Extensible Document-Based Model Web Engineering

Jean-Marc Lecarpentier, Romain Brixtel, Hervé Le Crosnier and Cyril Bazin
Normandie Université, France
UNICAEN, GREYC, F-14032Caen, France
CNRS, UMR 6072, F-14032 Caen, France
Email: {firstname.lastname}@unicaen.fr

***Abstract*—This paper proposes an Extensible Document-Based Model Web Engineering approach. First, we propose a document model for multilingual composite documents. Building on work by librarians that lead to the Functional Requirements for Bibliographic Records (FRBR), our document model gathers all versions of a document in a tree structure. It describes the relations between a digital document's various versions, translations, formats, etc. Then we propose a model for document content. Document data and metadata are modeled with relations between entity nodes and data, bringing data and metadata to the same level in the design process. A cascading inheritance model is proposed to allow component reuse and rapid prototyping of applications. Finally, a relationship model enables the creation of application specific relations between documents.**

Sydonie, a free software framework, implements the proposed models. Implementation details are provided and illustrated using two applications built with the framework. A discussion offers a perspective on other approaches and a qualitative evaluation is provided. Finally, this paper concludes with perspectives and further work being carried out using our model and framework.

I. Introduction

Web sites or applications are designed with various methods. Model Driven Web Engineering (MDWE) provides designers with tools to concentrate on the application's Content Model and Navigation Model. Practitioners in the Web industry often use Content Management Systems (CMS) to create web applications. CMS provide ready to use tools, but usually imply using a predefined model, which does not necessarily reflect the needs of an application. These approaches provide efficient ways to develop web sites. However, these models and tools focus on Content rather than Documents.

Web pages are very different from what they first used to be. In the early years of the web, pages were used to represent one document. Then pages were enriched with more content as web technologies evolved. Documents themselves evolved. They may be available in several versions, i.e. different languages, formats, sizes, etc. Documents may include components that are documents themselves, creating composite documents.

Documents metadata also have changed since the beginning of the web, when `META` tags used to be the only way to specify metadata. Nowadays, metadata is embedded in web pages with RDFa [1] or Microdata [2], in media files with XMP [3], only to mention a few examples. However, existing models and tools for Web development focus on document content, but not on document metadata and their management. On the other hand, tools that do manage metadata are professional applications that separate completely document content and metadata.

This paper addresses three main issues with documents in web applications:

- multimodality: a same "document" may be available in different versions (translations, formats, sizes, etc.)
- redundancy: avoid duplication of information when documents are available in different versions. For example if a document is available in HTML and PDF, most information and metadata is the same for the two versions, such as title, author, document language for example.
- composition: when a document includes an image for example, the image is another document in itself, with its own content and metadata. The image can not be considered as a mere file associated with the document for rendering purpose.

To tackle these issues, we propose an Extensible Document-Based Model, putting documents and their metadata at the center of the process. Based on previous Work by librarians, our model considers a document as being the set of various abstracts, translations, formats, etc. of a same intellectual Work. Using a tree structure, it manages multilingual composite documents, their relationships and their metadata. Document data and metadata are defined with RDF-like relations. An inheritance model enables the use of predefined document classes in an application. Statements between documents define their relationships and enable creation of composite documents without redundancies. The proposed model is implemented in an free software web development framework called *Sydonie*[1].

The remainder of this paper is structured as follows. The next Section gives an overview of related work in the MDWE field and CMS industry. Section III presents our document model, how document data and metadata is captured, introduces our inheritance model and relationship model between documents. Section IV explains how the proposed models are implemented in our framework and shows examples of applications developed with Sydonie. Finally, Section V summarizes the contributions of this paper and gives an overview of future work.

II. Related Work

Building web applications has become a complex task. Designers have to manage both client side and server side programing, user interaction and so on. Over the years, Web Engineering has become a discipline (Murugesan et al. [4]),

[1] http://sydonie.net/

and most applications implement design models and concepts (Gellersen and Gaedke, [5]). To address the complex tasks of designing web applications, MDWE approaches aim to model an application's concepts. The design is then mapped into an implementation to deliver the application, using a top-down process. On the other hand, CMS are commonly used in the web development community to create web applications. They provide functionalities to easily create and publish content. Predefined document types and ready-to-use modules allow for customization of the web site or application. CMS usually focus on content creation and publication, and allow rapid development of a prototype. However, they sometimes fail to deliver an ad-hoc document model for specific applications (Hinton, [6]).

A. Model-Driven Web Engineering

Software researchers and developers in the Model-Driven Development (MDD) community have created abstraction models and tools to support the creation of complex applications. Applying these concepts to web development, the MDWE community has come up with abstraction models that better suit the particular field of Web Engineering (Rossi et al. [7]). These top-down approaches model the application in its whole, describing the Content Model (or Domain or Structural), the Presentation Model and the Navigation Model. The Content Model defines the data content of the application, in terms of entities and relationships to other entities. Most models are based on the Unified Modeling Language (UML). Models such as Web Modeling Language [8] (WebML) use XML to create a high-level description of the different aspects of a website. Following these works, some metamodels have been proposed to bridge the various implementations of previous modeling approaches. For example, Koch and Kraus [9] or Schauerhuber et al. [10] propose such metamodels.

An ongoing issue with these approaches is the fact that the design process needs to be complete *before* any prototype can be created. Web Usability is therefore not included in the process, as mentioned by Koch et al. [11]. New approaches are emerging to take Web Usability and agile development [12] methods into account, as proposed by Rivero et al. [13], [14].

However, the above approaches focus on modeling the application, working on both the content (i.e. the documents to manage), and the interaction model (i.e. navigation, etc.). Since content is modeled with UML-based methods, it is therefore modeled with a list of properties and managed as an object within the implementation. Relationships to other content entities are managed by the Content Model, but these methodologies and models do not consider collections of documents formed by the same intellectual content in various forms.

B. Content Management Systems

CMS have become very popular in the web design community. Providing ready-to-use content management, web designers can then concentrate on graphics and interactions. Content can be customized to the application's need. For example, with the Drupal CMS [15], the CCK module [16] allows web site administrators to customize the content of entities by associating field names to a type of content. The administrator builds new content types on top of the core system by adding new fields. The system creates new tables in the database to reflect the changes and store content. The approach is therefore opposite to MDD ones. When using a CMS, the developer needs to adapt the application model to the tool used.

Moreover, these systems are well designed to manage content, but are not made to manage *documents*. The system manages content components but does not consider the set of components as a document entity. With a CMS, the approach is to define a document as the rendition of some content [17], where a document is considered similar to the web page displayed to a user. Most CMS do not manage documents metadata other than in a basic form. For example, if an image illustrates an article, the CMS considers the image as a file related to the article, whereas it should be considered as another document with its own data and metadata.

We need a hybrid approach, where documents on the web are not treated as mere content. A document is an information container where the information is both content and metadata. The document is an instance of some intellectual Work where the content itself can be either text, image, sound, video, etc. It can also be a composition of these elements. In this context, metadata is essential to Document Management Systems. The next Section details our approach and models addressing these issues.

III. Proposed Model

Web applications are usually aimed at managing and rendering content and files. Most systems are designed to only manage the HTML version of a document. Thus, when dealing with multiple versions of a document, duplication of content and data redundancy occur at various levels. Composition is often achieved by linking a content with media files in a web page. Our approach has been to study how other information professionals manage documents. From then, we used concepts and approaches that work in other domains and applied them to document management for the web and web development. Amongst information professionals, librarians have to manage large collections of documents, including many variants of a same intellectual work. For example, a book may be available in several languages, different editions, various material forms (printed, braille, audio...). On the other hand, library systems manage documents metadata but not their content.

This Section first introduces the work conducted by librarians and the lessons the web development community may reuse from that work. Then, we present our proposal for an extensible document based model.

A. Functional Requirements for Bibliographic Records

In order to base cataloguing rules on the intellectual Work instead of the physical items on a shelf, the International Federation of Library Associations and Institutions (IFLA[2]) has published the Functional Requirements for Bibliographic Records (FRBR) [18]. FRBR defines a model for bibliographic records based on entity relationships. This model has also been adopted by the International Committee for Documentation

[2]IFLA, http://www.ifla.org

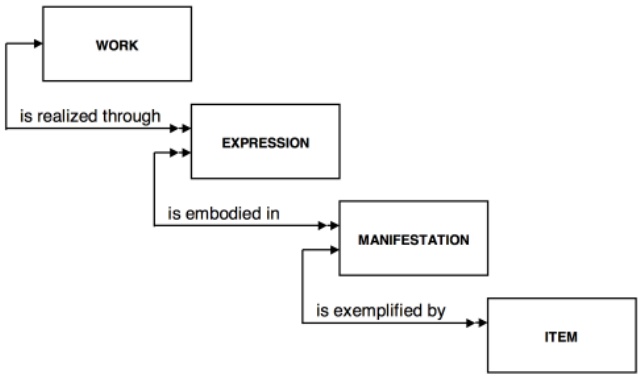

Fig. 1. Group 1 entities (from FRBR report)

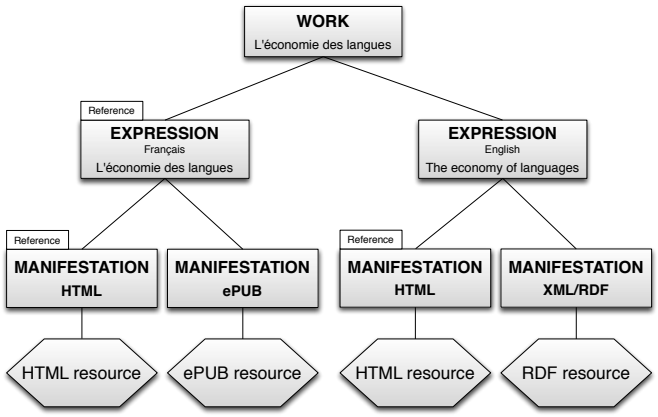

Fig. 2. Tree representing a document

(CIDOC) of the International Council of Museums (ICOM) with an Object Oriented (OO) model.

FRBR defines entities that represent the key objects of a document. These entities are organized in three groups. The first group defines *Work*, *Expression*, *Manifestation* and *Item* and represents the *model* of a document. The second group defines the entities *Person* and *Corporate Body*, which represent entities responsible for the editing process of a document (creation, production, publication, etc.). Finally, the third group defines entities that are the subjects of a document (*Concept*, *Object*, *Event*, and *Place*).

We focus on the first group of entities, since it captures most of the issues addressed in this paper. The first group's four hierarchical entity levels *Work*, *Expression*, *Manifestation* and *Item* can be illustrated as shown in Figure 1 and are defined in the FRBR report as follows:

> "The entities defined as *Work* (a distinct intellectual or artistic creation) and *Expression* (the intellectual or artistic realization of a Work) reflect intellectual or artistic content. The entities defined as *Manifestation* (the physical embodiment of an Expression of a Work) and *item* (a single example of a Manifestation), on the other hand, reflect physical form."

A tree structure represents any intellectual production. Multilingual support is achieved through the Expression entities, and different formats through Manifestation entities.

FRBR also defines relationships between entities to express relations between them and are used "as the means of assisting the user to *navigate* the universe that is represented". The *has part* or *is a part of* Work-to-Work relationship can be used to store the relationships between a *master* document and its components, therefore keeping that information at the highest level of the document tree. The *has a translation* or *is a translation* Expression-to-Expression relationship can obviously be used to store the fact that an Expression is a translation of another one.

We believe that the FRBR model reaches beyond documentation and addresses the problems of redundancy, multimodality and composition. Its principles may also guide us. We use them as a metaphor to apply to documents on the web, as the next Section will show.

B. Document Model

We propose an approach to manage documents as a whole, including content, composition and metadata, using the FRBR as a metaphor [19]. As suggested by Buckland [20], the functional view of what constitutes a document is defined in our document model.

Using the guidelines for group 1 entities from the FRBR report, we define a document model with the Work, Expression and Manifestation entity levels and a pointer to the resource itself. They represent intellectual or physical aspects of a document:

- the intellectual Work is represented through the Work entity. It contains information relevant to all the versions of the document, such as the author of the original Work, when it was first published, etc.
- Expression entities can represent a translation, an abstract or other versions of the same Work entity. Expressions contain metadata information describing the variant of the Work. For example the translator's name, the language used, etc.
- Manifestation entities represent the different formats, sizes, codecs, etc. that materialize the given Expression. It may contain metadata such as resolution for an image or bit rate for a mp3. Each Manifestation includes a pointer to a content resource containing the specific version.

For a same intellectual Work, these different entities are represented in a tree structure, as shown in Figure 2. Instead of considering each version as a separate document, as most systems do, our model considers a document as the complete tree. This model allows for language negotiation and content negotiation when a document is requested. It also makes it easier to create a navigational map to view other Manifestations of a same Work, thus providing a user with links to other available versions.

Like FRBR, our model groups versions, translations, formats, etc. of a document in a single tree structure using

Work, Expression and Manifestation entities. The Work entity represents the intellectual creation, therefore it only contains metadata. For example it contains the first publication date, information about the author, and additional metadata depending on the type of document.

Expression entities represent the various realizations of a Work. As defined in FRBR, expressions are therefore intellectual entities containing metadata. A "reference Expression" refers to the original Expression (i.e. the French Expression in Figure 2). In the case of a translation for example, Expression entities may contain the title of the translation of the Work, its language, information about the translator, plus specific information. Expression-to-Expression relationships express how a given Expression is derived from the reference Expression, with relationships such as *is a translation of* or *is an abridged version of* for example.

A Manifestation entity is an embodiment of an Expression of a Work. Manifestation entities can refer to various formats (such as HTML or PDF), image sizes or media encoding. A "reference Manifestation" refers to the occurrence that served for the creation of the other ones (i.e. the HTML Manifestation in Figure 2). For image documents, the reference Manifestation would be the original image that was used to create smaller ones such as thumbnails for example. Manifestation entities can contain information such as content type and file size for example. But they mostly carry the content of the document, using a pointer to a content resource.

Document views are therefore computed on a specific *branch* of the document tree, using metadata information from the Work entity to the Manifestation entity. When a user requests a document (considered here as the whole tree), the branch to be used for creating the view depends on which type of entity is requested. If the request occurs at the Manifestation level, for example requesting the PDF Spanish Expression of a document, then the branch used is that of the requested Manifestation. If the entity level requested is Work or Expression, negotiation will occur. As outlined in *Cool URIs for the Semantic Web* [21], a W3C note published in December 2008, HTTP Language and Content Negotiation can be used to serve the most suitable corresponding content to the client's preferences. Keeping in mind that a client will always be served a view on a branch, providing an access to a document through the Work or Expression entity level follows the algorithm:

1) at the Work level, Language-Negotiation is used to know which Expression of the document to use.
2) at the Expression level, Content Negotiation is used to decide which Manifestation to serve. A typical use case would be a web browser accessing a resource and being served HTML content whereas a robot would be served an XML or RDF content.
3) at the Manifestation level, the system can directly serve the content.

Once the Manifestation to use is chosen, the system uses the data and metadata from each entity level to build the rendered view. Since a document is a tree with data and metadata attached to the nodes, it is *self-aware* of the various translations and formats available. It is then easier to present the user with alternative choices such as other languages or formats available.

Using this approach, multimodal documents are modeled in a unified container. Each information about the document is described at the appropriate entity level. This avoids redundancies and insures data consistency throughout the various versions of the document, i.e. in the whole document tree. Using FRBR-like relations, document composition is addressed with the model presented in Section III-E.

In order to manage different types of documents using this document model, we need a way to specify what information a document will carry and model that information. The next Section presents how we represent document data and metadata in our model.

C. Data and Metadata Model

In the document tree, each entity node carries the information it is associated with. The specified information may be metadata or content data, depending on the application model. Since Work and Expression entities represent intellectual information, the information they carry is metadata about the document (at the Work level) or about a specific version (at the Expression level). Having a model that contains data, metadata and a pointer to the content resource avoids information redundancies.

However, the proposed model must be able to apply to any type of document, keeping the entity nodes as generic as possible. We need a way to model the various types of information each entity node may carry. To provide a generic way to manage the information attached to a node, our approach uses a RDF-like model. Each entity node has a set of predicates. The arc for each predicate points to an object modeling its value. Similarly to RDF, data is then represented as triples (subject, predicate, object) where:

- subject is an instance of a document entity node, i.e. a Work, Expression or Manifestation node.
- predicate is the name of the relation, or, in terms of OO concepts, the name of the attribute.
- object is the value associated to the predicate. In the implementation detailed later in this article, it is an object (in the OO sense). Its class models the information it represents.

The modeled data attached to an entity node may be of any kind. It may be scalar data such as text, or complex data such as a postal address for example. Figure 3 illustrates this model in the case of an article, using the document tree from Figure 2 and showing a detailed view of a branch.

In order to use this model in various applications, we need to provide ways to define classes of documents managed by an application. A class of documents is the definition of the type of data each entity node may contain. To define such a class, one specifies, for each entity level, the list of accepted predicates and the type of information each predicate points to. Using this model, any kind of document may be defined.

In order to do so, our model defines the notion of attribute. An attribute is a predicate and its associated object, called

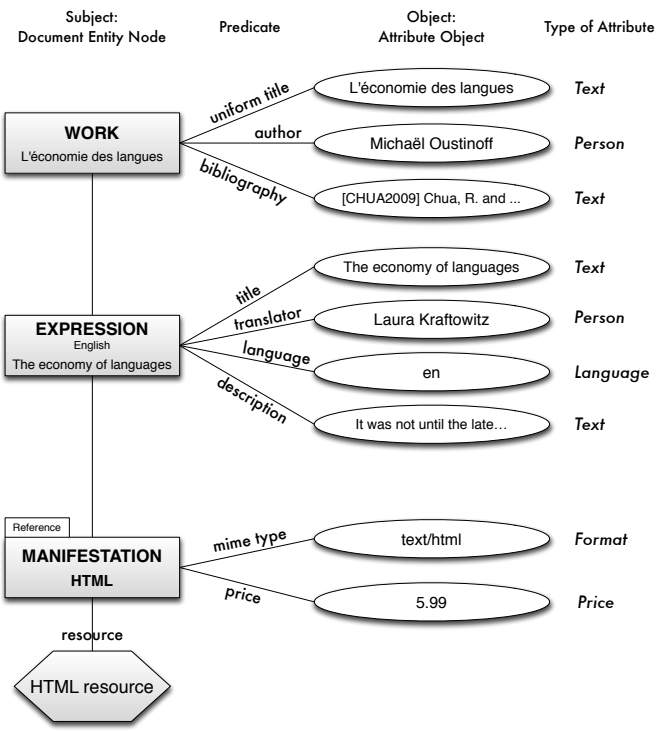

Fig. 3. Detailed model of a branch of a document with its predicates and object values, based on the document tree in Figure 2

```
<configuration>
   <class>Article</class> <extends>SydonieDocument</extends>

<!-- attributes at the Work level -->
   <attribute entityLevel="Work"
              minOccur="1" maxOccur="1" >
      <predicate>uniformTitle</predicate>
      <objectClass>Text</objectClass>
   </attribute>
   <attribute entityLevel="Work"
              minOccur="1" maxOccur="1" >
      <predicate>bibliography</predicate>
      <objectClass>Text</objectClass>
   </attribute>
   <!-- other attributes omitted for clarity -->

<!-- attributes at the Expression level -->
   <attribute entityLevel="Expression"
              minOccur="1" maxOccur="1" >
      <predicate>title</predicate>
      <objectClass>Text</objectClass>
   </attribute>
   <attribute entityLevel="Expression"
              minOccur="1" maxOccur="1" >
      <predicate>language</predicate>
      <objectClass>Language</objectClass>
   </attribute>
   <!-- other attributes omitted for clarity -->

<!-- attributes at the Manifestation level -->
   <attribute entityLevel="Manifestation"
              minOccur="1" maxOccur="1" >
      <predicate>price</predicate>
      <objectClass>Price</objectClass>
   </attribute>
   <!-- other attributes omitted for clarity -->

</configuration>
```

Fig. 4. Example of configuration file for an `Article` class of documents

attribute type. A class of documents is thus defined by specifying the list of accepted attributes for each node. An XML formalism based on our model enables the definition of a class of document. Using the example illustrated in Figure 3 for a scientific article and its translations, the class of documents for such articles can be represented using the XML code shown in Figure 4, in some ways similar to WebML's XML representation [8].

The flexibility of the approach resides in the fact that entity nodes are generic objects used for all types of documents. Attribute type objects may be defined as needed by application models. Therefore, any type of document may be created with this approach.

Since information is attached to the highest possible entity level (Work, Expression or Manifestation), information redundancy is avoided. For example, information specified at the Work level is used by all its Expressions. In the above example of a scientific article, when creating a translation, information about the author and the bibliography are already present at the Work level and therefore are automatically used by the new Expression. Similarly, when adding a PDF version, i.e. creating a new Manifestation, data and metadata at the Work and Expression levels are already present for that article and are reused. This process ensures that all versions of a document carry the same information and avoids redundancies between versions. Data and metadata consistency of a document and its variant forms are therefore improved.

Using this model, however, a document model first needs to be specified. In order to avoid the top-down drawbacks of MDWE and allow quick prototyping using agile development principles, some flexibility is added to our approach, as described in the next Section.

D. Inheritance Model

When building web applications, designers need to be able to create the document model that meets their needs, instead of using pre-defined models they can not alter. But, on the other hand, using pre-defined models may save a lot of time in a design and development process . We need a model that allows the use of an existing document type, but that also allows its alteration. To achieve this, we propose a *cascading inheritance* model that offers two ways to customize existing document types, combining classic object inheritance with alteration.

Classic object inheritance allows the creation of new classes of documents. It is slightly modified to accommodate our entity based document model: derived classes inherit the properties of their parent class, but the inheritance is applied to each entity level. Using a Document base class as the parent of all types of documents, an Article class of documents may be defined using inheritance. Figure 5 illustrates the Document and Article classes, and the resulting document model for articles.

This solution works well to create a new class deriving from an existing one. However, if using a ready-to-use Article document class, one could not alter its model. It would go against separation between predefined model and application model. Predefined models must stay generic models. In order to be able to use already made components and customize documents, the model for an existing class of documents must be flexible. When customizing existing document classes to

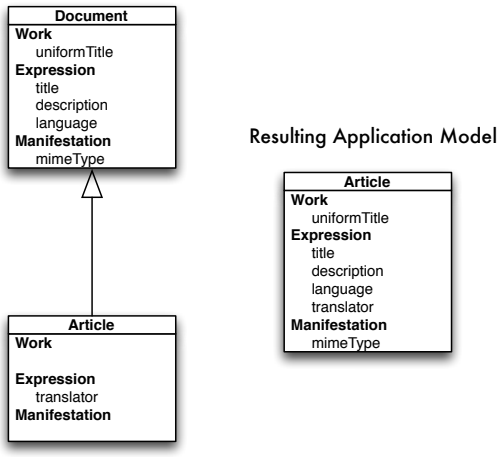

Fig. 5. Resulting document model with classic inheritance

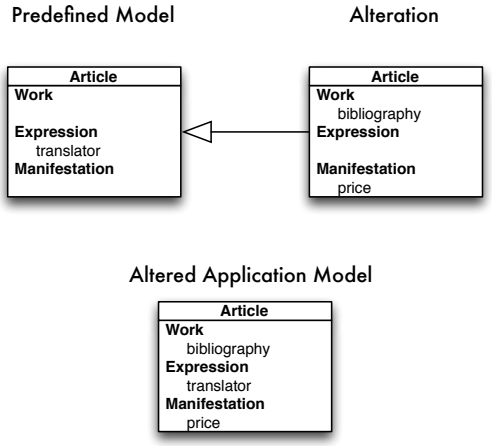

Fig. 6. Alteration of an predefined Article document class

create a custom design, the model uses the existing document class definition and adds the custom attributes defined for the application. This process is shown in Figure 6.

Let us now consider the previous example of a scientific article as illustrated above and in Figure 3. Existing document classes include the Article class and its parent class Document. The application itself may alter the Document class and the Article class using the principle described earlier. When creating the application to publish such scientific articles, a designer may want to specify:

1) documents in the application use the default Document class definition.
2) Document definition is altered: all documents in the application have an author attribute at the Work level.
3) Scientific articles belong to the Article class of documents. The default Article class derives from the Document class, adding a translator attribute at the Expression level.
4) Default Article model is enriched with a bibliography attribute at the Work level and a price attribute at the Manifestation level.

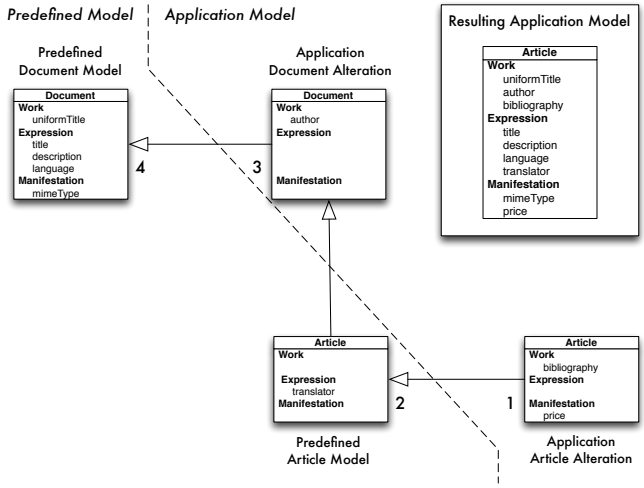

Fig. 7. Cascading inheritance model: combining inheritance and alteration

Our inheritance model combines classic inheritance and alteration to provide a *cascading* inheritance model. Using this cascading model, designers may use existing classes of documents and adapt them to the needs of the application. Figure 7 illustrates the computing process. The list of attributes at each entity level is computed from most specific to least specific. In case of conflict, the application definition takes precedence over default models.

This cascading inheritance can be applied, not only to classes of documents, but also to attribute object models. Using this process, a designer can customize both a document class and the type of data they contain.

The model presented so far answers the redundancy and multimodality issues raised in the Introduction of this paper. The next Section introduces how our relationship model tackles the composition issue.

E. Document Relationships Model

Relationships between documents serve several purposes. They may represent logical relationships or structural relationships. Logical relationships reflect the model or business logic of the application. In a blog for example, comments are related to a post entry, establishing a *is a comment of* relationship between the comment and the post. Structural relationships are used to describe how documents are intertwined. For example, structural relationships may express the fact that an image document is a part of an article document.

The FRBR report defines relationships between entities. FRBR relationships may be Work-to-Work, Expression-to-Expression, etc. They express various properties, for example a Work-to-Work *is an adaptation of* may be used between a book and its movie counterpart. FRBR does not define strict rules but mostly high level principles and guidelines. In the web and computer science sphere, relationships between resources are usually expressed with RDF triples of the form (subject, predicate, object), which allow to express almost every kind of relationship.

The similar relationship principles of FRBR and RDF offer a high level of expressivity and flexibility. Our model

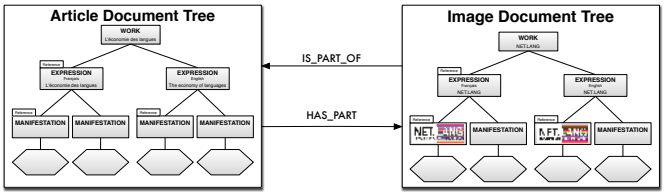

Fig. 8. Whole/part Statement between documents

is based on these principles. We define relationships between documents using Statements that are triples of the form (documentA, predicate, documentB). Each defined predicate is combined with its reciprocal predicate. In the example of the blog post and a comment, two reciprocal Statements express their relationships: (blogpost, has_comment, comment) and (comment, is_comment_of, blogpost). In the case of structural relationships, Statements between documents are used for the recomposition of the "main" document to include all of its components. Based on the branch being used for the main document, negotiation will occur to determine which branch of each component to use. Figure 8 illustrate whole/part Statements between documents. Even though most Statements are between documents (i.e. the whole tree), Statements between entities may also be defined. For example, Statements may be used between two Expressions to express the fact that one Expression is a revision of the other one.

In order to benefit from pre-defined models, default Statements may be defined in the model's implementation. However, application specific Statements must be able to be defined. Each application may define its statement policy and vocabulary. This approach brings high level of flexibility and adaptability to any application needs.

F. Summary

In this section, we proposed several models:

- a document model based on a tree structure, using Work, Expression and Manifestation entities. It groups all versions of a same intellectual work and models both documents data and metadata. The proposed document model avoids data and metadata redundancies, improves information consistency throughout the various versions of a document. It also models multimodal documents and provides a negotiation strategy to serve the most suitable information to a user.
- a model for document data and metadata based on RDF-like relations. Defining a class of documents is achieved by specifying the information each entity level may carry. Data and metadata consistency between all variants of a document are therefore improved.
- a cascading inheritance model to allow component reuse and alteration of existing document models.
- a Statement model to express relationship between documents. The Statement model brings a high level of flexibility and expressivity to design application models. Statements are used for the composition of multimodal composite documents.

The proposed approach answers the main issues mentioned at the beginning of this paper: data and metadata redundancy and consistency, multimodality and document composition These models are implemented in Sydonie, an free software web development framework. The next Section presents some implementation details and examples of real world applications built with our model.

IV. Implementation and Examples

Sydonie[3], *SYstème de gestion de DOcuments Numériques pour l'Internet et l'Édition*, is a web development framework implementing the proposed models. The name literally means Document Management System for Publishing on the Web. Sydonie is developed within Normandie Université and the CNRS. C&F éditions[4] is a publishing partner developing online services based on Sydonie. Sydonie is free software made available under a GPL license. Implemented in PHP and relying on a MySQL database, Sydonie can run on any basic LAMP server.

Several applications have been built with our framework. We will illustrate the implementation details with two of them:

- In 2012, C&F éditions published Net.Lang, Towards the Multilingual Cyberspace [22], a book about language diversity on the Web. Two versions of the book (French and English) were simultaneously released. A companion website[5] presents the articles in both languages. More translations may be published when articles are translated in other languages. This application illustrates how the proposed model simplifies the management of multilingual documents.
- The *Institut de l'Histoire et de la Mémoire des Catastrophes* (IHMEC), or Institue for Disasters History and Memory, created an application[6] to collect and publish witness stories about natural or industrial disasters. Stories are moderated before publication. Working groups allow specific projects to be carried out, with schools for example, where moderation is delegated to the group coordinator. This example illustrates the use of application specific document classes. It also shows how the relationship model is used to implement the application's workflow and stories management.

This Section is organized as follows. First, we present how the document model is implemented. Section IV-B explains how framework and applications articulate to form the base for our cascading inheritance model. Implementation of the data model is explained next. Section IV-D presents the implementation and management of Statements. Finally, the benefits of the approach and its implementation are discussed.

A. Document Model Implementation

The framework implements the document model with a Document abstract class. It defines the tree structure of documents, the articulation between a document and its entity

[3]http://sydonie.net/
[4]http://cfeditions.com
[5]http://net-lang.net
[6]http://memoiredescatastrophes.org

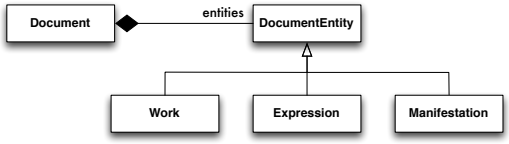

Fig. 9. Document abstract class and Document entity classes

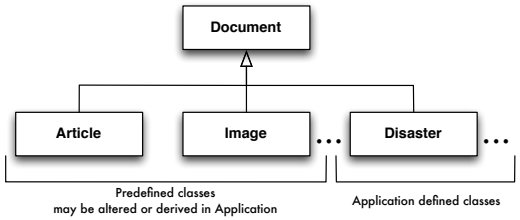

Fig. 10. Predefined document classes and Application classes

nodes (Work, Expression or Manifestation nodes). It also manages each nodes' attributes and their data. Entity nodes are implemented as deriving from a DocumentEntity class in order to manage their specificities. The document class is a composition (in the UML sense) of entity nodes. Figure 9 shows the relation between Document and DocumentEntity classes.

Document classes such as Article, Image, etc. derive from the base Document class. Applications may use existing classes of documents, with or without alteration. New classes of documents may be created depending on the application model. Figure 10 illustrates the inheritance between the Document abstract class and generic or specific classes of documents. Our inheritance model allows alteration and/or derivation of predefined classes. The next Section presents how framework and applications intertwine to implement the model.

B. Classes of documents, Framework and Applications

The framework's engine makes the low-level tasks totally transparent for the web designer, such as persistance management, session management, etc. Built on top of the engine, the framework's document layer provides predefined document models. Sydonie core's document layer provides basic classes of documents such as Article, Image, BlogPost or News for example. These classes of documents are illustrated by the left side of Figure 7 and provide base classes for an application. Application document models are built on top of the predefined document models. These models are represented on the right side of Figure 7. The cascading inheritance model presented in Section III-D is applied to build the resulting application model. As shown in Figure 4, classes of documents are specified using XML. The same syntax is used to specify framework predefined classes of documents or application classes of documents. The cascading inheritance model is applied to compute the different XML configuration files and give a resulting class of documents with specific data and metadata.

The Net.lang application manages scientific articles. Since the application does not need other types of articles, Net.Lang uses the framework's predefined Document and Article classes. Document and Article classes are altered at the application level, adding information such as author, bibliography and price, exactly how Figure 7 illustrates. To present short presentations of authors, an AuthorBio class was created at the application level, directly inheriting from the framework's predefined Document abstract class. Since the document model manages translations of documents, there is no need to specify in the application model how documents and their translations are related.

In the case of the IHMEC application, the Disaster class of documents is defined to model information about disasters. It derives directly from the framework's predefined Document abstract class, which models basic documents. It builds on top of Document, adding disaster related specific information such as begin and end dates, event type, place, associated links and references.

These two examples show how the Document model and the inheritance model enable the reuse of existing document models, with or without alteration, but also the creation of new classes of documents, as illustrated in Figure 10. Whether documents will have translations or not is transparent to the application's document design process since it is inherent to our document model.

C. Data and Metadata Model Implementation

Using our document model, a class of documents is the definition of what information each entity level may contain. The previous Section showed how to specify the set of information each entity node may carry. The kind of information associated to each node may vary depending on the application model. As explained in Section III-C, the information is modeled using Attributes in relation with an entity node. Attributes are composed of the predicate describing the relation and the object modeling the associated information. The framework implements this model with an Attribute class to bind an entity node with its object value. Object values are modeled with an AttributeType abstract class, derived to provide specific AttributeType models. These children classes can model scalar data, such as text or integer, or more complex structures, such as a price or a person as shown in Figure 3 for example.

Similarly to documents, the framework implements some predefined attribute type models. At the application level, new models can be created for attribute type objects using classic inheritance, but existing models can also be altered using our cascading inheritance model. The internal structure of a document is therefore transparent to the developer.

In the Net.lang application, a specific AttributeType class *Person* was created to model authors and translators. It provides the name of the person and a relation with the person's short biography. Note that the biography in itself is a class of documents, as introduced in the previous Section, and that it uses Sydonie's document model in order to be available in several languages and apply a language negotiation process if necessary.

In the disaster stories project, several specific AttributeTypes were created. A Reference class is used to specify if the disaster's description was written with external sources such as Wikipedia for example. For witness stories, an attribute

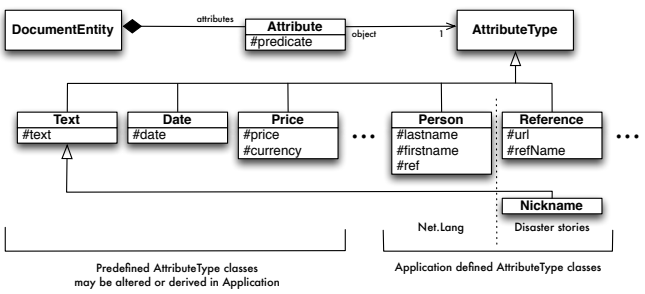

Fig. 11. Relations between document entity and attribute types

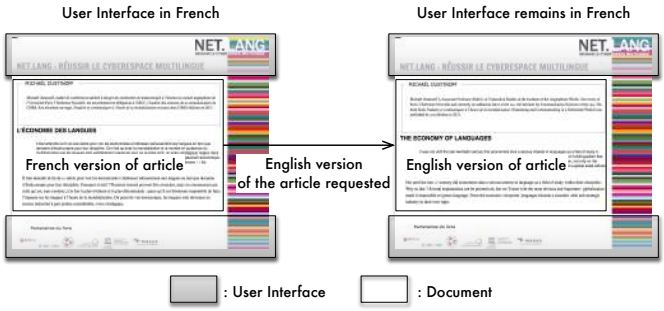

Fig. 12. Distinction between Document and Application Interface languages

type *Nickname* was created for people to choose the name under which their story will appear. It simply derives from the Text AttributeType class, but has a specific checker to avoid duplicate nicknames. Figure 11 shows relations between the DocumentEntity class (representing entity nodes), Attribute (modeling attributes) and AttributeType class and its children classes, and illustrates the above examples.

Creating specific displays or input for AttributeType objects does not necessarily require the creation of new AttributeType classes. Within the framework, the inheritance model introduced in Section III-D applies to templates and GUI bindings. Using the same cascading principle, they may be redefined in the application. This process applies to classes of documents as well as to AttributeType objects. It allows default templates to be defined at the framework level, and refined as needed in the application.

D. Statements

As explained in Section III-E, relationships between documents are managed using a RDF like model of statements. A Statement class implements this approach. Available predicates are defined at the framework level or at the application level.

At the framework level, default Statement predicates are used to express general relationships between documents or entities. For example, the whole/part relationship shown in Figure 8 is defined in the framework and may be used in any application. Application relationship policies are defined at the application level. It specifies the acceptable predicates between documents or entities. A predicate is always defined with its counterpart so that the relationship can easily be expressed from any end of the arc. In order to simply deal with Statements, a manager is provided by the framework. It consists of a singleton who registers available predicates for the application. It provides methods to easily add or remove statements. Given a document and a predicate, it can fetch objects having a relationship with the given document using the given predicate. This allows easy retrieval of related objects in an application.

Net.Lang defines predicates between Article documents and AuthorBio documents. It is used to link an Article with a short presentation of its author (or translator). The disaster stories application defines predicates to express the fact that a witness story is about a given disaster for example. Other statements are used to manage working groups relations to stories or to the group's description.

Statements offer a flexible way to manage relations. In contrast with CMS where relations between documents are defined in a static way, Sydonie lets the designer define its application policy. Using the application vocabulary, Statements reflect its document logic and are easily managed with the tools provided by the framework.

E. Discussion

The proposed document model, its associated data and metadata model, inheritance model and relationship model have been successfully implemented in Sydonie. Several applications are built using the framework, as illustrated in the previous Sections.

The benefits of the approach are substantial in both examples. The Net.Lang application provides articles in two languages. Using Sydonie's document model, adding a translation for an article does not require extra development. Relations between translations are managed at the framework level and therefore transparent to the designer who can then focus his work on the application and its interaction logic. Moreover, the framework makes a distinction between the language of the application's interface (menus, buttons, etc.) and a document's language. For example, if a user visits the site choosing the French interface, he will view the French version of the articles. But if he wishes to read the original version of an English article, the application's interface will remain in French unless otherwise requested by the user, as illustrated in Figure 12.

Building such an application with a CMS would require specific work in order to manage the Net.Lang's multilingual aspects. Using the Drupal or Wordpress [23] CMS, translating an article turns out to create a new article and then adding a relation to specify the fact that one is a translation of the other. With Wordpress, an plugin easily enables this. However, information that is common to both versions (i.e. attached to the Work entity level in our model) is duplicated. To avoid duplication of the bibliography for the translations of an article, it would need to be externalized from the article and linked to all the article's translations. Using our model, such data redundancies are automatically avoided if the document model places the information at the right entity level.

Moreover, these CMS do not differentiate document language from user interface language. This implies that when requesting a translation of an article also changes the application's interface language. In terms of navigation and

TABLE I. EVALUATION SUMMARY

Model	Evaluation (out of 8 testers)	Comments by evaluators
Document	positive: 50% neutral: 25% no answer: 25%	- avoids starting from an empty project - easy to set up documents
Data and metadata	positive: 37.5% neutral: 50% negative: 12.5%	- useful to start but not so easy to adapt to our application - concept seems very interesting but it is hard to apprehend - seems good but didn't have a chance to test thoroughly
Inheritance	positive: 100%	- really practical to overload components - allowed me to build on existing objects - easily create new types of documents
Document relationships	positive: 75% neutral: 12.5% no answer: 12.5%	- easy to understand and to use - makes the job easy! - useful but its syntax is not so easy (need more documentation)

user interaction, it may be frustrating for users to have their interface constantly switching between languages. One could compare this to his favorite word processing software's menus and buttons turning Italian, Spanish or other to reflect the edited document's language.

WebML's approach to define document model and content is similar to our model's approach. To design an application similar to Net.Lang using MDWE tools such as WebML, a *translation* relationship would need to be defined in the application model. The interaction model would also need to be defined in order to have links to other versions of an article form example. In both cases, the existing tools may well do the work needed to create a multilingual application. Using our model, however, does not require any specific modeling or any extra development. Having a truly multilingual model brings substantial benefits in terms of development time and interaction design.

An informal study was conducted to evaluate the usability of our approach and implementation with web developers. Eight master's degree students were enrolled in a two day Sydonie workshop. Participating students' experience in web development ranged from one to three years, making the audience an intermediate to advanced level. First, the models and how to use them were presented to the participants, then they tried to build a "mini blog" application, with the presence of an instructor. In a couple of hours, they all succeeded to build a basic application adding blog posts and displaying the latest posts on the home page. For each of the models proposed in Section III, participants were asked whether they found the model and its use helpful in their development, and the possibility to add comments was left open. Table I summarizes the answers and comments. Only the data and metadata model implementation, using the AttributeType classes, seemed confusing and/or too restrictive to use. This feeling of too much constraint may be due to the lack of predefined attributes in the framework. Overall, however, all participants found the models easy to use and found them practical for development.

V. FUTURE WORK AND CONCLUSION

In this paper, we propose a model for documents in web applications. First, we propose a document model for multilingual composite documents. Then we propose a model for document data and metadata, using RDF like relations. A cascading inheritance model is proposed to allow component reuse and rapid prototyping of applications. Finally, a document relationship model similar to RDF statements is proposed to reflect an application model. These models are implemented in our framework and are being used in several applications.

Other applications are currently being developed with the framework and will extend Sydonie's metadata management possibilities. Craham[7] is a historical and archaeological research unit within Normandie Université. It manages a collection of photographs of archaeological sites, and an application was built with our framework to manage digitized versions of the slides[8]. The Craham application reuses the framework's implementation of the image model. The image class defined in the application benefits from all the interactions already defined in the framework for images, such as the creation of thumbnails, the inclusion of metadata in images using XMP [3]. This work bridges our document model with metadata vocabularies. A metadata management model using our document model and inheritance model is being tested using the Craham application. Using our models, metadata mapping between Sydonie documents and their resources (i.e. files) is made transparent. This project aims for Sydonie to provide ready-to-use routines for web developers to use available metadata in their applications.

Future work also include a document exchange model. Since each community uses different standards depending on their needs (TEI [24], EAD [25], PRISM [26], etc.), the applications using our model would specify which standard to use. We find that it is important for our model and its framework implementation not to choose a standard over an other.

As we have seen in Section II-A, significant work has been done on modeling Web applications. Combining our model with MDWE application models would allow the design of the document model as well as the navigation and presentation models. Our models being already implemented, the framework would need to integrate existing MDWE models.

The definition of statements used by an application is left to the developer. A formalism, such as OWL for example, needs to be introduced in order to allow statements reuse and exchange between applications.

[7] http://www.unicaen.fr/crahm/

[8] https://craham.greyc.fr (under development)

Finally, providing users of the framework with a documentation and a public forge is under way. Providing ready to use application components and allowing developers to submit their own work will also be a key issue to the success of the open source project.

ACKNOWLEDGMENTS

This research Work is funded by Normandie Université, the Conseil Régional de Basse-Normandie with the CPER program, the European Council with the FEDER program, and the CNRS with the TGE-Adonis project[9].

REFERENCES

[1] M. Sporny and S. McCarron, "Html+rdfa 1.1," W3C, Tech. Rep., Mai 2011. [Online]. Available: http://www.w3.org/TR/rdfa-in-html/

[2] I. Hickson, "Html microdata," W3C, Tech. Rep., Mai 2011. [Online]. Available: http://www.w3.org/TR/microdata/

[3] R. Roszkiewicz, "Xmp primer," IDEAlliance, Tech. Rep., 2008.

[4] S. Murugesan and Y. Deshpande, "Web Engineering: Managing Diversity and Complexity of Web Application Development," *Lecture Notes in Computer Science*, vol. 2016, p. 357, 2001. [Online]. Available: http://www.springer.com/computer/communication+networks/book/978-3-540-42130-6

[5] H.-W. Gellersen and M. Gaedke, "Object-oriented web application development," *IEEE Internet Computing*, vol. 3, no. 1, pp. 60–68, Jan. 1999. [Online]. Available: http://dx.doi.org/10.1109/4236.747323

[6] E. Hinton, "The Twilight of the CMS," 2011, http://labs.talkingpointsmemo.com/2011/07/the-twilight-of-the-cms.php. [Online]. Available: http://labs.talkingpointsmemo.com/2011/07/the-twilight-of-the-cms.php

[7] G. Rossi, O. Pastor, D. Schwabe, and L. Olsina, Eds., *Web Engineering: Modelling and Implementing Web Applications*, ser. Human-Computer Interaction Series. London: Springer London, 2008. [Online]. Available: http://www.springerlink.com/content/j026q4502u787147/

[8] S. Ceri, P. Fraternali, and A. Bongio, "Web Modeling Language (WebML): a modeling language for designing Web sites," *Computer Networks*, vol. 33, no. 1-6, pp. 137–157, Jun. 2000. [Online]. Available: http://dx.doi.org/10.1016/S1389-1286(00)00040-2

[9] N. Koch and A. Kraus, "Towards a common metamodel for the development of web applications," in *Web Engineering*, ser. Lecture Notes in Computer Science, J. Lovelle, B. Rodrguez, J. Gayo, M. del Puerto Paule Ruiz, and L. Aguilar, Eds. Springer Berlin / Heidelberg, 2003, vol. 2722, pp. 419–422, 10.1007/3-540-45068-8_92. [Online]. Available: http://dx.doi.org/10.1007/3-540-45068-8_92

[10] A. Schauerhuber, M. Wimmer, and E. Kapsammer, "Bridging existing Web modeling languages to model-driven engineering," in *ICWE '06 Workshop proceedings of the sixth international conference on Web engineering*. New York, N.Y., USA: ACM Press, Jul. 2006, p. 5. [Online]. Available: http://dl.acm.org/citation.cfm?id=1149999

[11] N. Koch, S. Meliá-Beigbeder, N. Moreno-Vergara, V. Pelechano-Ferragud, F. Sánchez-Figueroa, and J.-M. Vara-Mesa, "Model-Driven Web Engineering," *European Journal for the Informatics Professional*, vol. IX, no. April, pp. 40–45, 2008. Available: http://www.upgrade-cepis.com/issues/2008/2/upg9-2Koch.pdf

[12] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Principles behind the agile manifesto." [Online]. Available: http://agilemanifesto.org/principles.html

[13] J. M. Rivero, J. Grigera, G. Rossi, E. R. Luna, and N. Koch, "Improving Agility in Model-Driven Web Engineering." *CAiSE Forum*, pp. 163–170, 2011. [Online]. Available: http://dblp.uni-trier.de/db/conf/caise/caisefo2011.html\#RiveroGRLK11

[14] J. M. Rivero, G. Rossi, J. Grigera, E. R. Luna, and A. Navarro, "From interface mockups to web application models," in *WISE11 Proceedings of the 12th international conference on Web information system engineering*. Springer, 2011, pp. 257–264. [Online]. Available: http://dl.acm.org/citation.cfm?id=2050963.2050985

[15] "Drupal open source cms," 2012. [Online]. Available: http://drupal.org/

[16] "Drupal cck module," 2012. [Online]. Available: http://drupal.org/project/cck/

[17] R. T. Pédauque, *Le document à la lumière du numérique*. C&F éditions, 2006.

[18] O. Madison, *Functional Requirements for Bibliographic Records*. Mnchen, Germany: K. G. Saur, 1998.

[19] J-M. Lecarpentier, C. Bazin, and H. Le Crosnier, "Multilingual composite document management framework for the internet: an FRBR approach," in *Proceedings of the 10th ACM symposium on Document engineering - DocEng '10*. New York, New York, USA: ACM Press, Sep. 2010, pp. 13–17.

[20] M. Buckland, "What is a 'digital document'?" *Document numérique*, no. 2, pp. 221–230, 1998. [Online]. Available: http://people.ischool.berkeley.edu/~buckland/digdoc.html

[21] L. Sauermann, R. Cyganiak, D. Ayers, and M. Vlkel, "Cool uris for the semantic web," W3C, Tech. Rep., Decembre 2008. [Online]. Available: http://www.w3.org/TR/cooluris/

[22] L. Vannini and H. Le Crosnier, Eds., *Net.lang, Towards the Multilingual Cyberspace*. C&F éditions, 2012.

[23] "Wordpress open source cms," 2010, http://wordpress.org.

[24] TEI, "Text encoding initiative," 2007, http://www.tei-c.org/.

[25] "Encoded archival description," 2012. [Online]. Available: http://www.loc.gov/ead/

[26] IDEAlliance, "Prism: Publishing requirements for industry standard metadata," International Digital Enterprise Alliance, Inc., Tech. Rep., 2009, http://www.prismstandard.org/.

[9] http://www.tge-adonis.fr