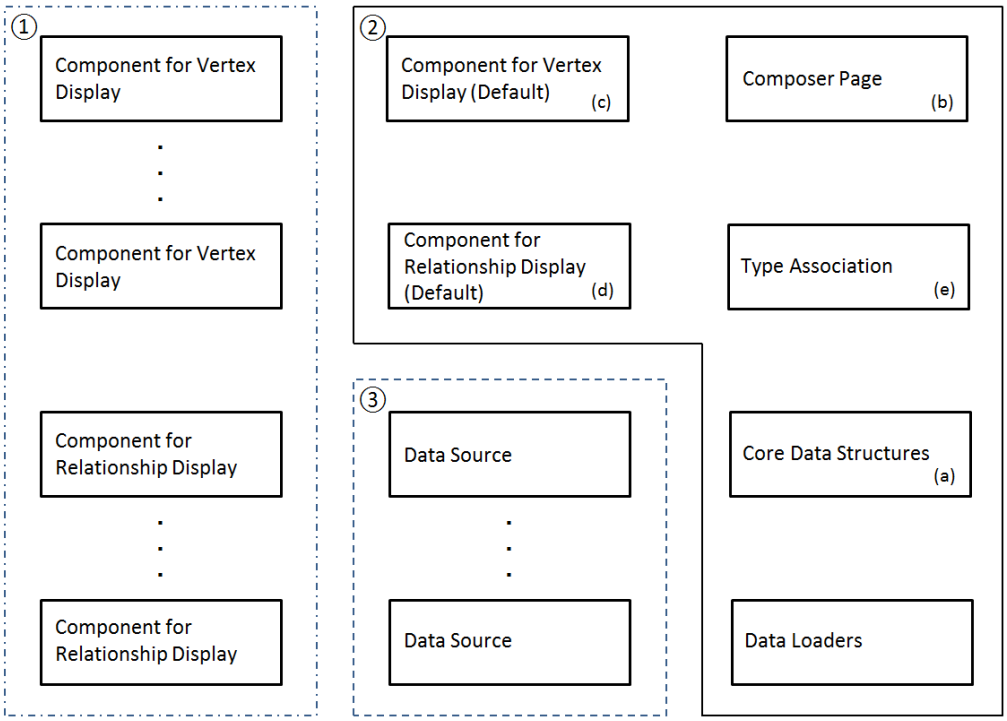

Figure 9: Application Package Structure

Types for which no custom component (1) for visualization is declared are handled by the default (c) (d) components. This method allows modular development and addition to the dictionary; user interface is not declared in a single hard-to-maintain file. Finally, data files are bundled (3) with the application or can be retrieved from the network.

5.2. Page Generation

Dictionary pages are generated on the fly by the page composer for a given “headword vertex”. The head vertex content is displayed at the top of the page by the default vertex presenter or a custom interface component if its type is associated with it. All neighboring nodes are grouped by relation type and each group is displayed by an instance of the default edge presenter or a custom component.

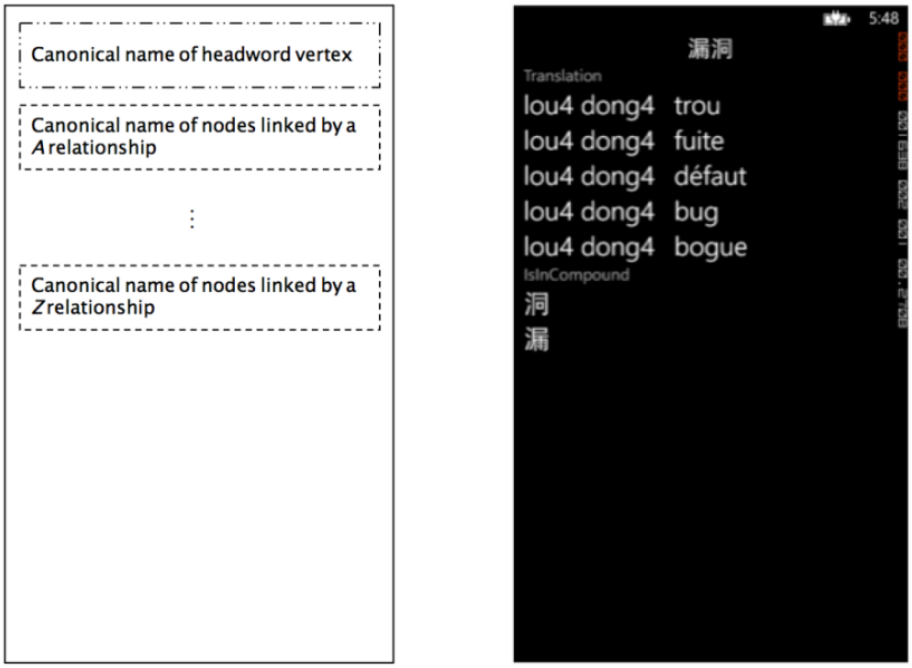

Figure 10: Generated Page Structure and Example

The behavior is the same with any number of nodes and relationship types. New types can be added to the graph at runtime, it does not affect nor break the page composer. New data are handled by the default display component if their content is tagged are direct. These are the two key points that allow the statement (2) made in section (2.1) concerning the immutable business logic of the application. Of course, this does not mean that the page composer cannot be modified if needed. On the contrary, the one used here is very simple and a more complex one could be written to modify the order in which relationship blocks are displayed for example. But this change is independent of the modifications made to the underlying data schema.

In addition to displaying vertex content, the default relationship presenter implements a navigation behavior: a click listener is added to each text data displayed, which changes the current headword with the target node associated with the data being clicked. By changing the current vertex head, a new page is generated and displayed. The user can navigate through the whole graph using this mechanism. The Figure 11 illustrates how a click (simulated by a red circle) on any text data leads to another page.

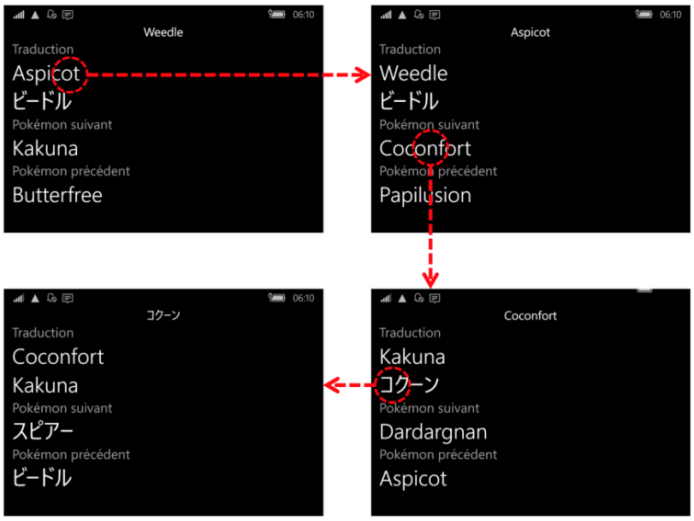

Figure 11: Navigation between Nodes

5.3. Custom Display Component

Finally, the composer can make use of self-contained display components that interpret a node content to produce a rich formatting. The formatting can include colors, multimedia elements and custom click behavior. Figure 12 below shows a page that uses two different components for data display instead of the generic one.

Figure 12: The Use of Custom Component for Richer Interface

The first custom component displays the content of nodes reached by the *Translation* relationship in a similar way of the default display, except it appends the language of the node after the data itself (in gray and between parentheses). The second is used to display an image related to the node reached by a *HasImage* link.

Default and custom display components is the way the statement (3) of section (2.3) is hold true. The final user interface is broken down in various and totally independent software classes. Additional components can be added without any need to touch any existing others when a new data type is added to the dictionary. If this type is just a string that can be directly displayed, the creation of a component is not needed at all.

5.4. Fixable Display Issues

The approach presented offers the possibility of exploratory navigation within the dictionary in a way that is not offered by traditional applications: every displayed information can be touched to display more information about it be changing the headword vertex. It however also raises a few issues, that are fixable within the framework.

First, there is no display order specified for relationships. The headword vertex is always displayed at the top of the page, but the order of groups of vertices reached by links is unspecified. A simple yet effective solution could be to specify the order in which known relationships should be ordered in respect to a vertex type in a configuration file. That file could be updated when new types are added or deleted to the graph.

Secondly, the graph connectivity can have a bad effect on display. Some nodes are connected to a huge number of vertices so their display can be overwhelming for the user. An example of the problem is given in Figure 13 with the Chinese character 口 (kǒu$_{zh}$, *hito*$_{jp}$, mouth) which is used in a huge number of compound words.

Figure 13: Display Issue for a Node with a High Outdegree

There is no obvious fix for this case: frequency could be taken into account for example, but this requires the data to be available and an informed choice is still hard to make automatically as a very infrequent word can still be very relevant to the user. Node pruning can be achieved by different algorithms that take various information in account but certainly should be done statically if it uses resource intensive computation. The best solution here in term of quality of entries is to manual annotated the most interesting entries, display a limited number of linked vertices and provide a button to load more at user's will.

6. Conclusion and Future Research

6.1. Conclusion

The present paper has introduced a lightweight hypergraph model that alleviates the difficulties that existing frameworks based on RDF impose for modeling our target language dictionaries. Two issues of RDF are addressed: the need for an annotation system and the reification of relationships. Moreover, the graph traversal is made consistent for relationship of any arity.

In addition, it lays the way to create dictionary applications which can be updated to support more languages more easily than those implemented in a more traditional fashion with a relational data model. In particular, the prototype we built on the described abstract application architecture is robust to changes in the dictionary data: new types

(which encode part of the dictionary micro-structure) can be added or deleted without breaking the application. It is even capable of displaying some of this data without further modification. Advanced displaying can be added merely be adding a new software component; no change is required to any existing user interface files.

The annotation system makes it easy to add information in a compact way to existing nodes or edges of the graph. Information added may be lexicographic data per se (such as part of speech) or metadata. This feature can by leveraged to include metadata like source references or licensing at the node and edge level.

Finally, relationship that may recursively link nodes or relationships allows expressing complex lexicography situations. It could allow for example the user to navigate to the meaning of a word in the context from a higher level lexicographic construct such as a Chinese proverb. The future work will be done to provide a demonstration of this capability.

One issue though is the lack of a constraint system for relationships which could hinder the discovering capabilities of a client processing the graph (for example a program building a SQL database from a graph file). This point will be addressed in a future revision of the model.

Acknowledgments. This paper was partially written with the support of the MSMT-10925/2017-64-002 scholarship grant from the Czech Ministry of Education, Youth and Sports.

7. References

Bastian M., Heymann S., Jacomy M. (2009). Gephi: an open source software for exploring and manipulating networks. In *International AAAI Conference on Weblogs and Social Media*.

Brandes, U., Eiglsperger, M., Lerner, J., Pich, C. (2013). Graph Markup Language (GraphML) In Tamassia, R. (ed.) *Handbook of graph drawing and visualization*. CRC press.

Breen, J. (2004). Jmdict: A japanese-multilingual dictionary. In *Proceedings of the Workshop on Multilingual Linguistic Resources*, MLR 04, pages 71–79, Stroudsburg, PA, USA. Association for Computational Linguistics.

Breen, J. (2013). Jmdictdb - online development and maintenance system for the japanese-multilingual dictionary. In Kwary, D., Wulan, N. et Musyahda, L., editors: *Proceedings of AsiaLex 2013*, pages 188–199, Indonesia. Airlangga University Press.

Chine Informations. (2001). Accessed at: https://chine.in/mandarin/dictionnaire/ CFDICT/ (20 May 2017).

Chudy, Y., Desalle, Y., Gaillard, B., Gaume, B., Magistry, P. et Navarro, E. (2013). Tmuse: Lexical network exploration. In *Sixth International Joint Conference on Natural Language Processing*, IJCNLP 2013, Nagoya, Japan, October 14-18, 2013, pages 41–44.

Declerck, T., Wand-Vogt, E. et Mörth, K. (2015). Towards a pan european lexicography by means of linked (open) data. In Kosem, I., Jakubícek, M., Kallas, J. et Krek, S., editors: *Proceedings of eLex 2015. Biennial Conference on Electronic Lexicography (eLex-2015), electronic lexicography in the 21st century: Linking lexical data in the digital age*. Trojina, Institute for Applied Slovene Studies/ Lexical Computing Ltd., Trojina, Institute for Applied Slovene Studies, Ljubljana.

Gader, N., Lux-Pogodalla, V. et Polguère, A. (2012). Hand-Crafting a Lexical Network With a Knowledge-Based Graph Editor. In Committee, T. C. O., editor: *Third Workshop on Cognitive Aspects of the Lexicon (CogALex III)*, pages 109–125, Mumbai, India.

Gao, Y. (2013). On the appification of dictionaries: From a chinese perspective. In Kosem, I., Kallas, J., Gantar, P., Krek, S., Langemets, M. and Tuulik, M., editors: Electronic lexicography in the 21st century: thinking outside the paper. *Proceedings of the eLex 2013 conference*, 17-19 October 2013. Ljubljana/Tallinn: Trojina, Institute for Applied Slovene Studies/Eesti Keele Instituut.

Gephi. (2008). Accessed at: https://gephi.org/gexf/format/. (20 May 2017)

Hashimoto, J. M. (1973). Current Developments in Sino-Vietnamese Studies. Journal of Chinese Linguistics, 1-26. Accessible at: http://www.jstor.org/stabe/23752818

Jibiki Project. (2001). Accessed at: http://jibiki.fr. (20 May 2017).

JMDict. (1999). Accessed at: http://www.edrdg.org/jmdict/j_jmdict.html. (20 May 2017)

KANJIDIC2. (2003). Accessed at: http://www.edrdg.org/kanjidic/kanjd2index.html. (20 May 2017).

Koide, S., Takeda, H. (2013). Rdfization of japanese electronic dictionaries and lod. In Chiarcos, C., Cimiano, P., Declerck, T. et McCrae, J. P., editors: *2nd Workshop on Linked Data in Linguistics: Representing and linking lexicons, terminologies and other language data.* CEURS.

Lopes, N., Zimmermann, A., Hogan, A., Lukácsy, G., Polleres, A., Straccia, U., & Decker, S. (2010, June). RDF needs annotations. In *W3C Workshop on RDF Next Steps*, Stanford, Palo Alto, CA, USA.

Lecailliez, L. (2016). Pour une modélisation de dictionnaires de japonais sous forme de graphe. [Towards Graph Modeling of Japanese dictionaries] Master thesis, Paris Diderot. Accessible at: https://louis.lecailliez.net/dl/memoire_m2_jap_Lecailliez.pdf.

Lecailliez, L. (2017a). Preliminary Thoughts on Issues of Modeling Japanese Dictionaries Using the OntoLex Model. In Horák, A.; Rychlý, P.; and Rambousek, A., editor(s), *Proceedings of the Eleventh Workshop on Recent Advances in*

Slavonic Natural Languages Processing, RASLAN 2017, pages 11-19, 2017. Tribun EU. Accessible at: https://nlp.fi.muni.cz/raslan/raslan17.pdf.

Lecailliez, L. (2017b). Ordering of nodes in a graph-modeled dictionary. Research Note. Accessible at: https://louis.lecailliez.net/dl/Ordering_of_nodes_in_a_graph.pdf.

Mair, V. H. (1996). Modern Chinese Writing. In Daniels, P., Bright, W. (eds.) *The World's Writing Systems*. Oxford University Press, Oxford.

Mangeot, M. (2016) Collaborative Construction of a Good Quality, Broad Coverage and Copyright Free Japanese-French Dictionary. *International Journal of Lexicography 2016*; doi: 10.1093/ijl/ecw035; 35 p.

Mangeot, M., Chalvin, A. (2006). Dictionary Building with the Jibiki Platform: the GDEF case. In *Proc. of LREC 2006*, Genoa, Italy, 23-25 May 2006, pp 1666-1669.

Mangeot, M., Sérasset, G. (2001). Papillon Lexical Database Project: Monolingual Dictionaries and Interlingual Links. In *Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium*. November 27-30, 2001, pages 119–125, Tokyo, France.

McCrae, J., Aguado-de-Cea, G., Buitelaar, P., Cimiano, P., Declerck, T., Pérez, A. G., Gracia, J., Hollink, L., Montiel-Ponsoda, E., Spohr, D., Wunner, T. (2010). The lemon cookbook. Technical report, Monnet Project (June 2012), www.lemon-model.net.

McCrae J., Spohr D., Cimiano P. (2011) Linking Lexical Resources and Ontologies on the Semantic Web with Lemon. In: Antoniou G. et al. (eds) *The Semantic Web: Research and Applications*. ESWC 2011. Lecture Notes in Computer Science, vol 6643. Springer, Berlin, Heidelberg.

Polguère, A. (2014). From Writing Dictionaries to Weaving Lexical Networks. *International Journal of Lexicography*, 27(4):396–418.

Powers, S. (2003). *Practical RDF*. Sebastopol: O'Reilly & Associates.

The GraphML File Format. (2001). Accessed at: http://graphml.graphdrawing.org/. (20 May 2017)

Robinson, I., Webber, J., Eifrem, E. (2013). *Graph databases*. O'Reilly Media, Inc.

W3C. (2006a) W3C Ontology Lexicon Community Group. (2016). Final Model Specification. Accessed at: https://www.w3.org/community/ontolex/wiki/Final_Model_Specification. (20 January 2018).

W3C. (2006b) Accessed at: https://www.w3.org/TR/swbp-n-aryRelations/. (20 May 2017)

Williams, S. (2000). *The associative model of data*. Second Edition. Lazy Software.

Williams, S. (2001). The associative model of data. *Journal of Database Marketing & Customer Strategy Management*, 8(4), 336-359.