Modeling Tools for the Integration of Structured Data Sources

by

Jyotsna Venkataramanan

B.S. Massachusetts Institute of Technology (2009)

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

~~December 2010~~ [February 2011]

Author [illegible]
Department of Electrical Engineering and Computer Science
December 14, 2010

Certified by [illegible]
Dr. David Brock
Principal Research Scientist
Thesis Supervisor

Accepted by [illegible]
Dr. Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

Modeling Tools for the Integration of Structured Data Sources
by
Jyotsna Venkataramanan

Submitted to the
Department of Electrical Engineering and Computer Science

December 14, 2010

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Disparity in representations within structured documents such as XML or SQL makes interoperability challenging, error-prone and expensive. A model is developed to process disparate representations to an encompassing generic knowledge representation.

Data sources were characterized according to a number of smaller models: their case; the underlying data storage structures; a content model based on the ontological structure defined by the documents schema; and the data model or physical structure of the schema. In order to harmonize different representations and give them semantic meaning, from the above categories the representation is mapped to a common dictionary. The models were implemented as a structured data analysis tool and a basis was built to compare across schema and documents.

Data exchange within modeling and simulation environments are increasingly in the form of XML using a variety of schema. Therefore, we demonstrate the use of this modeling tool to automatically harmonized multiple disparate XML data sources in a prototype simulated environment.

Thesis Supervisor: Dr. David Brock
Title: Principal Research Scientist, MIT Data Center

Acknowledgements

Foremost I would like to express my sincere gratitude to Dr. David Brock for being an incredible mentor. Even through project deviations and grammatical disasters he has been very patient and helpful and I am extremely fortunate to have him as my thesis supervisor.

I am especially thankful to Anne Hunter, MIT Course 6 administrator, for her incredible support, always lendigng an open-ear and providing guidance through my years at MIT.

I would also like to thank Dr. Duane Boning for his guidance through my career at MIT.

I am indebted to the amazing professors at MIT who have given me an appreciation for my subject and for engineering in general. For this I would especially like to thank Dr. George Verghese.

I thank my fellow labmates at the MIT Data Center especially Chris Spassimirov for their ideas and hard work. Their comments helped me improve my research by leaps and bounds.

I would like to express my immense gratitude to my family for providing such a supportive environment. Especially to my grandparents V.G. and Rukmini Rajagopalan for their support, even from half-way around the world.

Most importantly I would like to thank my brother Dhananjai for all the emotional support and for never doubting me. And to my parents P.S. and Bhavani Venkataramanan, who raised me, supported me, encouraged me and loved me. I dedicate this thesis to them.

Table of Contents

Table of Figures

1. Introduction

Structured documents are prevalent in industry especially as a means for exchanging information. To manage the different types of data, hundreds of structured languages have been created. Although these languages facilitate communication it makes interoperability and integration very challenging. Any approach to interoperability and integration is ultimately dependent on standards, ontologies, common vocabularies and higher level declarations or some abstraction that relates vocabularies to each other [1].

This paper presents a new approach to interoperability by using a new conceptual model. This model resolves inconsistencies in languages at multiple levels.

1.1. Structured documents

Very little research or implemented systems can be universally applied to unstructured data (e.g., text documents), semi-structured data (e.g., XML documents), structured data (e.g., relational databases). This model leverages the information provided by the nested hierarchy and regularity of structured documents.

Structured documents are defined by a hierarchy of tags and a self-describing schema that are associated with some data. It is used in the majority of the business world in information whether it is the tagging, transport or storage. Structured documents are defined by their document and schema. The document contains the data that the information system was created to manage. The schema describes the generic document by providing constraints for the semantics, structure and data objects and types.

1.2. Goal and project structure

This research examines a conceptual model for structured documents to facilitate interoperability. What we want to do is take structured documents and their schema and break them down into layers of abstraction according to the conceptual document model. Then facilitate interoperability by examining the differences in different documents across these layers.

The main goal of this project is to design a conceptual model for structured documents for the purpose of interoperability across this model. We implement this model and provide a modeling tool to utilize certain aspects of the model on schema.

We discuss the challenges and the scope of this task in section 2. Section 3 continues by describing the research carried out by others in this field. In section 4 we describe the model. This conceptual model is separated into the following layers of abstraction: case, schema model, content model and data storage structures described in section 4.2, 4.3, 4.4 and 4.5 respectively. The semantic basis for this model, the m language dictionary, is described section 5. The implementation of the modeling tool is discussed with screenshots in section 6 and the last section discusses the direction of this work in the future.

2. Defining the Problem

Critical to industry is communication. And important to communication is the ability to understand, translate and integrate different types of data.

An automated integration system would be ideal. However, automated systems have been studied for quite some time with minimal progress. There are not even any standard measures on how to structured documents compare to each other. While some research has focused on statistical and querying techniques to compare documents, very little effort has focused on the model *behind* the data. This dearth of progress illustrates the difficulties and challenges in the area of data integration.

In this section, we discuss motivation and challenges in the field of structured data integration.

2.1. Motivating Example

Government schemas are some of the largest and most complex in existence. They are also the most frequently used, as their application is mandated for many industries. Consider, for example, what happens when two schemas must be integrated. Generally, a committee is formed to map terms from one schema to another. This process can take many months to even years. At other times, multiple schemas are combined to form a larger 'umbrella' schema. This is exactly what led to the creation of the National Information Exchange Model (NIEM). NIEM contains many sub-schema that describe aspects such as emergency management, immigration, intelligence, international trade, criminal justice and so on.

To create this model, the designers first analyzed all the separate domains and decided on which element to include. After determining the common elements, a working data model was created. From the data model, a 'base' model was formed along with a number working groups to manage the various subdomains. An oversight committee was also formed to oversee any changes in the base model. With every new version of the schema an updated base model was released.

While this approach was very comprehensive and exhaustive, significant time and resources were necessary to create and maintain it. An automated system would, of course, be a much more efficient and cost effective solution, but the challenges facing its creation are equally significant.

2.2. Challenges

The main challenge is the complexity of integration. Even the *concept* of integration can be ambiguous [2]. Does integration mean simply combining data? What does data from one context mean in another? Which elements should be selected among conflicting sources? How should data be fused to yield optimum results? Every variable we consider increases the complexity of the integration problem. Thus data interoperability and/or integration have been referred to as 'hard problems.'

Many structured documents are written using the Extensible Markup Language (XML). However, XML must be consider a 'semi-structured' language in that the hierarchical format provides some structure, but the tag names and their contents are wholly derived from natural language. Parsing XML documents thus requires techniques similar to natural language processing (NLP), which is largely an unsolved problem in computer science.

The semi-structured nature of XML allows flexibility in communication, but increases the difficulty in parsing and integration. Furthermore, the flexibility allows multiple disparate documents to describe exactly the same data. Given the number of dimensions by which two schemas may differ presents a challenge for a unified conceptual model and integration algorithm. However the limited regular can provide some advantage in building a document model [4].

The degree of depth, or hierarchical structure, varies among schema – even those describing the same data. Information can be represented a different hierarchical positions, which make comparisons among large schema difficult and time consuming.

Tag name describing the same data can vary between schemas. The names and meanings of XML tags are arbitrary. Even the method by which tags names are constructed can vary, as shown in the following examples

```
AFCT_EQPT_CODE_TYPE
deliveryDate
SSN
HasDestinationOf
TPRcategory
BID_Collection_List
IsEmployedBy
```

The tag names and their design patterns are at the sole discretion of the designer. As a result, every schema generally has a completely different vocabulary. To map between tags, we require a common vocabulary. Some dictionaries, such as WordNet™ have been used for this purpose, though this approach has some difficulties, as will be described in Section 5.

Level of detail can also be inconsistent across schemas. The same information can be modeled in at varying levels of granularity. For example 'name' may tag a simple character string or encompass multiple sub-tags providing additional levels of detail, as illustrate as follows.

```
<name>Jyotsna Venkataramanan</name>

<name>
   <fname>Jyotsna</fname>
   <lname>Venkataramanan</lname>
</name>
```

As a result of these semantic dependencies, automated systems that parse and integrate semi-structure data must process natural language.

Context is key to solving semantic ambiguity. While some tags make sense in one context they may represent wholly different concepts in another [4]. Some context is provided by the XML hierarchy, but in general is not well represented. Understanding context, however, is necessary to derive full meaning from the document [3]. We discuss the challenges of context and propose a solution in section 5.

There can also be challenges matching *data objects* across structures documents. There are three main categories of this problem

- Parsing and identification of data tokens - Different tokens might be used for the same object in different sources. Additionally data values may be truncated, abbreviated, incorrect or missing. And large amounts of human effort may be required to develop a database which records determinations for equivalent data objects

- Translation - Several fields to describe what one source used a single field to describe. Keys, attributes, structures and encoding conventions may differ across applications.

- Representation - Key relationships might be hidden because a key structure that linked the documents wasn't provided. One example is if multiple account numbers might block the fact that all are from the subsidiaries of a parent account.

Another interesting problem with data object inconsistencies is in units of measure and their representation. Some of the ways to describe the units are as follows

- Incorporate into the data

```
<transfer>$750</transfer>
```

- Use a separate tag

```
<transfer_amount>750</transfer_amount>
<transfer_currency>$</transfer_currency>
```

- Use an attribute

```
<transfer unit = "$">750</transfer>
```

- Place together in data element

```
<transfer>
   <amount>750</amount>
   <currency>$</currency>
</transfer>
```

The sheer variety of ways in which the units can be represented provides a challenge to parsing and interoperability.

In general there are structural, semantic and data object challenges in parsing and integrating structured documents. In this thesis, we address all these challenges using a multi-layered abstraction described in section 4.

3. Background

Interoperability of structured data has been researched academically and developed commercially. Most of the work in interoperability of data sources and schema has been carried out in the following areas [5]:

(i) database schemas – e.g. SQL

(ii) XML and XML-based schemas

(iii) ontologies in knowledge representation

Similarities in the structure of these representations mean that similar models and algorithms based on the structure will apply to all these types of representations. In this section we consider interoperability in the case of the semi-structured forms mentioned above and provide specific examples from each of their domains. In Section 4.1 we go into further details about the specific scope of our method and where it lies along the spectrum of non-structured text to structured data documents.

The adoption of more structured languages such as XML, SQL and other ontologies as standards was initially a herald of more interoperable information exchanges. The flexibility of the structure was initially designed to help in integrating structured, semistructured and unstructured data [6]. However the variations in versions have made interoperability very difficult. Since these structured languages are just textual languages we need to be able to create a mapping or ontology between two different representations to connect them. This is the basis of most integration approaches.

3.1. Global Schema

Previous research into data integration and schema matching has led to a number of different approaches. Most of these solutions involve a global schema that integrates different data sources to one universal model. A global schema differs from the solution proposed in this thesis in that it is not an abstraction but merely combines many different data sources. The disadvantage, however, is that without a governing abstraction, global schemas are manually tedious, unwieldy, difficult to manage and hard to expand. The model proposed in this document hopes to provide a better solution by addressing these challenges.

There are two main methods to create such a global schema; a global-as-view approach where the global schema is expressed in terms of the data sources and a local-as-view approach where the global schema is represented independently of the local schemas and serves as a view over the local schema [7]. The two approaches can be implemented through two main concepts that constitute the architecture of a data integration system: mediators and wrappers [8].

The global-as-view approach is examined in Section 3.1.1 in the context of the creation of the government project NIEM (National Information Exchange Model) [9]. The patterns common across global-centric schemas are discussed and the advantages and disadvantages of them examined. In section 3.1.2 we discuss the local-as-view in the context of two systems, the Dublin Core [10] as well as the government standard UCore (Universal Core) [11]. We also discuss methods that have been developed to automate this process.

3.1.1. Global-as-View Approach

In the global-as-view approach the global schema is expressed in terms of the integrated data sources. Every concept in the global view has a view to a concept in a local schema [12]. A global-centric schema integrates local schemas to a global schema. To create such a global schema a new schema core needs to be derived and then consistent semantics be created to manage the local schemas. A mediator maintains the core global schema and also the mappings between the source and global schema [8]. A mediator can be a group of knowledgeable users who maintain the schema.

An example of the global-as-view approach in practice is NIEM, an XML based framework developed by the United States government that integrates schema from a vast variety of sources (NIEM, 2010). NIEM is an example of a global schema that combines the frameworks from 10 different sources such as CBRN (Chemical, Biological, Radiological, and Nuclear), Immigration databases into an integrated model and converted to an XML based language. The creation of this standard arose from the critical need to integrate the 10 different domains to promote the efficient dissemination of information between agencies within the government.

The core components within NIEM are as follows [9]:

- Data Components – are the "fundamental building blocks" of the NIEM framework. These represent a concept such as a person or an event. These are the basic components that are collected across the domains which contain the relevant data.
- Core – A collection of the data components that are consistent across the domains. For a global schema this is the information that remains consistent, that we want to share across its integrated schemas.

- Domains – a narrow system organized to facilitate governance. Each has a group of subject-matter experts who harmonize the discrepancies that arise with the domain

The NIEM model is very representative of general global schemas; it is based on the core and corona [13] design pattern which is common to many unifying schema frameworks. With respect to the example given above, the core is similar to the NIEM Core and the corona is an extension of the core and contains domain specific extensions of the schema. Some smaller scale examples of the core and corona pattern in a global schema are the XML wildcard *xs:any* that allows the user to utilize a type of their creation or a editable database that allows the addition of a column [13].

The common core and corona pattern means that the disadvantages of the NIEM model are also very representative of those of global-centric integrated schema. NIEM is able to provide data exchanges that are consistent and interoperable but does provide a number of challenges. The elements in the NIEM core schemas have been designed to be consistent across all domains and therefore tend to be over inclusive in the number of data components. The data component "person" for example contains about 1.5Mbytes of attributes [14] which makes mapping from the domain to the corresponding attribute of a core component time-consuming. Here mapping techniques, discussed in Section 3.2, could automate some of this process. The sheer volume of the core and domains and relationships provides a challenge for users who require extensive training to gain the knowledge to effectively utilize the system. To compensate for the lack of such a knowledgeable user we would need to produce a more automated system that could extract semantic knowledge, similar to the tools presented in this document.

The disadvantages of creating a global-centric global schema for interoperability are that it is a time consuming manual process, mostly it results in a tag-heavy data model and usually the data structure is not fluid enough to expand easily at a future date. Another challenge of NIEM is the addition of new information we wish to exchange across domains. A change to a single domain will need to be matched to the core and across domains both semantically and structurally to maintain a consistency in the model. It also requires the dissemination of the new inclusion to all users.

Another one the problems of a global schema like NIEM are the management of the extensible, evolving domains. While there are many commercial tools that are built to support other forms of integration there are very few tools aimed at the core and corona pattern specifically. Utilizing the core and corona pattern there are systems that have been built to increase the likelihood that data across the corona is interpreted in compatible ways. Galaxy [13] is a schema manager that utilizes the consistency across the core and maintains a derivation graph to exploit for data exchange, discovery and query. First, developers use Galaxy to search a metadata repository for existing data models that partially meet their information needs. This is the base of the creation of the derivation graph. Galaxy is then used to customize the model. Galaxy keeps expanding the graph based on evolving corona. The graph automatically provides interoperability over the shared portions of the model [15]. Galaxy also provides tools that encourage schema reuse and automate the generation of inter-schema relationships. The generation of these inter-schema relationships is similar to the schema mappings, discussed in Section 3.2, so we see how systems like Galaxy are moving to combine models harnessing the advantages of a cohesive integration provided by a global-centric model and the automation provided by schema mapping algorithms.

3.1.2. Local-as-View Approach

In the local-as-view approach the global schema is specified independently of local schema [13]. Every source is defined as a view over the global schema. A wrapper allows the addition of components of the source schema directly to the global schema through a layer of code. It wraps information from the source and models the source using the source schema [8]. This is another method to integrate sources, create a base core and incorporate parts of the local schema by the way of wrappers.

One example of a system created using the local-as-view approach using wrappers is Dublin Core implemented in XML which provides a small set of semantic elements to describe and catalog information resources [10]. The metadata records based on the Dublin Core are used to facilitate cross-domain information resource description and have become standard in the fields of library science and computer science. Because of the elements that have been added to the Dublin Core it has evolved past being just an interoperability standard to being a model. It is now defined on a "ladder of interoperability" [10]. The levels of the ladder are, top down:

- Description Set Profile Interoperability – Shared formal vocabularies and constraints in records
- Description Set Syntactic Interoperability – Shared formal vocabularies in exchangeable records
- Formal Semantic Interoperability – Shred vocabularies based on formal semantics
- Shared term definitions – Shared vocabularies defined in natural language

This ladder of interoperability approaches a base model for interoperability. It proves to us that it divides and integrates information syntactically and semantically. The aspects lacking in this model are the ability to extract meaning from the information and tools to automate extracting all these layers of information and data. The model proposed in this thesis hopes to create an all-encompassing system that tackles these issues.

Another unique implementation context of a local-as-view model is Electronic Business XML (ebXML). EbXML is a set of independent specifications and these specifications construct the implementation framework of e-business [16]. Currently there are 7 modules specifications in ebXML: requirements, technical architecture, registry information, registry services, message services, business process schema and collaboration-protocol profile and agreement. These all interact together and a layer on top of these modules forms a united core.

Another United States government standard, the Universal Core (UCore), describes a small set of essential data along with the provision for domain specific enhancements [11]. Similar to NIEM, UCore is a government project designed to promote information exchange across different domains. While NIEM took a global-as-view approach, creating a new standard and extensible domains, UCore takes a local-as-view approach. It focuses on providing a lightweight exchange schema used across agencies. It wraps messages across agencies in a taxonomical structure under the headings *who, what, when* and *where.* As expressed in [17]*:*

> *if I have a UCore-wrapped National Information Exchange Model [NIEM] message from Immigration and Customs Enforcement about illegal immigrants wounded during criminal activity and I have a UCore-wrapped Health and Human Service Department message on visitors to emergency rooms, I have enabled immediate cross-domain search. ... UCore is a process of extracting cross-domain commonality from your message flows, thereby massively broadening the possible adoption and use of your shared information. In information sharing, adoption by consumers is the key value metric.*

Therefore UCore allows you to create simple messages between domains; it acts like a wrapper around messages from different schemas and provides a consistent and lightweight way to share the messages.

There are a few drawbacks for such a wrapper based system, there is a lot of effort required to write a wrapper for each component we wish to integrate. This is not quite the case for UCore where a very simple wrapper is used specifically to keep the model lightweight. True integration never fully takes place because you are just combining parts of other schema. While UCore provides a base to exchange messages, you are only exchanging messages in different formats, there is no extraction of information and we cannot combine different message representations into one coherent one. Since we cannot efficiently integrate within a single message different messages, there is often redundancy in the information the wrappers contain that isn't removed. The data isn't even considered or parsed or integrated in any way. The difficulty with this approach is the compatibility between multiple disparate extensions. These integration problems, in fact, mirror the difficulty of structured data interoperability in general.

There have been many initiatives developed to overcome some of these problems by automating the creation of the wrappers. Article [18] provides an approach for semi-automatically generating wrappers for semistructured web documents by identifying the underlying structure of a page from its formatting. With such a utilized and analyzed source like HTML there are a plethora of developed automatic wrapper approaches ([19], [20] and [21]). However they do require a modification to fit the source schemas. A mediator is required who understands the source schemas well enough to transform the algorithm to fit the particular circumstance. Also, generated wrappers have their disadvantages as they do not extract data structured by other sources like tables or textual formatting.

3.2. Schema Mappings

Schema integration can also take place by mapping between individual schema. In previous implementations, schema matching is typically performed manually, which has significant limitations of time and correctness. However, more research papers have proposed many techniques to achieve a partial automation of the match operation for specific application domains.

There is a significant overhead in designing global schemas and some systems have overcome this by relying on source-to-source mappings [22]. We have shown in section 3.1.1 how schema mapping can be used in conjunction with global schema to produce a more integrated system. While schema mappings cannot be by itself as an interoperability standard it is an integral step in any integration system. It is useful tool that can be combined with other approaches to promote interoperability.

There are several surveys of schema mapping solutions ([5] and [23]) that have mapped out the approaches that been tackled and the challenges left in the field. Schema matching algorithms can be classified in the following ways [23]:

- Instance vs schema – matches involving instance data or schema level information
- Element vs structure matching – matches for elements or more complicated structures
- Language vs constraint – matches based on names and descriptions or on keys and relationships
- Matching cardinality – matching cardinalities at each level

- Auxiliary information – matches using dictionaries, global schemas, user input

There are commercial solutions available that facilitate the integration of XML sources. The XML Schema Mapper, for example, from Stylus Studio provides a visual development environment to quickly generate element-to-element schema mappings [24]. This method therefore functions on the element vs structure matching level.

The IBM schema mapping management system, Clio, derives likely mappings by analyzing the schemas and the underlying data using a Naïve-Bayes-based matching algorithm [25]. It consists of a set of readers which translates a representation of the target schema to a Correspondence Engine (CE). A mapping generator generates mappings between the data in the source and the data in the target schema. Then mappings between the target and source schema are derived given the auxiliary information from the correspondence engine.

While schema mappings provide us with the most lightweight and automated approaches we have to data these solutions lack an encompassing abstraction and sometimes do not combine semantic understanding with their matches. This is a disadvantage that requires manual input from a knowledgeable user to correct.

3.3. Models for Interoperability

Recently research is turning more towards building a model or representation that can fully capture the problem of interoperability between disparate representations of structured documents. A model is intended to bridge the gap between the conceptual design that was intended by the designer of the schema and the actual implementation [26]. While some of the methods described in the previous sections utilize a model in some small part there hasn't been a large-scale standard based wholly on a model. The Dublin Core has expanded its core and has evolved into a semi-conceptual model. The "interoperability ladder" has many elements of a conceptual model; it is able to divide a document up into smaller categories for which we can generate smaller models.

Proposed models, from [26] and [27] define ways to characterize a model for schema. The model put forth by [26] intends to define interoperability in separate layers and use them to characterize a system. The goal would be to be able to use such a conceptual model to define in the early stages on integration the amount of interoperability the model would achieve a priori. On the other hand [27] defines a very specific model to represent the semantic information in XML models. The model consists of a component called the semantic mapping agent (SMA) that allows it to store a semantic "definition". The component then maps values from one semantic definition to the other. While both these models are at different levels of granularity, [26] looks at the system interoperability as a whole and [27] tackles the problem of modeling the semantic relationships contained in an XML document, they both show that it is possible to create a model that will improve interoperability. The two models also define the gamut of modeling problems within the field of interoperability from the system as a whole to an abstraction layer within a schema.

3.4. Conclusion

All the approaches described in this section aim to unite separate local schema either by creating a coherent a global schema or matching schema or data elements. They all differ in the level of automation or human/user involvement in their creation. The first approach, described in section 3.1, of creating a new global schema is very dependent on users for creation and management of the system. This solution is mainly manual and while it is tedious to manually compare and match across disparate schemas, automated solutions has proven difficult to develop [22]. On the other hand algorithms for schema mappings are highly automated and lightweight. We also described some systems that attempted to combine these approaches to automate the process of data integration.

If we could automate this process of mapping the schema to semantically sound concept representations, the resulting integration would be easier, faster and more correct. This is our main aim in creating a generic encompassing model, to make a highly automated, computer readable system.

4. Document and Schema Model

The following section describes our model for interoperability. We refer to this model as the 'document model'. It is a concept of structured documents designed to promote interoperability.

4.1. Document Model

A model is an abstraction of reality according to a certain conceptualization [29]. Once represented as a concrete artifact, a model can (1) support better understanding, (2) provide a basis for analysis and detail, and (3) allows the interoperability of different semantic representations. Therefore by derivinga conceptual model of a structured document we can extract meaning from the document and use the model to aid interoperability between disparate versions.

Our approach to an abstraction is to transform disparate representations into a neutral format using a common conceptual model and a shared vocabulary. This section defines the basis of a conceptual model for structured documents. This conceptual model when combined with a common vocabulary described in Section 5 should provide the basis to discover the intended meaning of a schema and document. By representing our document as a generic model, disparate documents could be understood and integrated.

4.1.1. Conceptual Modeling of Meaning

The relationship between conceptualization and reality is depicted in Ullman's triangle [30]. This depiction derives from the initial work of scientists as Ogden and Richards who built the foundation for the science of language. The figure below shows this semiotic triangle. The figure describes the three elements of meaning. The 'thing' refers to the actual physical manifestation that we are trying to retrieve meaning from. The 'concept' is an abstraction or conceptualization of the manifestation; it is some sort of abstract model that defines the 'thing'. The final vertex of the triangle is the 'symbol' which is the representation of the 'concept'. This means it is the language that defines the conceptualization of the real life manifestation from which we wish to derive meaning.

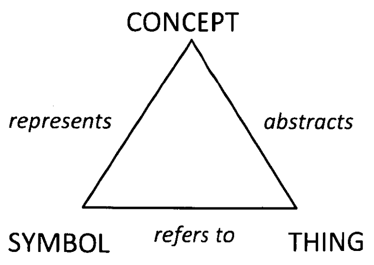


Figure 1. The semiotic triangle defined by Ullman defining the three elements of meaning.

This network also applies directly to our thesis. We wish to derive the meaning behind a structured document and make it computer understandable. In this context the 'thing' refers to the document and/or schema that we are trying to analyze. The 'concept' will refer to the abstraction that we wish to derive in this thesis and the 'symbol' refers to semantics and/or a computer understandable language. The flow of this meaning is shows in the figure below.

The main objective of this work is, thus, to contribute to the theory of conceptual modeling by proposing a basis for such a model of structured documents. We focus on the theory, design and analysis for the 'concept' vertex of the semiotic triangle. This thesis also proposes some of the semantics required to represent the 'symbol' as well as an implementation of the 'concept'.

4.1.2. Previous Models of Document Meaning

In Section 3 of this thesis, previous versions of theoretical models for structured documents have been discussed. Many approaches to modeling a structured document involve creating a domain specific language of the structured documents. Ullman's triangle shows why such an approach is flawed. In the context of the semiotic triangle, while a domain specific language is the 'symbol' that refers to the structured document or 'thing', without a 'concept' we are lacking a critical vertex. And without all three vertices there cannot be a full understanding of the meaning of the document. This means that without a 'concept' vertex fully defined we are leaving out a key part of the equation that leads to extracting the meaning of the document. This theory suggests that without having a conceptual model of a document as well as the semantic language to define it we do not have all the tools to retrieve meaning from the document.

Many other previous approaches have modeled a document simply in terms of the syntactic model and the semantic model [28]. The basic model for analyzing a document has always been considering two models: the syntactic and the semantic. It was considered that all information could be analyzed and defined by one of these two models. The syntactic model defines the structure which in structured documents is already fully modeled by xml or whatever structured language is used. The semantic model is the element that defines the meaning of the information. It is usually represented by an ontology. This is a very simplistic model that doesn't take into account any of the details of the structure, data or the relationships between them. Each sub-model covers such a large scope that it can be interpreted and defined in any number of ways. The semantic model can consider the semantic structure of the encasing language or it could consider the meaning of the data defined or the relationship between the two. It has to fully describe the choice of semantics in describing the tags, the semantics of the actual data or the relationship between the two. The disadvantages of such a two layer model is that it is a very simplistic model but yet very hard to actually define. We hope to give more structure to the semantic model by capturing it as a system of more concise and definite sub-models.

4.1.3. Thought Experiment

To extract the true meaning of a structured document or the schema we have to consider how the author created them. One approach is to guess the thought process of the author. How did he/she model the data and create the schema? In this thought experiment we assume a knowledgeable author that creates both the schema and an instance of a schema or document. We consider an a-priori model wherein the author first designs a schema and then builds a document around it. The steps below describe this thought experiment:

(i) If the data isn't already stored in a structure, the author considers which of the data can be grouped together or how it can be divided up to relate to each other. With this in mind the author builds a storage structure to hold the data - table, database, hierarchy etc. depending on the aforementioned relationships between the data. If the data corresponds to each other in a very regular way, the author might consider a database. If, however, many data elements relate to just one other in a recursive fashion the author might consider a hierarchy.

(ii) After building a structure for the data, the author now considers ways to represent the relationship between the groupings of data. For example, if the author knows that he/she is using a table, he/she would define a title and column headings.

(iii) The next step is to combine the syntactic and semantic elements. The author must decide which elements in the schema are derived from each other and which are derived from external schema. For example if the author is considering a hierarchical structure of nodes, he/she would decide which nodes can be grouped together to create an object and which objects can derive from one another. At this stage the author must decide the physical structure of the schema based on the semantic relationships.

(iv) The final step is to choose a language that can represent the derivations described above. For example, XML, SQL or the Resource Description Framework (RDF), may be used to implement the author's data model. The language chosen gives us constraints of the amount of information a document can retain. The case model characterizes this.

One approach to building the basis of a conceptual model is to utilize this experiment to engineer a model.

4.1.4. *Layers of the Document Model*

Creating a model means extracting the implicit structure of a document. Schemas outline the semantic structure of the document and are made to efficiently display the data. Therefore a complete model must consider both the data and the schema that it was compiled from. Interestingly, schemas are structured documents themselves which emphasizes the recursive nature of data representation. Therefore defining a model could then lead to a system that would generate schemas. This is another potential application for a model defining structured documents.

To create a conceptual model we can use the thought experiment described in Section 4.1.2. The four stages of the experiment define one path to the complete transformation from data to schema to document. Therefore we can identify this process backwards and create a model that considers a schema and document and identifies the core data and the meaning that the human author meant to convey through this data.

Based on the assumptions and a priori schema described above, we create a base document model that is partitioned in four main layers of abstractions that fully describe the meaning and both the data and the schema:

1. Case
2. Schema Model
3. Content Model
4. Data Storage Model

The figure below depicts this model as per its layers of abstraction.

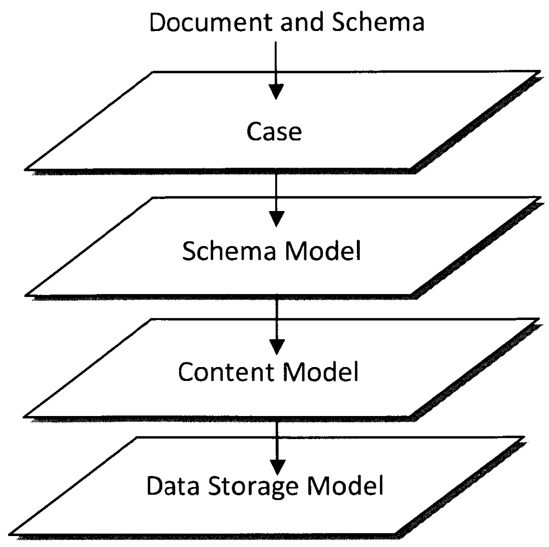


Figure 2 Diagram describing the layers of abstraction in the structured document model.

The *case* defines the language in which the ideas from schema and the document are conveyed. Each language has a different structure and this structure constrains the amount of information that can be relayed. The case layer describes the constraints on the data defined by the structured language. This is the fourth and last step in the though experiment. In Section 4.2 we explore the different cases of structured data we hope to include in this model and ways to expand case to less structured documents.

The schema model differs from the content model because most the physical structure is omitted from the previously defined ontology. It is the second to last step, step (iii) in the thought experiment. Having looked at the data structures of the document and the ontological structure of the schema, the other logical step is to look at the relationship between the data and the schema and this is described in Section 4.3 under the schema model.

The content model describes the semantics and structure used to represent the data; it defines the language of the structured document. It is the second step described in the thought experiment. It is one of the critical elements and must also be analyzed to draw a complete model for the document. The content of the document, in this case, refers to the schema tags and relationships between the tags. The model depends on creating an ontology of the information that the schema can represent and is discussed in detail in Section 4.4 of this chapter.

The last layer of abstraction of the model maps to the first step, step (i), of the thought experiment. After tearing through the other layers of the model we are now left with the data. The data storage model identifies the underlying data structures on which the document is defined. Documents are designed with a specific underlying data structure and classifying the document is a main step to revealing the meaning behind the data.

The layers described above are important characteristics that must be considered in extracting meaning from a document. They include the case or the language in which the document is written, the physical schema structure, the content or the ontology of the schema and the underlying data structures depicted by the document. After analyzing across these layers and applying across a shared vocabulary we aim to provide a generic model for interoperability.

4.2. Case

The first layer of the model defined, is the case of the document and schema. This refers to the language or ontology used in the document and defined by the schema whether it be HTML, XML, RDF, SQL or any even natural text. In this chapter we discuss case as the spectrum of documents and where on this spectrum our model applies.

An interesting thing to note is that the particular language that is chosen provides constraints for the information it can represent. A document that is natural text is unstructured and can represent more information than an xml structured document. We are trying to characterize this metric by providing a numerical equation for the structure of a document. This value of structure is equivalent to the amount of information it is constrained to hold.

We can categorize documents into three main groups depending on the level of structure in the document; structured semi-structured and unstructured documents. Along this spectrum of document categories, there are different cases. Languages like RDF and SQL are more structured, XML and HTML fall in the semi-structured category and any example of natural text is an example of an unstructured document.

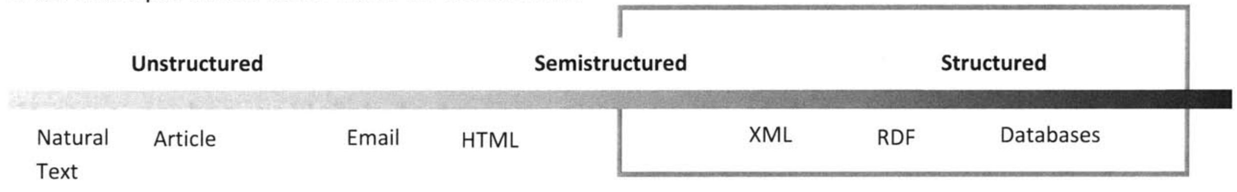


Figure 3: The spectrum of structured data with examples and the red box represents the part of the spectrum where the model defined in this thesis can be applied.

Modeling a document structure is a relatively new research topic and there are very few systems that can work for all cases of data documents. They are all too varied in structure and format. Modeling a highly structured document like a relational database is redundant since its schema already defines its data model. Our goal was to begin with a document type that is relevant, prevalent and has a base structure but captures a lot of unstructured information. This is how we chose the semi-structured to structured end of the spectrum. Therefore this paper focuses on parsing and interoperability of XML schema and some other semi-structured cases. However a comprehensive model should differentiate between but involve different cases. While in this paper most of the examples refer to XML, due to the prevalence of XML-based languages, the ideal model should apply to any and all these different languages or sources. Parallel research is taking place within the Data Center to extend this model to apply to the other cases. We hope to create at the end a unifying model that will parse documents across all categories and cases.

The outline of this chapter is as follows, section 4.2.1 discusses the spectrum of documents as categorized by structure and examples of cases that apply to each. In Section 4.2.2 we define a metric for structure and analyze how it follows our findings in the previous section. Section 4.2.3 follows by discussing case in the context of this project.

4.2.1. Spectrum of Documents

All written or textual documents (i.e. not graphics) can be placed on a spectrum from unstructured, free form text to semi-structured documents to structured relational tables.

Structured documents have a very strict structure meaning, a consistent appearance. They also have a framework or data model attached to it. Examples include relational databases or SQL. This is the easiest category to parse and extract information from since it is in the most computer readable form and already comes with a data model. The base model therefore covers this category, structured documents.

Unstructured data consists of any data stored in an unstructured format at an atomic level. That is, in the unstructured content, there is no conceptual definition and no syntactic definition [31]. Examples of cases which are used in unstructured documents include any sources solely with textual information such as emails, reports, articles, books and any other free form text. Unstructured data does not have a data model and it cannot be easily parsed by a computer program. Unstructured documents provide the freedom to incorporate different data and elements without having to change the schema or data model. Of the three categories this has the most free form and least structure but often also contains the most information.

Most data is unstructured data. There are actually estimates that say that 80% of all business relevant information is unstructured [32]. At this point there is no way to automate the processing of unstructured data. So human involvement is necessary to make it readable by adding tags to the information or providing an ontology or some structure. The Fields of data mining and text analytics are dedicated to parsing unstructured data information and extracting meaning from it, we have mentioned many examples in Chapter 2. However very few of these methods use the approach of building a conceptual model of the document. We've taken the view that this unstructured data is merely data that hasn't been given a model. Research is being conducted in parallel to expand the model to also parse unstructured data.

Semi-structured documents contain elements of both structured documents and unstructured documents. They have some defined structure to them like tags and also have some unstructured text attached. Examples of semi-structured documents are XML-based languages, HTML and therefore a large part of the World Wide Web. Ironically semi-structured data became popular when it was seen as a way to integrate unstructured and structured data [3]. Now there is a world of semi-structured data sources, and the sheer number and different formats makes interoperability difficult. Because of its base structure and inherited elements from both structured and unstructured data the initial version of our model presented here applies to semistructured data.

4.2.2. Quantitative measure of structure

A measure of structure can be useful in characterizing a document. For this thesis, it provides a quantitative measure for the first layer of the model. A measure of structure helps classify documents and aid in their automatic analysis.

The amount of document structure is proportional to the amount of semantic information embedded in the meta-data. From the spectrum discussed in the previous section, we know that databases provide the most structure. A database document contains labeled tags that identify each piece of information. From such a structured document, significant semantic information about the data and relationships between data may be extracted.

On the contrary, an unstructured document has little regular structure and not explicit meta-data. Without parsing the document using a well-defined dictionary, we would not have any semantic information at all. Furthermore, we would not know how any piece of information would relate to any other.

We observe that the ratio of the amount of data to the number of meta-data elements is a factor in quantifying the degree of structure in a document. The deeper the hierarchy of tags the more structured the document contains. We define a hierarchy factor in the equation below. The limit of this equation goes to infinity for unstructured documents when the number of tags approaches zero, and to unity for an equal number of tags and data elements.

$$hiearachy\ factor = \frac{\#\ of\ data}{\#\ of\ tags}$$

We also have to consider the number of words in the data and the number of words in the tag. The higher these values the more parsing the document requires and the more unstructured it is.

$$word\ factor\ for\ data = Avg\ number\ of\ words\ in\ each\ data$$

$$word\ factor\ for\ tags = Avg\ number\ of\ words\ in\ each\ tag$$

From these observations we design the following quantitative measure for structure:

$$structure = \exp\left(-\frac{\#\ of\ data}{\#\ of\ tags} * \frac{\#\ of\ words\ of\ data}{\#\ of\ data} * \frac{\#\ of\ words\ of\ tag}{\#\ of\ tags}\right)$$

The range for this equation is from 0 to 1, where 0 for unstructured text and 1 for a very structured document, such as the theoretical ontology that defines natural language.

We examined fifty (50) XML and HTML documents and observed the following structure values:

XML: 0.670 - 0.079

HTML: $0.444 - 4.24 \times 10^{-18}$

We can see how this formula is very biased toward structured documents. The range for structured documents is much larger than that of unstructured documents. This is because an unstructured document contains text with little to no tags. Also since this thesis refers to more structured documents this scale is reasonable.

4.2.3. Case

Case is the first layer of abstraction in our document model. Case refers to the language in which the schema and the document are represented. More specifically it refers to the constraints on information imposed by the language. It defines the amount of structure in a document. We have categorized the possible cases in terms of the amount of structure they provide to a document.

We believe that while this model was built with semistructured documents in mind it is generic enough to also cover unstructured documents. This is the reason and advantage for having a case layer. The other layers of abstraction will apply across all documents so with a modification in the case layer this model should still apply across all languages [34].

This version of our model was implemented to apply to semistructured documents and schema. This includes XML based languages which offer some structure but also face many other interoperability problems that our model hopes to overcome.

4.3. Schema Data Model

In the previous sections 4.1 and 4.2 we considered the document and the next logical step is to look at the relationship between the data and the schema and this is described in this section. The second layer of abstraction in this model is the schema data model. The schema data model refers to the physical structure of the schema including the way that schema elements are grouped together.

The schema structure differs from the content model discussed in the following section 4.4 because a lot of the metadata/physical structure is omitted from the previously defined hierarchical ontology. This missing information is key and could provide valuable information to complete the model of the structured language. This includes information like units, restrictions and other characteristics of the data. Other data that can be important and should be distinguished in the model is the structural data. As shown below while the document on the right has the same information as the one on the left there are structural differences that should be noted in our model of the documents.

Document 1

```
<Banker>
      <FName></FName>
      <LName></LName>
</Banker>
```

Document 2

```
<Banker type="PersonInfo"></Banker>
<PersonInfo>
      <FName></FName>
      <LName></LName>
</PersonInfo>
```

Since most of this information is already represented in the hierarchical structure of the XML Schema it is logical to represent this information also in a hierarchy. It can be combined with the previous ontological hierarchy to create a cohesive schema model.

Another example of groupings in the xml document is the use of the extension base element. The extension element creates a new element by extending another element and adding attributes. An example of this element is shown below in the original xml schema language [1]. When this extended element is used in a document it loses the connection that tells us what element it is inherited from. This however is an important characteristic of a schema.

```
<xs:complexTypename="personinfo">
  <xs:sequence>
    <xs:elementname="firstname"type="xs:string"/>
    <xs:elementname="lastname"type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:complexTypename="fullpersoninfo">
  <xs:complexContent>
    <xs:extensionbase="personinfo">
      <xs:sequence>
        <xs:elementname="address"type="xs:string"/>
        <xs:elementname="city"type="xs:string"/>
        <xs:elementname="country"type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

In the implementation section we consider the schema and identify a way that elements are inherited across schema.

It is important to know schema structure because the way elements are grouped together tells us a lot about the author of the document. Some interesting characteristics can arise from studying the inheritance pattern of schemas.

4.4. Content Model

Constructing a hierarchy based data model gives us a clear visual data representation of structured data. By comparing hierarchies of concepts we can compare and convert between structured languages to create interoperability.

In this research we are considering structured languages that are composed of a nested structure of tags e.g. XML. therefore a hierarchy is an obvious choice for a basic data model. However in lieu of tags XML can also be represented as a hierarchy of concepts. A concept in this hierarchy represents a discrete idea behind the tags. A concept may represent an abstraction or generalization of a set of other concepts. To simplify this hierarchy and make it easier to process we have restricted these concepts to be represented by a single word or a compound phrase (e.g. namespace).

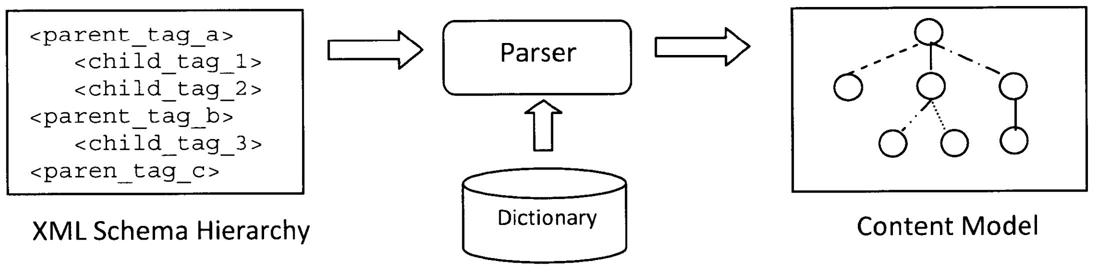


Figure 4. Process of mapping a xml schema hierarchy to the content model utilizing the dictionary

The parent-child relationship in the data hierarchy maps to the relationship between the parent and child nodes in the content model so in structure the two models are similar. However the content model leverages the dictionary defined in Section 5 as a semantic base to add meaning to the xml tree hierarchy. By using the relationships described in the dictionary and enforcing it in the model we add semantic and structural information to the XML schema.

4.4.1. XML Model

There is no real definition of a parent- child relationship in a structured language schema. This is partly there to allow flexibility in the way an XML document is used. However it is also one of the reasons there an XML document can be so ambiguous. This loosely defined parent-child relationship means that there are multiple ways to represent the same information using different parent child node pairs.

While an XML schema can be derived with any parent-child relationship, logical or not, we make one main assumption about the schema. In a proper schema the children nodes describe a consistent aspect of the parent node. In a well made XML schema the child node describes some aspect related to the parent node and the siblings of the child node add to the description of this aspect.

By applying the content model and the m-language dictionary to such an ambiguous relationship we hope to bring more semantic information to the structure and meaning of the XML hierarchical.

4.4.2. *Content Model*

Within the content model hierarchy the parent-child relationships can be one of the following four forms: an abstraction, an aggregation, attribute or an association. In this section we describe the content model relationships and describe some ways in which the XML hierarchy maps to the content model relationships.

4.4.2.1. *Abstraction*

An abstraction relationship, also known as a 'type-of' relationship, is where the parent concept is a generalization of the child node. The figure below shows an example of such a relationship. This relationship is also synonymous with the 'instance-of' relationship where the child is an instance of the parent.

```
<Account>
   <Checking>
   <Savings>
   ...
```

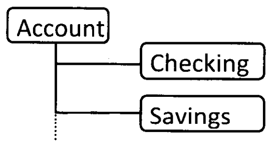


Figure 5. An example of an Abstraction relationship in XML schema (left) and in the content model (right)

This relationship is a one-way mapping relationship. This means that if we have a pair or concepts, parent and child that have a type-of relationship they will only occur in the same order in an xml hierarchy. In other words, the parent node in the content model will also only occur as the parent node in the xml hierarchy. The child node is an instantiation of the parent so given our assumption in section 4.4.1 we can say that if the two are connected to each other in a hierarchy the content parent will be the xml parent node and similarly with the child node.

4.4.2.2. Aggregation

An aggregation relationship, also known as a 'part-of' relationship is where the child concept is a subset of the parent. This can either be a physical subset or a subset of a certain group. By a similar argument as in section 4.4.2.1 this relationship is also a one-way relationship.

```
<Bank>
    <Branch1>
    <Branch2>
    ...
```

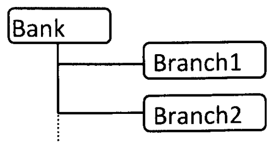


Figure 6. An example of an Aggregation relationship in XML hierarchy (left) and in the content model (right)

4.4.2.3. Attribute

An attribute relationship, also known as a 'has-a' relationship is where the child concept is an attribute of the parents'. The idea of attribute is a vague notion. The Merriam Websters dictionary describes attribute as "an object closely associated with or belonging to a specific person or thing". An attribute of a concept can be any other concepts that define it. This is also a one-way mapping relationship.

```
<Account_holder>
    <Name>
    <Age>
    ...
```

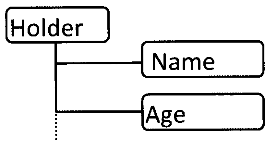


Figure 7. An example of an Attribute relationship in XML schema (left) and in the content model (right)

In the example above, in figure 4, the attributes of the account holder are name and age. Name and age describe an aspect of this account holder who is a person. Such information is critical to understand yet very difficult to infer. Deriving these attribute relationships in the dictionary is a challenge. We discuss some techniques to achieve this in section 5 of this thesis.

4.4.2.4. Association

An association relationship, also known as a 'associated-with' relationship is where the child concept has a semantic or concept relation to the parents. More generally, the child concept is associated with the parents'.

```
<Bank>
   <Account>
   <Interest>
   ...
```

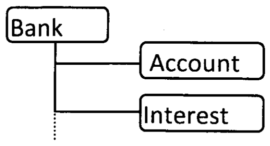


Figure 8. An example of an Association relationship in XML schema (left) and in the content model (right)

The association is a two-way link. In the example above we can order the data differently because of this two-way association. The example orders the information by bank but if we rotate the nodes we can order it by account and the concept will consist of the bank attached to each account. This feature allows the content model the flexibility in comparing different schema based on the same information. In the parser we merely consider any one ordering of the nodes. However when using the model to compare schemas we can use this measure to look at the structural and ordering differences in schema and compare them.

4.4.3. Parser

While a human could easily identify a specific XML concept in text and create a hierarchy based on the information, there is no obvious way to automate this process. In this section we will discuss the various methods to parse the xml hierarchy to the content model and the challenges in automating this process.

To parse an XML schema hierarchy into the content model the following steps need to be followed:

1. Breaking up tags into concepts

2. Determining the relationship and order between the concept nodes

4.4.3.1. Parsing tags to concepts

In the content model proposed, the hierarchy is made up of single word or compound phrase concepts. However most XML tags schema tags cover a wide range of formats from simple words to compound phrases to complex sentences to phrases containing conjugations, acronyms and abbreviations. A combination tag can be represented as a hierarchy of concepts where each concept is a single word from the tag.

To parse more complex tags, first the tags must be broken down into words. Acronyms and abbreviations must be expanded so we are left with a number of concepts that make up the tag. Extraneous and redundant concepts resulting from this parsing must also be removed. The example below shows this basic conversion.

<accounttypecodesimpletype> $\Rightarrow$ <account> <type>

Figure 9. Example 1 of parsing a tag into concepts and removing redundant and irrelevant words

There are many ways to split a tag into individual words. In some contexts a tag can be split by special characters such as underscores or capital letters. In the example above where there are no clear ways to divide the words we use the dictionary and run a pattern matching algorithm to identify the words. Since the dictionary contains context specific information it can also be used similarly to identify acronyms and abbreviations.

In the example above, there is a difference between the original tag and the resulting concepts. There is some information lost in this conversion. 'SimpleType' has been dropped from the tag. A simpleType element constrains the value to a built-in datatype. This information is only relevant to the XML structure and in the context of the concept hierarchy it is not necessary. The word 'code' in this instance implies that the information is a token. Again in the context of the concept hierarchy this is not relevant. In the implementation we automate such a conversion so that these irrelevant key words are identified and then removed.

Many words are irrelevant and are eliminated when parsing a complex tag. The challenge is to remove these extraneous words while maintaining the integrity of the concept. Some of the words eliminated add XML information to the tags. For example 'complextype' and 'simpletype' explain the structure of the XML tag elements but bear no relevance to the concept that is described by the tag. Any data which refers strictly to the structure of the XML tag (e.g. length of data, form of the data) is not relevant to the concepts and can be eliminated.

Other words that are eliminated when creating the hierarchy are redundant words. The example below has the words 'category' and 'type' in the tag. These two words are synonymous and therefore one can be removed creating a more efficient hierarchy without losing the concept. By manually training our tool on many schema we can identify the redundant words and word pairs and create features to automatically remove them in the parsing process.

<AccountCategoryType> $\Rightarrow$ <Account> <Category>

Figure 10. Example 2 of parsing a tag into concepts and removing redundant and irrelevant words

4.4.3.2. Ordering Concept Nodes

One of the biggest challenges in automating the process of creating a concept hierarchy is the ordering of the concept nodes in a combination tag. The parent-child relationships are defined as being one of four types 'type-of', 'part-of' , 'has-a' and 'associated-with'. Therefore once we break the tag up into its composite words and eliminate redundant words we can study the words to see which words has one of these relationships with any of the other words.

The advantage the dictionary and the content model having consistent formats is that it facilitates the mapping between these structures. We can map the word to the dictionary concept, which immediately gives the word meaning and converts it from a simple tag to a concept with a semantic structure. We do this mapping for each of the words and consider the relationship between the resulting dictionary concepts. Because the relationships between the words in the dictionary are identical to the parent-child relationships defined by the content model ('type-of', 'part-of' , 'has-a' and 'associated-with') we can find the parent-child relationships between the words.

In the first example, figure 5, since the concept 'Account' is in a 'has-a' relationship with the child concept 'Type' and 'has-a' is a one-way mapping relationship we can order the concept nodes easily. Therefore if we have a database with all these word associations we can order the hierarchy correctly. The dictionary provides such connections so we use the word relationships provided by the dictionary to find the relationships between all words in a sub tree.

To order the nodes we can work in a bottom-up manner, starting the concepts in the leaves of the XML schema and creating an ordering for a subtree. We can leverage dynamic programming to make the matches bottom up and speed up this algorithm. We run dynamic programming on an XML tree to maximize the number of agreeing relationships provided by the dictionary that fit the concept model.

To create the most efficient concept hierarchy, redundancy has to be removed in the nodes across multiple levels. After the concepts and relationships are assigned by the dynamic programming algorithm we do another pass to remove redundancy. It might be necessary to do several iterations of ordering and redundancy removal till the concept hierarchy stabilizes. In the example below the second stage has a redundancy because the node <Name> is repeated. It is simple to combine these two nodes to make a more efficient hierarchy.

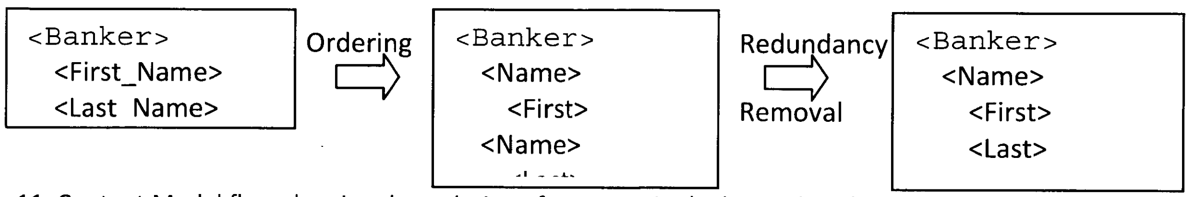


Figure 11. Content Model flow showing the ordering of concepts in the hierarchy. The second step shows removing redundancy

4.5. Data Storage Model

In structured data interoperability most existent systems merely compare data elements at the instance level. However this is a very 1-dimensional view of the data model. The data structure model is the last layer of abstraction in the document model. It provides a higher level view of the data elements.

The thought experiment in section 4.1 described the steps to building a conceptual model by backtracking the creation process. The first step of the creation process was to study how the data given can be stored together in a more structured format. In other words how data points can be pulled together into a node, table, database etc. Accordingly, the final abstraction layer of the document model is the data structure layer.

Most structured data sources, such as xml, are used as communication tools for data that is already held in other formats like free text, databases, tables or hierarchies. The data held in these storage formats are encoded for communication across a channel. The data stucture layer identifies the most likely data storage structures that the data in the document was designed to be placed into. These underlying data structures can be just a hierarchy, or databases: relational, hierarchical and object-oriented.

One approach to identify the underlying data structures is to first classify the document as either document-centric or data-centric. Documents are designed with a specific underlying data structure and classifying the document is the first step to revealing these structures.

4.5.1. Spectrum of Data Storage Documents

The structure of a document as defined in Section 4.2 gives us a numerical value from 0 to 1 as to the amount of structure in the document. This thesis is only concerned with documents that are in the semistructured to structured range. Within this range of documents we can define a spectrum of document storage formats ranging from a hierarchy to object-oriented, hierarchical and relational databases.

In the diagram below we show how the spectrum of data storage structure maps to the spectrum of case.

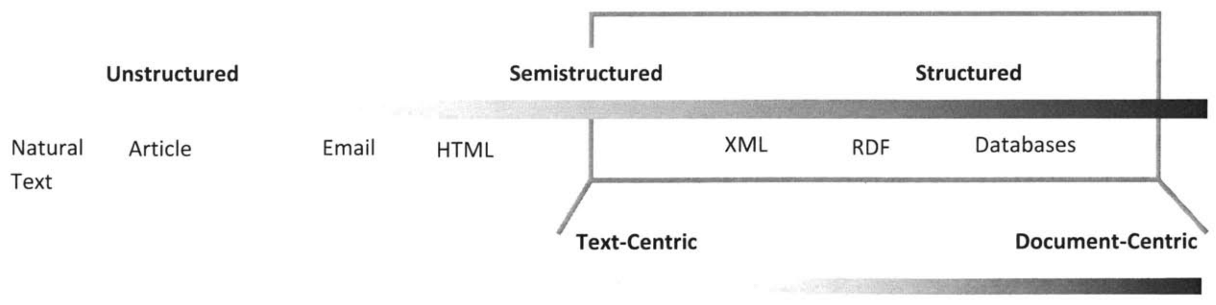


Figure 12. Mapping between the spectrums of the case of a document and document structure format

The figure above shows us that there is a link between the structure of the document, as we defined in Section 4.2, and the data storage model. In the following sections we will discuss how we can use the variable of structure to help determine the data storage model of the document.

The challenge is to correctly identify the document as either text-centric or data-centric and within each area recognize the data structure used. Data-centric documents can contain elements of text-centric documents and vice-versa. Most documents are some combination of the two, for example most simple HTML web pages are both text-centric and data-centric. It is possible however to separate the characteristics and use the value of structure we derived in section 4.2 to make a guess about the type of document and identify the underlying data structure.

4.5.2. Text-Centric Documents

A text-centric document, as shown in the figure below, is made to be processed manually and requires a lot of user input. There are certain characteristics that can identify a text-centric document. The XML tags defined in the schema are more descriptive so a human can understand their meaning, the structure of the schema is more irregular and a large portion of the data is unbroken text. Text-centric documents are designed so the inherent data can be processed by humans and therefore the underlying data structure is either a readable hierarchy or a native XML database.

```
<Para>Account choices:</Para>
   <List>
   <Item><Link URL="checking.html">Open a checking account in the state of
Massachusetts linked to your original bank account</Link></Item>
   <Item><Link URL="savings.html"> Open a savings account in the state of
Massachusetts linked to your original bank account </Link></Item>
   <Item><Link URL="new.html">Open a new account </Link></Item>
   </List>
```

Text-centric information is not made to be stored in any other format other than its inherent hierarchical view. It is meant to be stored in the document hierarchy and the document instead of a storage document is mean to be parsed.

As we can see a text-centric document is a less structured document. Therefore we can refer to the quantitative representation of structure we derived in section 4.2. In the measure of structure we also defined the word factor as being the average number of words in each data. We can use this measure to estimate if a document is text-centric. On average text-centric documents have a score of 5 for the word factor. We use this measure in defining a document as a text centric document.

4.5.3. Data-Centric Documents

A data-centric document, as shown below, is designed to be processed by a computer. It can be easily identified because of its very regular structure and finely-grained numerical or non-text attribute values.

```
<Library>
      <Book>
         <Name>ABC</Name>
         <Author>Alice Bard</Author>
         <Editor>Clara Dan</Editor>
      </Book>
      <Book>
         <Name>The Farm</Name>
         <Author>Han Green</Author>
         <Editor>Clara Dan</Editor>
      </Book>
</Library>
```

The underlying data structures of a data-centric document are mostly databases: relational, object-oriented or hierarchical. We can differentiate between the type of database by looking at the tag and the data. Therefore the measure of structure will become useful in identifying the type of data storage structure underlying these documents.

When presented with a document we first identify the document to be a data-centric document using the word factor of 5. Other factors that can be helped to identify a data-centric document is the amount of tag repetition. If the same tag is repeated multiple times it is probably used to define a column in a database. Then for the documents defined as data-centric we use the techniques described below to define the underlying structures.

4.5.3.1. *Hierarchical Databases*

A hierarchical database in one in which the data is organized into a tree like structure. It is used to define relationships which have a 1:N relationship. A hierarchical database structure interfaces very efficiently with XML because of their similar hierarchical structures [36]. It is easy to convert data from a hierarchical database to XML format. Therefore it becomes a preferred format for data storage.

To identify an XML document as a hierarchical database we consider the hierarchy factor. The hierarchy factor defined in section 4.1.3 gives us an idea of how many layers of hierarchy is used to define each data piece. The range of this value is from 0 for structured documents to approaching infinity for unstructured documents. If this value is approaching 0 we know that the underlying data structure is a hierarchical database.

The challenge of using this method is that there are shallow hierarchical databases and it is difficult to distinguish an underlying shallow hierarchical database from an object-oriented database.

4.5.3.2. *Object-Oriented Databases*

An object-oriented data structure would be necessary for documents where the data is mostly composed of objects. The schema can be identified because it would contain many wrappers to define these objects. As we mentioned before, in some contexts it is difficult to differentiate an object-oriented database in XML from a hierarchical database.

One way to tell a difference is that an object will most likely have an ID related to it that can be used to reference it. If we can identify such a tag around an object we can identify the underlying structure as an object-oriented database.

In our implementation we understand that due to the similar structures there is a margin for error in recognizing these databases.

4.5.3.3. *Relational Databases*

A majority of current schemas are defined to hold data for relational databases. The example below shows one such document. We can distinguish it as a document designed for a relational database because of the multiple references to color by means of a unifying code. A lot of such documents contain codes that point to a token that is referenced through the entire document.

To identify an underlying relational database we can consider the tags in the schema. If the exact same tag names are referred to in different parts of the schema we can consider this a relational database. In the document we can check is in different blocks there are similar values for similar tags. This would tell us that the underlying data structure is a relational database.

```
<ParkingLot>
      <Car>
         <Model>Ford</Model>
         <ColorCode>A12</ColorCode>
      </Car>
      <Car>
         <Model>Toyota</Model>
         <ColorCode>A14</ColorCode>
      </Car>
</ParkingLot>

<Colors>
      <Color>
         <Code>A12</Code>
         <Name>Red</Name>
      </Color>
      <Color>
         <Code>A14</Code>
         <Name>Blue</Name>
      </Color>
</Color>
```

5. Dictionary

The previous section defines a model for parsing different structured languages. To add semantic meaning we need a common semantic base that is computer understandable. Such a system also lends itself to many applications in knowledge based inference and problem solving [37].

For interoperability between two structured sources we need to find correspondences between the data. In Chapter 4.1 we define these layers across which we need to create the correspondence: the syntactic case layer, the data structures and the semantic schema content layer. In the case layer we derive the correspondences are simple to find, if the syntactic language of the schema are similar or not. At the semantic content layer, it is more complicated to identify the correspondence across layers. Synonyms, attributes, associations and other relationships between words need to be considered when matching two tags across schema. Therefore, we need to be able to represent the real world semantics of English in a computer interpretable form [38].

Having a computer understandable base across which to compare terms facilitates interoperability. There is no need for a user to spend the time to understand source schemas anymore if a computer can understand and integrate them. In this chapter we derive the characteristics necessary for such a semantic base. We also consider the advantages and disadvantages of WordNet for this purpose. We design our own system, M-language, to compensate for the disadvantages.

5.1. Semantic Requirements

It is necessary to first derive a set of minimum semantic conditions for the semantic base. This is a list of the minimum characteristics that our system needs to semantically serve as an appropriate base.

The basic requirement is to be able to create associations between two related tags. Given two tag names the base should able to decide if they should be a correspondence between them. There are two levels semantic relationships to consider in a hierarchy of tags:

- Tag-to-tag relationship – To find out if two tag words are associated with each other we need to decide if they mean similar things. If they have correspondingly similar information attached to each. The simplest correspondence is that between two exact words in different schema. A more complex correspondence is between two words that are synonymous or between two words correlated through a context. i.e. kimono and robe or kimono and Japan.

- Parent-child relationship – In a schema hierarchy the child describes an attribute of the parent. Therefore to understand the meaning of the root tag we also need to look at the children and what aspects of the root tag are being described. Often while two root tags may be highly correlated their children tags may be describing different aspects. In this context a correspondence would be difficult to make. An example of such a correspondence is shown below:

  Schema #1:

  ```
  <Bank_Account>
      <Site>
          <Id>
          <Address>
      <Interest rate>
  ```

  Schema #2:

  ```
  <Bank_Account>
      <Customer>
          <Name>
          <Address>
  ```

These requirements describe a system that to should be able to understand associations between words and attributes. I.e. we are trying to make the system understand the semantics of the English language.

Making a computer understand real world semantics has been a controversial problem. While the field of Natural Language Processing is aiming to build a computer that understands natural language, there are contradicting papers that show that this is not possible. The Chinese room argument [39] says that even if we were to come up with a program that generates natural language the computer is just following a set of rules. Therefore a computer can only ever apply rules and transform semantics; it can never really understand language. Both views have their merits and such a system that will refute either hypothesis hasn't been conclusively built.

Given the realm of this problem simplify our requirements. We reduce our problem from having to understand the semantics of the English language to a more manageable one. We only require our system to understand a tag or word as it pertains to other tags or words. This leads to another requirement, that it be semantically self-describing. That the definition of a word is based on other words. To describe a word by its relationship to other words is the basis of a common dictionary. So when trying to correlate two tags we just need to see if one tag exists in the definition of the other tag.

So our goal is to use a system that is computer understandable and models the semantics of the English language for the purpose of creating correspondences between schema tags. We propose that a dictionary or a similar system that defines the relationships between words.

5.2. WordNet

WordNet was built on the assumption that if it was possible to build the correct patterns of semantic relations for a word, a definition can be inferred from it [40]. This is one of the requirements we defined in the previous section

WordNet has the following grammatical word categories: nouns, verbs, adjectives and adverbs. The semantic relationships described by WordNet are [41]:

- Synonym for words that are similar. WordNet groups words by groups of synonyms called synsets to represent word senses.
- Antonym for words with opposing meanings
- Hyponym meaning sub-name and its inverse hypernym for super-name helps organize the words into hierarchical relationships
- Meronym meaning part name and its inverse holonym meaning whole name. WordNet also distinguishes component, substantive and member parts
- Entailment relations between verbs

Currently Wordnet contains 155,287 words, 117659 synsets and 206941 word-sense pairs [42].

Originally, we used the WordNet corpus as the semantic base. It contains the main characteristics necessary; it is a large readily available, computer understandable language that provides the semantic relationship between words. To find the semantic relationships between the tags in schema and also correlating parent-child pairs we can look at similarities between words. A method for finding the similarity between two words in WordNet is derived in [43]. Another advantage is that WordNet is related to our content model. The hyponym relationship is similar to the 'type-of' relationship described earlier in chapter 3.3 and the meronym to 'part-of'. This would make it easier to incorporate our model with WordNet.

However WordNet has certain disadvantages. WordNet creates similarities between noun pairs and verb pairs but not across them, it cannot relate noun and verb pairs that may highly correlate e.g. murder and gun [43]. Also while WordNet contains adjectives and adverbs these are not organized into hierarchies so it is hard to find correlations within these word groups[43]. Another disadvantage is that WordNet encodes sense distinctions that are too-fine grained. These synsets are too narrow and therefore the WordNet net is very deep but not broad enough. There are fewer correlations between words than actually exist.

These issues, especially the last one, made WordNet an unsuitable choice to use. It is imperative for there to be all the available correlations between two words. We therefore decided to create our own semantic base that would incorporate the concepts we need to our model.

5.3. M Language Dictionary

The M Language dictionary is an effort by the Data Center at the Massachusetts institute of technology to create a semantic knowledge base.

We created a new 'dictionary' by usage frequency subsets and domain of use. Schemas are domain specific and therefore restricting it to a specific domain makes correlations more correct. By separating our base out into domains it gives us a way to integrate context-specific words. In WordNet, for example, common terms such as 'frog' and 'animal' were included with words from the scientific context such as 'animalia' and 'cordata'. This is an advantage over the WordNet system which had very narrow categories and within the categories maintains a large set of terms.

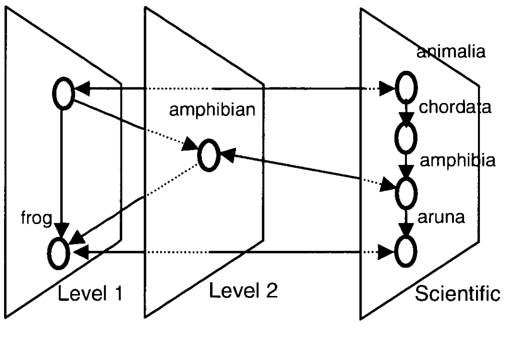


Figure 13. The common dictionary groups words into varying levels of complexity as well as different concepts. Ontological relations links words across these levels.

Ontological relationships in this dictionary included connections both within and across vocabulary subsets. A domain specific vocabulary linked terms both within and across vocabulary subsets using *typeof*, *partof*, *attributeof* and *associatedwith.* These are the semantic relationships based on the content model we described in Section 4.4 and therefore the dictionary and the model should integrate very well.

Words were scoured and organized according to domain from children's literature and Simple English articles for the core vocabulary, news articles and Wikipedia entries for common terms and domain specific terms from engineering, science and governmental sources.

The M language dictionary is an evolving work. The goal is to use it not only in the context of structured documents but also for unstructured text. The dictionary is constantly evolving to fit the developing model of the unstructured parser. We have used the initial versions of the dictionary to implement this research but the concepts still apply to any newer versions.

6. Implementation and Evaluation

This section presents the modeling tools that developed and implemented based on the conceptual model previously described. The dictionary is the basis for this conceptual document model. Section 6.1 considers the implementation of the dictionary. It also discusses some methods that could be used to evaluate the thesis and the results we would expect. Section 6.2 examines the basic document model tool. This is an implementation of the aspects of the document model – the case, schema model, content model and data storage model.

6.1. Dictionary

The dictionary is a continuing effort of the Data Center at MIT to create a system that gives semantic meaning to words by their relationship to other words. This thesis uses only a second version of the dictionary with a limited number of words and concepts.

The version of the dictionary used by this research has about 5,000 words all of which are nouns or adjectives. The representation of verbs has not been discussed firstly because such a representation has not been created. Secondly the majority of schemas we have encountered do not involve verbs.

Since the dictionary is not exhaustive we cannot evaluate its performance without biasing the results to some extent. The M Language Dictionary is a graph in which the vertices correspond to concepts and edges to relationships between concepts. While this graph can be used in the document model to identify the relationship between two words, the graph is not exhaustive, and therefore can only be used to evaluate a portion of the model. However there are some evaluation methodologies that we have identified.

One of these evaluation algorithms involves comparing the M Language Dictionary to WordNet. Comparing the two vocabularies provides us with a potential evaluation metric. For example we can compare the path between two words in the M Language Dictionary to that in WordNet. Such a comparison would indicate the scope of the word by examining the connections in each dictionary.

Since the M Language Dictionary is not cover the entire scope of English, enough we cannot fully compare it to WordNet. However if we were to carry out the experiment described above there are a few results we would expect to see based on the nature of the M Language Dictionary:

- The M Language Dictionary is 'flatter' than WordNet, in that the ontological 'typeof' hierarchy in the M Language Dictionary is wider but shallower as compared to WordNet. This was in fact one of the objectives of the M Language Dictionary to have a more complete word relationship tree. So the path between two words in the m language dictionary will be a much bigger tree.
- There will be planes of context defined in the M Language Dictionary. The M Language Dictionary defines words by their context.

6.2. Conceptual Modeling Tool

As inputs the model requires a document and the corresponding schema. While there are tools to derive a schema from a structured document our tool requires the original schema. This is because given a schema it is possible to create two different documents with the same information. To eliminate such duplicity it is important to consider the original schema.

The tool parses the inputs to each of the sub-models: case, schema model, content model, data storage. Below is a screenshot of the tool. In the following sections we discuss the implementation of each of the sub-models and the way that they are implemented. We also provide screenshots of the tool for each of these examples.

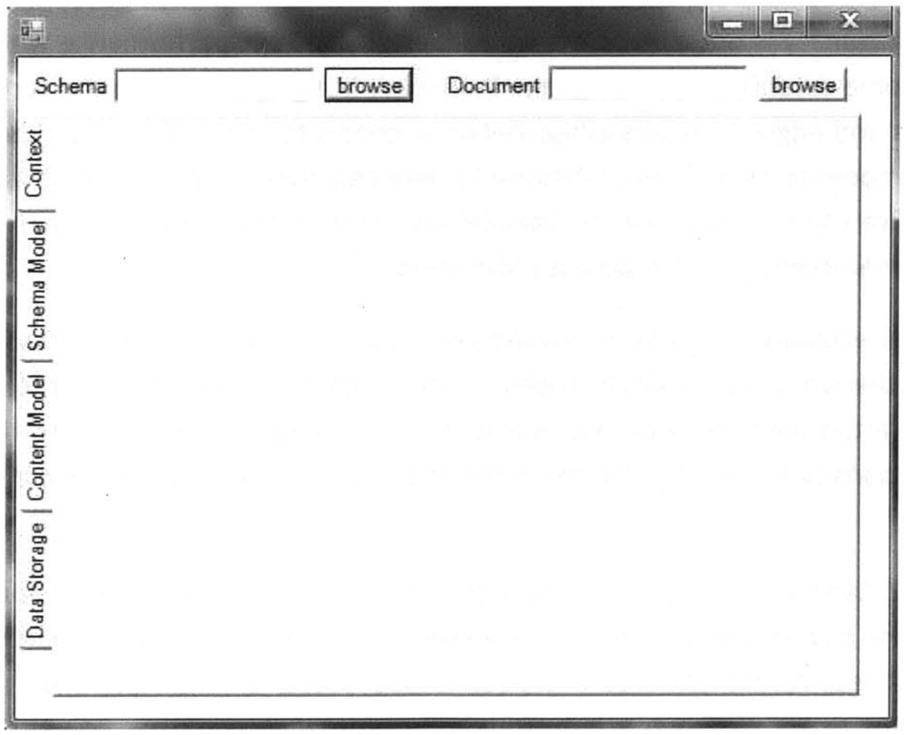


Figure 14. Screen capture of the modeling tool with no data loaded into it

6.2.1. *Case*

For this sub-model we implemented the structure metric described in section 4.2. Given the document, the modeling tool gives the value of structure as defined by the following equation:

$$structure = \exp\left(-\frac{\#\,of\ data}{\#\,of\ tags} * \frac{\#\,of\ words\ of\ data}{\#\,of\ data} * \frac{\#\,of\ words\ of\ tag}{\#\,of\ tags}\right)$$

The tool also places this value of structure on the spectrum of structure, from unstructured to structured documents. It also provides a guess as to the case of the document. The guess is based on a small subset of the documents that we analyzed. The values for the guesses are given below:

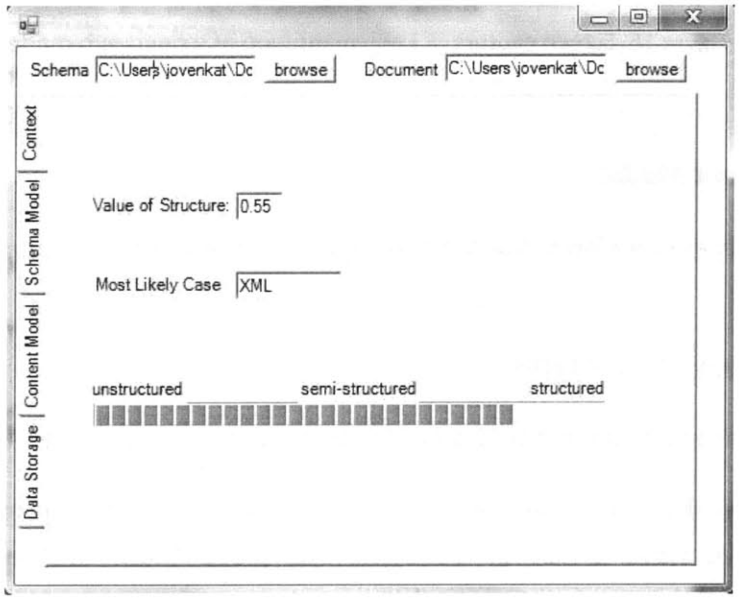
Text: 0

HTML: 0.444 – 4.24*10^-18

XML: 0.670 - 0.079

SQL: 0.542-0.998

This shows that the majority of the scale from 0-1 is focused on structured documents. Therefore the majority of the scale 0 to 1 is devoted to semi-structured and structured documents. However visually we represent the scale as a linear scale and merely adjust the values. We take our boundaries for unstructured, semi-structured and structured documents from the values above and between the boundaries, linearize the scale.



Schema: C:\Users\jovenkat\Dc

Document: C:\Users\jovenkat\Dc

Value of Structure: 0.55

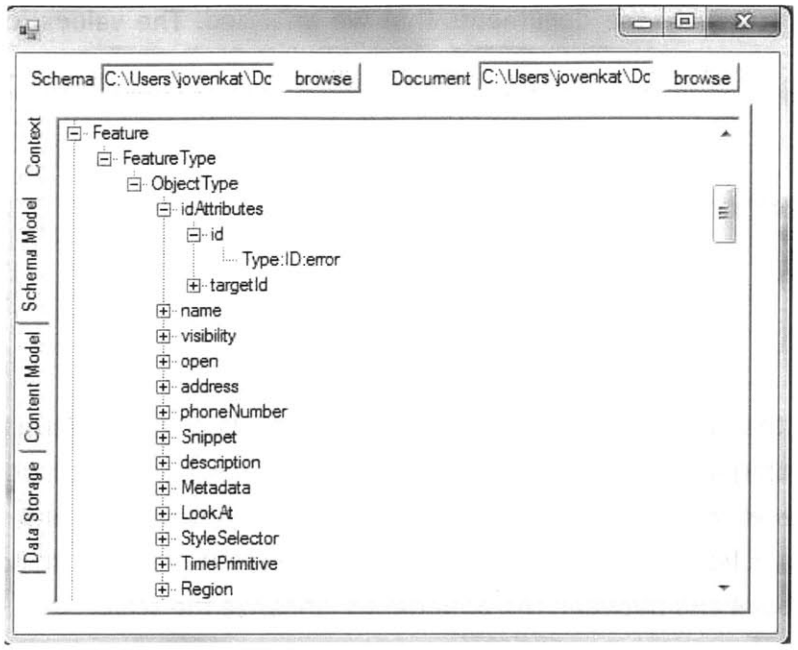
Most Likely Case: XML

Figure 15. Screen capture of implementation of case sub-model

6.2.2. *Schema Model*

The schema model examines the way that different elements of the schema are grouped together. Therefore for the implementation of this sub-model we parsed the schema and extracted all the structures. We are careful to keep the structures that each tag inherits from. This might involve tracing the inheritance through other schemas.

We also implement a function that allows the user to track where elements are inherited. By clicking on one element it highlights the other places where it is inherited.



Schema C:\Users\jovenkat\Dc
Document C:\Users\jovenkat\Dc

```
⊟ Feature
   ⊟ FeatureType
      ⊟ ObjectType
         ⊟ idAttributes
            ⊟ id
                  Type:ID:error
            ⊞ targetId
         ⊞ name
         ⊞ visibility
         ⊞ open
         ⊞ address
         ⊞ phoneNumber
         ⊞ Snippet
         ⊞ description
         ⊞ Metadata
         ⊞ LookAt
         ⊞ StyleSelector
         ⊞ TimePrimitive
         ⊞ Region
```

Figure 16. Screen capture of implementation of schema sub-model

6.2.3. *Content Model*

To parse a schema hierarchy from the schema model to the content model the following steps were implemented:

1. Breaking up tags into concepts
2. Determining the relationship and order between the concept nodes

For the first step we have to split the tags into words and match them to concepts in the dictionary. Once we have a list of concepts we place them in the dictionary to find relationships. By matching relationships between concepts we are ordering the concept hierarchy. In cases where two concepts aren't matched by a single relationship link we copy the list of links connecting the two concepts. When there are multiple paths showing different relationships between two concepts we simplistically choose the smallest path.

We create orderings for the concept hierarchy bottom up. We start with the concepts in the leaves of the schema hierarchy and create an ordering for each subtree. We can leverage dynamic programming to optimize this algorithm. We run dynamic programming on a concept hierarchy to maximize the number of concept relationships provided by the dictionary.

In the implementation we use colored text of the child node to represent the relationships between parent and child. Red, blue, brown and orange represent 'type-of', 'part-of', 'attribute-of' and 'associated-with respectively.

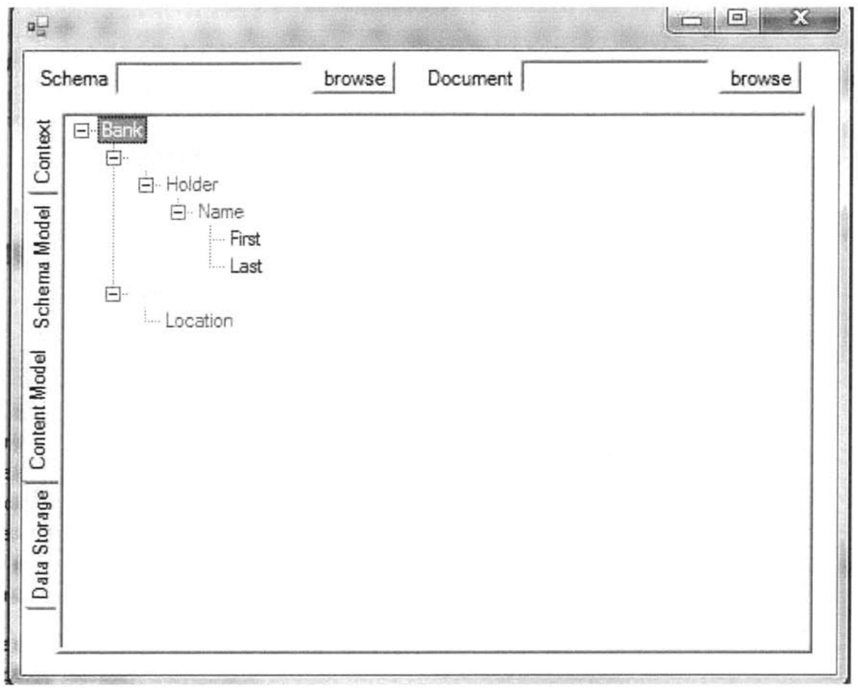


Figure 17. Screen capture of implementation of schema sub-model

6.2.4. *Data Structures Model*

The data structures model classifies the document as text-centric or data-centric documents. We were able to define this by using the word factor value. Within data-centric documents we also wanted to consider if the underlying data structures were relational, hierarchical and object-oriented databases. We implemented an log-linear algorithm for classification and used features to distinguish the different types. The features are described in the section 4.5.3. For example the features for hierarchical factors included the hierarchy factor as we described in 4.5.3.1. The hierarchy factor is defined as:

$$hiearachy\ factor = \frac{\#\ of\ data}{\#\ of\ tags}$$

We also have features for object-oriented and relational databases. We train the model on a small number of documents that we know are created from a certain database. The number is low because it is difficult to know the underlying data structures of a structured document. Creating a schema for training would be biasing the result.

Based on this small training set we train the algorithm to classify the document as having one of the following underlying structure: hierarchical, object-oriented or relational.

7. Conclusion

This thesis tackles the problem of interoperability of structured documents. We approach this problem in a novel way, by proposing an underlying conceptual model that captures the knowledge represented in a structured document.

A model of structured documents must consider both data and schema. An important characteristic that must be considered include the 'case' or the language in which the document is written. In fact, the language used by the document constrains and even dictates the knowledge that may be conveyed. In addition, the schema structure, the content or the ontology of the schema, and the underlying data structures are also critical factors in characterize a document. These characteristics provide a framework for a comprehensive model of structured documents. This thesis proposes a novel dictionary to be used as a common semantic base that gives meaning to the meta-data in a structured document. This model was used to compare and contrast across multiple layers of abstraction.

We proposed a four level abstraction model – case, schema structure, content model, and data storage – as described in this thesis. We then developed algorithms to model structure documents at each level of abstraction. Finally, automated tools were developed and implemented to analyze each layer of the conceptual model for a given the document and the corresponding schema.

This thesis presents a novel approach to interoperability. It provides a simple conceptual model that abstracts the information within a structured document. Unlike other methodologies which require a knowledgeable user to extract information from the document, our method makes the document computer-understandable. The process of automatic interoperability can be advanced by mapped structured documents to our conceptual model and applying the automated comparison and integration tools.

8. Future Work

The techniques developed here are an attempt to transform multiple, disparate structured data sources into a common conceptual model. This consists of deriving a knowledge representation for these structured data sources that retains the underlying information while giving it context. By using the dictionary we place data in the context of other data and give this information meaning.

The next step is to expand the conceptual model presented in this thesis to parse unstructured and semi-structured data sources. Our future goal is to create a representation that not only models structured data but also unstructured and semi-structured data. The current effort of the Data Center at MIT is the development of such a system that defines a conceptual representation that parses all documents. The goal is to have a system that can parse information from any data source and provide tools for data manipulation, storage and knowledge extraction.

The versatile tool described above can be used for applications beyond just data management. It can be used as a sophisticated chat bot that parses different forms of input and uses the knowledge representation to understand the information and respond appropriately. A structure for such a chat bot, utilizing the conceptual model defined here, was derived by the Data Center at MIT and a simplified version of this structure is shown below. The structured data parser represents the implementation of the model presented in this thesis. We are working on finalizing this structure and implementing the different parts of it.

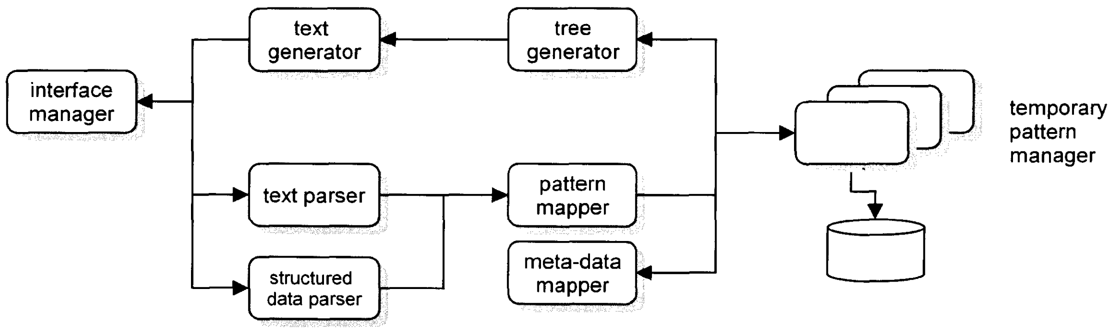


Figure 18. Chat bot Architecture based on the structured data parser presented in this research

9. References

[1] L. V. S. Laksmanan, F. Sadri. Interoperability on XML Data. In *Proceedings of the International Semantic Web Conference (ISWC)*, Lecture Notes in Computer Science, 2003, pp.146-163

[2] A. Halevy, A. Rajaraman, J. Ordille. Data Integration: The Teenage Years. In *VLBD '06 Proceedings of the 32nd international conference in Very large data bases*, 2006

[3] S. Madnick. The misguided silver bullet: What XML will and will not do to help information integration. In *MIT Sloan Working Paper No. 4185-01; CSL Workign Paper No.01-08.* http://ssrn.com/abstract=2818239 (Last accessed December 2010)

[4] D. Megginson. *Structuring XML documents.* Prentice Hall. 1998

[5] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches.

[6] E. Bertino and E. Ferrari. XML and database integration. In *IEEE Internet Computing*, 2001, pp. 75–76.

[7] M. Lenzerini. Data integration: A Theoretical Perspective. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, 2002, pp. 233-246

[8] L. Xu and D. W. Embley. Combining the best of global-as-view and local-as-view for data integration. In *Information systems technology and its applications, 3rd International Conference ISTA '2004*, 2004, pp. 123-136

[9] US Government. National information exchange model. http://www.niem.gov/

[10] A. Powell and P. Johnston. Guidelines for implementing Dublin Core in XML. http://dublincore.org/documents/dc-xml-guidelines/. Last Accessed November 2010

[11] B. Smith, L. Vizenor and J. Schoening. Universal Core Semantic Layer. In *Proceedings of the 4th conference on ontology for the intelligence community (OIC)*, 2009

[12] D. Calvanese, D. Lembo, and M. Lenerini. Survey on methods for query rewriting and query answering using views. Technical report, University of Rome, Roma, Italy, April 2001.

[13] P. Mork, L. J. Seligman, M. Morse, A. Rosenthal, C. Wolf, J. Hoyt and K. Smith. Galaxy: Encouraging Data Sharing Among Sources with Schema Variants. In *International Conference on Data Engineering (ICDE)*, 2009

[14] D. Webber. NIEM Canonical XML Dictionaries and Rule Engine Systems. In *Semantic Web Rules,*2010, pp. 2-15

[15] P. Mork, L. J. Seligman, M. Morse, A. Rosenthal, C. Wolf, J. Hoyt and K. Smith. Information Interoperability and Provenance for Emergency Preparedness and Response. MITRE Report, 2009

[16] P. Liang, K. He, B. Li and J. Li. Interoperability test of ebXML e-business solutions. In *Computer and Information Technology, 2004. CIT '04*, 2004, pp.1004-1007

[17] M. Daconta. UCore: The twitter of information sharing. http://www.gcn.com/Articles/2009/06/15/Reality-Check-commentary-UCore-info-sharing.aspx/. Last accessed November 2010

[18] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. Presented at the ACM SIGMOD Workshop on Management of Semistructured Data, 1997

[19] T. Critchlow, M. Ganesh, and R. Musick. Automatic generation of warehouse mediators using an ontology engine. Presented at the Proc. 5th KRDB Workshop, 1998.

[20] T. Catarci, L. Iocchi, D. Nardi, and G. Santucci. Conceptual views over the web. Presented at the Proc. 4th KRDB Workshop, 1997.

[21] J.-R. Gruser, L. Raschid, M. Vidal, and L. Bright. Wrapper generation for web accessible data sources. Presented at the CoopIS, 1998.

[22] L. Lakshmanan, and F. Sadri. Interoperability on XML Data. Proc. 2nd International Semantic Web Conference, 2003

[23] E. Rahn and P. A. Bernstein. A survey of approaches to automatic schema matching. In *Very Large Database J.,* 2001, pp.334–350

[24] Stylus Studio, XML Schema Mapper. http://www.stylusstudio.com/xsd_to_xsd.html/. Last Accessed October 2010

[25] R. Fagin, et al.. Clio: Schema Mapping Creation and Data Exchange. In *Conceptual Modeling: Foundations and Applications*, 2009.

[26] A. Tolk and J. A. Muguira. The levels of conceptual interoperability model. Presented at the *Fall Simulation Interoperability Workshop*, 2003

[27] K. F. Abdalla. A model for semantic interoperability using XML. In *Systems and Information Engineering Design Symposium*, 2003

[28] D. Pfeiffer and A. Gehlert. A framework for comparing conceptual models. In Proceedings Workshop on enterprise modelling and information systems architectures (EMISA), 2005, pp. 108-122

[29] G. Guizzardi. On Ontology, ontologies, conceptualizations, modeling languages and (Meta) Models. In *Databases and Information Systems IV*, Seventh International Baltic Conference, 2006

[30] S. Ullmann. *Semantics: An Introduction to the Science of Meaning*, Basil Blackwell, 1972

[31] G. Weglarz. Two worlds of data – unstructured and structured. In *DM Review Magazine*, 2004. http://www.dmreview.com/article_sub.cfm?articleId=1009161/. Last accessed November 2010.

[32] S. Grimes. Unstructured Data and the 80 percent rule. In *Clarabridge Bridgepoints*, Issue 3, 2008. White Paper.

[33] P. Buneman. Semistructured Data. In *Symposium on Principles of Database Systems*, 1997

[34] s. Abiteoul, P. Buneman, D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufman, 2000

[35] XML Schema extension element. http://www.w3schools.com/schema/el_extension.asp . Last accessed December 2010

[36] R. R. Some, A. Czikmantory. XML Hierarchical Database for Missions and Technologies

[37] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam and S. Slattery. Learning to construct knowledge bases from the World Wide Web. In *Artificial Intelligence*, Volume 118, Issues 1-2, 2000, pp. 69-113

[38] D. Pfeiffer and A. Gehlert. A framework for comparing conceptual models. In *Proceedings workshop on enterprise modelling and information systems architectures (EMISA)*, 2005, pp. 108-122

[39] J. Searle. Minds, brains and programs. In *Behavioral and Brain Sciences*, 1980, pp. 417-424

[40] C. Fellbaum. *Wordnet: An electronic lexical database*, 1998, MIT Press

[41] G. Miller. Wordnet: A lexical database for English. In *Communications of the ACM*, Vol. 38, Issue 11, 1995

[42] WordNet. http://wordnet.princeton.edu/. Last Accessed November 2010.

[43] T. Pedersen, S. Patwardhan, J. Michelizzi. Wordnet::Similarity – Measuring the relatedness of concepts. In *Proceedings of the Joint Human Language Technology Conference and Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, 2004, pp. 38-41