Universidad de Valladolid
Departamento de Informática

Dynamic exploitation of relationships between documents in digital libraries: application to legal documents.

Submitted in candidacy for the degree of Doctor of Philosophy by

María Mercedes Martínez González

April 24th, 2001

Contents

List of Algorithms

List of Figures

List of Tables

1

Introduction

Contents

1.1 Motivation and aims

Digital libraries are advanced, complex information systems that complement traditional libraries; they provide more resources and services and make the development of new solutions to users' problems possible. In addition they have two very highly valued advantages in the information society: A cheap support for large quantities of information and the possibility of easy access for all to the information. They provide access to their resources for a wide range of users, who would not otherwise possess the ability to gain access to the said resources. The resources of a digital library do not suffer the greatly feared "wear and tear" that their physical counterparts suffer and it is not necessary therefore to restrict access to the resources in order to preserve them. In addition, access to a digital library does not require the user to physically go to the place where the library is; it is, rather, the library which "moves" to where the user is, offering its resources via interfaces, in some cases so popular as that of the Web navigators.

However, these advantages hide an intriguing complexity: digital libraries are multidisciplinary, the data they store being heterogeneous in nature, format, type, etc., and the applications used as supports are also generally heterogeneous. To speak of digital libraries is to speak of such diverse fields of interest as information searching or filtering, the distribution of resources and/or applications, heterogeneity, manipulation of information and documents, the interaction between man and machine, or the aspects related to security and legal rights of access to the resources offered. Thus they share some of their problems and solutions with that enormous source of information available in Internet, and this contributes to increased interest in them and activity around them.

The documents stored in a digital library can be accessed by the user through a series of services. Some are "classical", such as *search* -which allows users to locate documents containing one or more key terms- and *retrieval* of documents. Other services complement these, thus turning them into advanced systems that can classify documents and/or the results of queries according to one or more criteria specified by the user. The user can also manipulate documents and generate other new ones, as well as interact with the library to "improve" his/her knowledge of the library and viceversa.

Of the possible desirable qualities in a library, the following receive special attention in this thesis: firstly, the capacity of the library to offer access to heterogeneous documents through its services, which allows additional information about them to be discovered (for example, relationships between them), and secondly, that the user should be able to ignore the existence of multiple versions of the same document or not (as required by the circumstances). That is, library documents are normally related one with the other and this information should be accessible, able to be manipulated automatically, and of benefit to the user and the applications. In addition, the library should be able to provide users with any version of a document. All this should be possible without reducing the basic functionality of any library, which means that the new services must be able to be *integrated* in a "classical" library.

Heterogeneity in documents

In addition to the variety and complexity of the services offered to users, the heterogeneity of the information is characteristic in digital libraries. A digital library may contain heterogeneous data at a structural level (structured, semi-structured, non-structured information), according to its nature (text, video, audio, ...), its format (.doc, .tex, .fm, ...), etc. Concentrating on the documents, they can be subdivided into structured (made up of well defined elements between which there is an inclusion hierarchy) and unstructured documents. Even structured documents can be of different classes: as does the inclusion hierarchy between them.

However, heterogeneity in structured documents may be due to the diversity of structures[66], or to the nature of the information they contain. The structures introduced artificially by the information manipulators adequately satisfy the purpose for which they were designed, but they have the disadvantage of hiding information about its semantics (and even making it inaccessible) [65]. The possibilities of having access to digital documents whose structure accurately reflects the semantic structure of the associated abstract entity are many. Some will be presented when access to internal fragments, the detection of citations and the modelisation and exploitation of these relationships are commented on. This aspect has, up to now, been undervalued in many of the implementations of existing digital libraries. Formats and structures where relevance is given to the document's presentation (for example HTML [112]) have been widely used in these implementations, thus leaving the semantic structure of the document hidden (see [81, 80, 78, 74]). This problem is not so much due to the lack of technology to support it (both SGML [75], initially, and XML [117] later, allow structures and labelling to be defined at will), as to the fact that, as far as documents are concerned, the urgency of enabling mass access has relegated the aspects related to its semantics, structure and manipulation to a secondary level.

Relationships between documents

The generic services that can be found in any library are those that allow the search and retrieval of documents, while the advanced services are defined, in each case, with respect to the needs or possibilities of each particular library. The classification of the query results according to similarity of topic, origin, author or other criterium are some examples. Another desirable aspect incorporated in many libraries is the possibility of "navigating" the collections: the user starts the exploration in a guide document which has document references grouped into collections which can, in turn, be subdivided into smaller collections. Any of the mentioned classifications (the collections are the result of a classification process) is a reflection of the existence of semantic relationships between the documents grouped in the same category: documents with the same topic, documents created on the same date, documents cited by the same authors, etc.

In most libraries these relationships are expressed through links created manually [26] and those cases where some type of automatic detection of relationships of any kind have been achieved are very scarce. This in itself is a limitation but, in addition, the possibilities of extracting the information implicit in the relationships are restricted by the way in which they are modelled. The most frequently used options are: the already mentioned one consisting of modelling relationships as links inserted manually in the

document, or as metainformation associated to documents [23]. The insertion of links in the documents gives rise to hypertext, and this means that the way of "querying" the relationships is to navigate through it. Some kind of querying that does not require this navigation has only been considered in very few cases, and implemented in even fewer: It can therefore be concluded that the relationships between documents has been, up to now, a little exploited aspect. However, it is in the metainformation (the relationships are a particular kind of metainformation) where the possibilities to advance towards a better semantic use of digital libraries is to be found. This is one of the challenges for the future and it is in this field where a great deal of the professional and research effort will be centred.

Manipulating documents

Documents present in a library can be reused to create new documents. The manipulation of documents can consist of a format conversion, so that a copy with the same content but different format is obtained; for example, the generation of HTML pages for the final presentation in a user interface is a format conversion. Another example is the composition of collections, series or journals by the inclusion of articles. This reuse is really a composition of documents that can be expressed as a set of links between the framework document and its fragments [109]. Even more interesting is the reuse of fragments of documents instead of complete documents. The manipulation of fragments requires the characterisation and access to the fragments of other documents, which is possible if the work documents are structured.

Document versions

Over time, documents can be modified, which gives rise to new versions of the modified document. These historical versions, which owe their name to the temporal factor of the changes, coincide partially in their content, the part affected by the modification being different. In some cases it is interesting to have (or be able to obtain) all the versions of a document; this is the case, for example, for the historical versions of the preceptive documents in the legal sphere, which must be read exactly as they were at the time a sentance was given for it to be understandable. Also, in some cases, authors describe these modifications as a new document or part of another document, so the original version, the modified version, and the document containing the modification (which is a separate document with its own identity) coexist. For each modification, the author cites the document fragment in question and indicates how the said fragment could be modified (eliminating it, substituting it, ...).

The existence of multiple versions of a particular work implies a relationship between them, and this has been dealt with as a question of maintaining the database and the links between the different versions of the same document stored [38]. The difficulties of this approach arise from the risk of an explosion in size in the database if the modifications are frequent or if the documents with different versions are very large; while for the maintenance of the links, each link that affects a document can affect all its different versions.

The architecture of digital libraries

The architecture of a digital library is defined according to the functionalities required in it. It should facilitate the distribution of tasks among the services of the library. It is possible to extract a reference model for the basic functionalities: queries, document retrieval and navigation[1]. There are also proposals which concentrate on services oriented to the treatment of relationships: detecting semantic relationships of the type citation [?], management and maintenance of the library links [32, 97], and others which enable users to create links [33]. The integration of databases and services requires a protocol to regulate the interaction between the services participating in each operation. The incorporation of new services supposes a modification of the system: its interaction with the existing services means the protocol must be enriched.

Once the implementation phase is reached, in practically any library, it is possible to find indexing and search components, as well as a user interface to implement the basic services, alongside the databases (documents, metadata, others). In the case of heterogeneous (and/or distributed) libraries, there are also "integrating" components[2]. Finally, the fact that the documents (or data in general) that make up the library can, in turn, be distributed among different elements and databases should be taken into account, as this means that the architecture of the data must be considered. The incorporation of a new service supposes the incorporation or substitution of a component or set of components, whose cooperation implements the whole group of services of the library: the new service and those existing previously.

The services designed for the manipulation and exploitation of relationships are scarce and mainly concentrate on the management of the links [33] or the above mentioned citation detection. The exploitation of relationships and manipulation of documents have not yet been integrated into many libraries or protocols.

1.2 The proposal

The objectives outlined in the previous section include the extraction of information in the documental libraries on the relationships between documents and the use of some of this information as part of the generation process of new documents (to be precise versions of documents). The incorporation in a digital library of services "oriented to relationships" is considered, so that each service can be accessed through a series of interfaces which form part of the interaction protocol between these services and the other services present in the library. It would seem feasible to get an architecture based on a basic services model, so that the new proposals can be integrated without affecting the existing services. It is assumed that it is the structured documents which allow fragments to be reused in the composition of new documents. However, to obtain really advanced services and semantically coherent documents in document generation, it is essential that the structure of the digital documents stored in the databases of the library

[1] Navigation is a later addition than the services of search and retrieval of documents, but because of its widespread use in libraries (in degrees of variable sophistication) and the fact that the libraries created in recent years incorporate some kind of navigation, it can be considered a classic functionality.

[2] These components can also be refered to as "mediators".

accurately reflect the semantic structure of the abstract entity that the digital copy represents. Only if the logical structure associated to the abstract document is taken into account, is it possible to obtain a digital copy so as to be able to model and exploit the relationships between documents, with a maximum level of granularity, as well as create new documents where the historical modifications suffered will be reflected. Given its semantic origin, the necessary structure is obtained from the document *content*. The most adequate way of achieving this would seem to be the most natural: the same method that allows a reader to create a mental image of the said structure while reading any copy of the document from start to finish.

It is not considered necessary to store the copies corresponding to the different versions of a document: it is possible to generate them, provided the modification relationships between documents have been adequately represented. The possibility of obtaining an adequate representation for the relationships between documents and their fragments, and the (prior) detection of these modifications are closely linked to the availability of the document structure. An adequate representation of the said structure allows the link between the fragments mentioned in the texts and the fragments that are part of the stored documents to be established. In addition, if it is possible to address the said fragments, the integrity of the document in which they are located can be maintained, while, at the same time, the fragments can also be reused in as many operations as necessary. It would seem possible to generate the versions by deducing the rules of composition of the virtual document being created, instead of storing the composition rules directly. The initial hypothesis is that, given a graph containing the information on the structure of a document and the modifications that affect it, it is possible to consider a traversal of this graph in such a way that the new version is created during the traversal of the said graph.

1.3 Implementation Prototype

The exploitation of the relationships and access to multiple versions of a document are necessities which are especially relevant in certain environments where the manipulated documents have a rigid semantic structure. This is the case with legal documents, that include all the characteristics of interest in this thesis: they are highly structured, closely interrelated one with another and a jurist needs to have access to the version of a document (for example, a law) as it was at a particular moment in time. The differences between different versions are due to partial modifications to the content of the document, in such a way that each modification gives rise to a new version. Given their temporal nature, they are considered to be historical modifications.

The prototype where the theoretical proposals of this thesis are implemented is a digital library of legislative information. It is a specially interesting environment, with the additional peculiarity that the modifications are, in turn, expressed as part of the content of a document where the affected document is cited, in order to then include the modification.

1.4 Evolution of the work

This thesis began by defining the aims, influenced by the needs as far as functionalities are concerned. These aims are then considered in an environment such as that of the prototype. Thus the need arose to gain access to the versions of a document, to exploit the relationships between documents and to have a semantically structured copy of any document that is to be automatically versioned. From this starting point, the work continued in a series of stages that correspond, almost unidirectionally, to the organisation in chapters of this memory:

- The stage following the definition of the aims was the expression of these requisites as a set of services. The principal initial requisite was to consider them as services which could be *incorporated* into a library, thus enriching its functionalities.
- The possibility of having some means to express the semantic structure of a document was then investigated: to obtain a copy of the document that reflects the said structure and allows access to its fragments. The result of this stage was an algorithm to obtain a copy of a document whose logical structure reflects the semantic divisions of the abstract document. Entry to the said algorithm is through one of the copies of the document whose logical structure does not reflect the semantic divisions. The algorithm analyses the content of the document to detect the semantic divisions.
- Once the properly structured documents were available, the representation of the modifications and relationships as heterogeneous links was considered. The associated information was stored in a labelled links database, from which the graph needed for generating new documents could be recomposed. An algorithm for the generation of versions traverses this graph, using the structural tree of the versioned document as support, and progressively constructing the new version with each step on the traversal.
- Finally, the automatic detection of the relationships was implemented in the prototype, on the basis that the said relationships are detectable as citations between documents and that positive experiences exist in the detection of citations in legal documents.

1.5 Organisation of the thesis

Chapters 2 to 4 deal with the first three points mentioned in the previous section. Each chapter includes a review of the state of the art in that field, the proposal and a final section where it is compared to other previous proposals and the relevant aspects of the proposal presented are analysed.

Chapter 2 introduces the differences and similarities between the abstract document as an entity and its corresponding digital copies: those with the same content but different formats, etc. It is thus possible to state the proposition of the chapter: An algorithm to obtain a digital copy of a document whose logical structure accurately

reflects the semantic structure implicit in the conceptual document. Given that structured documents are the only ones able to exploit the relationships of interest in this thesis (those affecting parts of a document and not only entire documents), attention is focused on this aspect and an algorithm that allows a semantically structured document to be obtained from one whose structure does not comply with the semantic criteria is considered. The use of XML is proposed as the ideal standard for modelling the logical structure of a document, and the implementation of the convertor algorithm on a tool for manipulating XML documents is considered, as this would simplify the lexical analysis.

The existence of many types of relationships between documents gives rise to a labelled graph in which the vertices are not mere documents, but fragments of documents affected by the relationship. The exploitation of this graph, obtaining partial graphs and subgraphs from the original, and querying the said graph allow new documents to be generated. Chapter 3 looks at some types of relationships: structural, citations and modifications. The structural relationships were obtained in the previous chapter, as they are hierarchical relationships in the tree associated to the logical structure of the document. An algorithm for generating historical versions is considered. It is based on a recursive treatment of the *versioning graph* obtained from the tree associated with a structured document and the links that represent the modifications which affect fragments of the said tree. The information on this graph can be stored as a links database, in such a way that the traversals on the graph become queries to the links database. The implementation of this algorithm also allows the languages provided by XML to address the subtrees in the documents (XPath [115]) to be used, as well as its contributions to links modelling (XLink [119], XPointer [116]).

Chapter 4 gives a brief review of the most usual services in digital libraries and an update on the functionalities offered in some digital libraries, their expression as services and their impact on the architecture and protocols used. In this chapter, an architecture is proposed in which classical services are incorporated into digital libraries alongside the new services dealt with in this thesis: that is, the exploitation of relationships, the manipulation of documents and querying relationships. Document translation services are also integrated, thus allowing a structured document to be obtained through its semantic structure. The services proposed in this chapter correspond th the algorithms described in previous chapters.

Chapter 5 is dedicated to the prototype. The document databases on which the above proposals have been tested are presented. The chosen documents are heterogeneous with respect to the type, some of them being highly structured. They are thus the preferred candidates for experimentation on the algorithms presented in chapters 2 and 3. The possibility of automatically detecting some of the relationships used in chapter 3 is also commented on, as this would enable the relationships considered to be dealt with completely automatically. However, this kind of detection is itself an area of research, thus restricting its inclusion in this work to the feasibility of the said automization in certain fields with well defined conditions, as is the case of the prototype considered here.

Finally, chapter 6 summarises the conclusions arrived at in this thesis.

2

Documents in digital libraries

Contents

Documents are the main elements in digital libraries. However, the definition of such a concept is not precise and when used, there is only a general understanding of what a document could be in the context the term is used. This derives from the fact that the mental representation of what a document is can differ from one community of users to another. As digital libraries offer services to various communities, one such service is *retrieval of documents*. When a user asks for a document, he/she has an abstract entity in mind that may not match exactly any (digital) document in the library, or it may match more than one document. In these cases, the required *document* has to be composed on demand to deliver it to the user, or the user is asked to select a copy from the set of candidate document copies. If the document to be delivered to the user is obtained by composing fragments from several documents, it is easier to accomplish this task if working with *structured documents*. Different document structures may be associated with a unique document. These structures can describe the document content, the document's physical aspect or how the document can be obtained by composition of *document fragments*.

Documents made by composing well-delimited pieces of content (that can present an inclusion hierarchy between them) are said to have a *logical structure*. Sets of documents that have similar logical structures (they share the set of division types allowed, as well as the inclusion rules permitted between components) constitute a *document class*. Such classes have grammars that define what elements can be found in any document of the class, as well as the inclusion rules between them. These grammars, as well as the documents themselves, can be described by means of specially designed standards. Hereafter, attention is centered on the XML standard. XML is well suited for structured documents, and provides its users with tools that facilitate their description. It also facilitates their manipulation, taking advantage of their structure (this aspect is dealt with in the next chapter).

As the concept of what a document is can differ from one community to another, this can also happen with the idea of which document components are permitted in a class of documents. That is, a document can have different logical structures, depending on the author that created it. One of the most interesting structures to have available is the abstract structure implicit in the document content semantics (that exists even before any physical copy of the document entity is created), due to its inherent semantics and the possibilities it offers for treatments that exploit it. Given that, at the moment of copy creation, the criteria used by authors to structure the digital document are mostly formatting considerations instead of the semantic division of the abstract document, there is a need for an algorithm able to capture the semantic structure of a document, and to create a document copy whose logical structure accurately reflects the abstract document semantic structure: a semantically structured representative. The input to such an algorithm would be one of those document copies that do have arbitrary (non-semantic) structures.

There are three possibilities to consider about existing document copies: the input copy is tagged and its DTD is known, the input copy is tagged and there is no knowledge about its DTD and, lastly, the input document is not tagged. The case considered in this chapter's proposal is the second one: input documents that are tagged, but whose DTD is not known. Tagged documents become more and more frequent with the expansion of markup languages, which makes this situation more probable. The first

case , that where the input DTD is known, is the one considered when talking about DTD transformations [27]. And the third one (the input document is not tagged) will be discussed in section 2.6 as a simplification of the situation considered by this thesis proposal.

The semantic structure will be valuable -as explained in chapter 3- for detecting references between documents, and for exploiting them. Legal document databases are good examples of documents with a precise semantic document structure, associated to the abstract document, which can be extracted from whatever copy of the document.

2.1 Documents

2.1.1 Abstract document, document copies and document versions

There is no precise definition of the concept *document* as used in computer science. According to the Spanish *Diccionario de la Real Academia de la Lengua* a *document* is *algo que da testimonio de algo* (something that gives proof of something). This definition is not very suitable when speaking about a digital library that has documents, considering one of the properties required in a digital library (presented in chapter 4): digital library users want intellectual works; that is, they access the library with the aim of manipulating intellectual documents, not digital files. The idea of what a document is can be different in various communities. For example, a document can be an abstract entity for a user of the library, and a physical file or web page for the library administrator. Moreover, the abstract entity of the user may match several copies in the library, or correspond to the composition of fragments coming from several library documents.

A similar idea can be found in the IFLA report [72][1], which specifies the difference between *work*, *copy*, and *manifestation*:

> A *work* is an abstract entity; there is no single material object one can point to as the *work*. We recognise the *work* through individual realizations or *expressions* of the *work*, but the *work* itself exists only in the commonality of content between the various *expressions* of the *work*. When we speak of Homer's *Iliad* as a *work*, our point of reference is not a particular recitation or text of the *work*, but the intellectual creation that lies behind all the various *expressions* of the *work*.

Similar considerations have been taken into account in some different digital library contexts. The NCSTRL [42] digital library architecture (presented in chapter 4) design includes the *data* architecture, to reflect the fact that a document may have several copies corresponding to different formats. They hide this information during all the user search process, only informing the user of the existence of such copies at the moment of document retrieval in order to ask him/her which one to retrieve. Other authors

[1] An equivalent classification was made in Europe under by the *indecs* (*in*teroperability of *d*ata in *e*-commerce *s*ystems) [73] initiative.

[14] make the difference between *copies* -different formats- and *renditions*, which are versions of a document where the content has suffered modifications.

Given that throughout this chapter we will talk about *documents*, we adapt the definitions that are fundamental to understanding the rest of this manuscript. An *abstract document* is the abstract entity an author or user has in mind; it corresponds to the *work*. Similarly, a document may have several *copies* corresponding to different formats (.doc, .pdf, plain text, .xml, etc.). All document *copies* share the same content; their only difference is in formatting aspects. The application of *modifications* to a document content results in document *versions* -note that they would be named *expressions* in the IFLA ontology-.

Example 1. Modifications originating different *versions* are frequently made of songs, theatre plays, technical books, course notes from one year to the next, laws, and others. □

Example 2. Legal documents, as for example rules, are entities that exist independently of the presence of copies of such documents. Moreover, the aspect (paper, electronic version, fragmentation of text in pages) of such copies is irrelevant, as the valuable information of these documents is in their content. □

2.1.2 The Document logical identifier

In the previous section it was explained that there can be several *copies* and *versions* of a given document. The question now is: "*What does a user refer to when interacting with a digital library: the abstract document, the copy, the version?. How are these versions managed in a digital library? Are the properties of a copy or version different from the properties of the abstract document? And, if not, how does the system differentiate them?*".

A first set of properties of a document can be distinguished that are proper to the intellectual object, and thus shared by all its versions or copies. These properties are *metadata* that provide additional information about the object or document. For example, the *author* of a novel is a valuable piece of information about the document.

When a user talks about a document, the intellectual object is normally identified first (the *Bible*, the *Iliad*, the *LRU* law, ...). This description is usually sufficient to identify the object. Only in some cases is it of interest to specify the *version* (version of the *LRU* at the present moment) or the *copy* (the *postscript* file, the *ascii* copy, ...).

Document identifiers should have some properties to facilitate the manipulation of the metainformation and objects associated to a given document. The permanence of these identifiers, which make for easy maintenance, should also be guaranteed. The first requisite is that there is a unique identifier associated to a document and all its copies/versions. The benefits of such a policy are many: more user-friendly identifiers, location-independent identifiers, etc.; a detailed explanation that completes our argumentation can be found in [31]. Properties of such identifiers include *uniqueness* and *multiple resolution*. *Uniqueness* guarantees that an identifier is unique in its namespace. If namespaces are well managed, uniqueness will be preserved in larger universes by adding the namespace identifier to the document identifier. *Multiple resolution* deals

with potential access to all versions/copies of a document from the document identifier. There can be several copies of a given book (for example, the *Bible*). These copies can be in different formats, and may even be fragmented in several files. Nonetheless, when a user requires a copy of the *Bible*, this identifier clearly designates a unique work, of which several copies can be offered. That is, there is a *Document Logical Identifier*, associated to the work or abstract document.

The question of identifiers is of great interest in the *reference linking* [2] domain [31]. Multiple definitions of what an *identifier* is and work to try to discover, treat or *resolve* identifiers in the most appropriate way, have emerged in this area. The *Serial Item and Contribution Identifier (SICI)* [87] standard provides rules for calculating identifiers for journal articles. The *Digital Object Identifier (DOI)* [94]. This standard defines a standard way to characterize an abstract work. This identifier is an implementation of a *Uniform Resource Name*, and it establishes a series of fields that inform an application about document copies, their location in the library collections, the provider, and some more information that could be useful for some applications. Other identifiers used in reference linking are the *ISSN (International Standard Serial Number)*, with an internal *article identifier* that discerns articles in the publication. Metadata about the documents is, therefore, related to the document identifier.

2.2 Structured documents

2.2.1 Document structures

Several types of structures can be associated to a document [7, 5, 59]. Classification varies from one author to another [66, 30, 106]. In any case, there is general agreement about the two main types of structures: *physical* or *layout* structure, and *logical* structure. The *logical* structure of a document shows it as the composition of abstract objects. The *layout* structure is associated to the physical placement of text on a page. A third type of structure is considered by some authors [106]: the *content* structure, which describes the purely semantic relationship within documents.

The *layout* structure divides a visible representation of the document into rectangular areas. It is the structure found in graphic formats.

The *logical* structure characterizes a document by the hierarchy obtained from the containment relationships between *abstract* objects; relationships such as those formed by references in the text do not form part of the logical structure. Parts of a structured document have *type* [58]. For example, an article or a book may have parts of type *chapter*, *section*, etc. The logical structure can be represented as a tree, where relations between nodes in the tree represent the inclusion hierarchy between document components. Document *contents* are always placed at the lowest level of the tree hierarchy. Documents that have a *logical* structure are called *structured* documents, in contrast to those where information is found as a chunk of text.

[2] *Reference linking* is the general term for links from one information object to another. It is mainly concerned with relationships derived from citations, which is why it is also referred to as *Citation linking*.

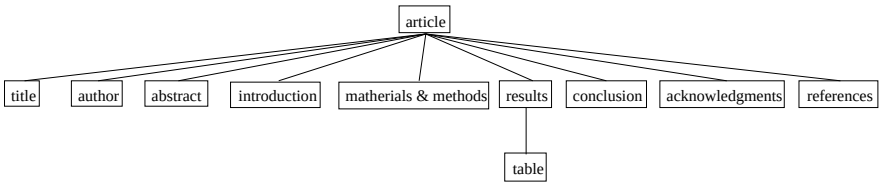


Figure 2.1: Logical structure of a scientific article. The lowest levels (paragraphs and content) are not represented, to keep the figure clear.

Example 3. Figure 2.1 shows the logical structure of a scientific article. This article is composed of a *title*, *author* information, an *abstract* that summarizes the content, an *introduction*, the presentation of *material* & *methods* used for experimentation, the *results* obtained, a *discussion* explaining the results, *acknowledgements* and *references* cited in the article. The *Results* include a *table* providing a visual overview. □

Logical structures can be classified as *content-oriented* or *layout-oriented* [106]. A logical structure is more content-oriented than another if its definition relies more heavily on internal content semantics; a more layout-oriented structure has a definition that relies more heavily on visual presentation (this means that some authors consider them as layout structures, even if they do have an associated tree that other layout structures do not have). Content-oriented structures require linguistic cues, as their structure is, in most cases, embedded in the content of the document [106]. Layout-oriented structures make automatic searching in documents difficult, as searches are commonly done with semantic criteria, not formatting. A well known example of layout-oriented structure is that of HTML pages, where document components are defined according to their final aspect in a browser.

Example 4. The article in example 3 could be found as an HTML page. In this case the content would be exactly the same, but the fragments would not be the ones in figure 2.1. They could be something like a sequence of `h4, h3, p`, and other HTML elements. □

The advantages of having the structure of a document are many. Access to document components allows them to be reused to compose new documents [14, 96]. Linking is improved when accessing the part of a document of interest [121, 109, 53], thus also improving navigation [40], and queries about documents can be refined to make sure that a search term is only of interest if it appears in a concrete part [1] of the document. The use made for linking and link querying in this thesis will be shown in chapter 3.

Multiplicity of structures for a document

The logical structure of a document divides and subdivides it into items meaningful to the human author or reader; since the logical structure is based on the "meaning" of the document parts, there is no single, unique, logical structure for a given abstract document.

Example 5. The trees in figure 2.2 are two different representations for the same Spanish rule, which could be found in different collections. As can be seen, they are very similar, but some subtleties, that do not affect their content, differentiate them. The

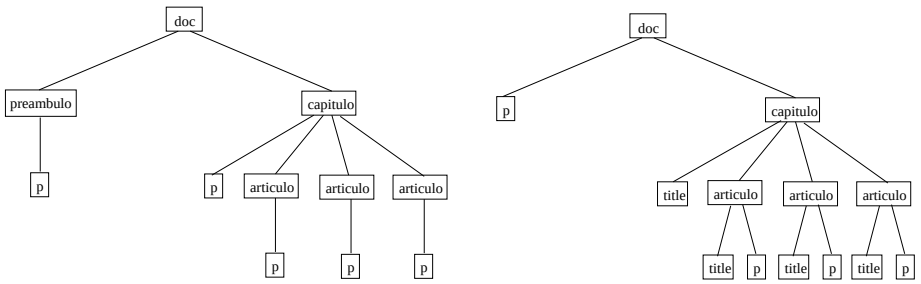


Figure 2.2: Two different logical structures for the same Spanish rule. The content (not shown in the figure) is the same, but the organization into logical parts is slightly different.

leftmost structure considers the document divided into two main elements: `preambulo` and `capitulo`. To the right, there is a division of the document into an element of type `p` and a `capitulo`. Yet more interesting is to see how the `articulo` elements are obtained in each case. When considering all the text of an `articulo` to be enclosed in an element `p` in the left option, the hierarchy to the right differentiates two elements in every `articulo` element: a `title`, and a `p`. Despite this difference in structure, the content is exactly the same in both cases. That is, we can state that it is the same document with two different structures. □

One common source of multiplicity in structures for a document is due to document copies created with formatting criteria in mind (layout-oriented logical structures). These structures are poor from a semantic point of view and difficult to exploit for retrieval or document manipulation purposes.

2.2.2 Capturing the logical structure of documents

The advantages of knowing the structure of a document, make this an active research field. The structure of documents is not always directly accessible from the available document copy. Sometimes input documents are images [19, 70]. When the input consists of files where the information has been stored according to layout criteria [123, 107], the document logical structure can be recognised by using layout cues or content information; a revision of this problem and aspects related to it can be found in [106]. There are other cases where the input are markup documents [102, 83].

The problem that has been generally dealt with is the discovering of the grammar that describes the logical structure. There is no knowledge of the target grammar, so layout cues are used to deduce the logical structure of the document. This being a difficult process, it is helped either by the availability of a subset of documents whose structure has been extracted manually [123], or by analysing the document in several steps such that each step adds some information until the complete structure has been recognised [107, 38].

There are other cases where the problem is centered in a single document: the goal is to discover a document logical structure (the class grammar is known or it is not meaginful). The utility proposed in [83] takes HTML documents as input and

discovers the containment relationships between HTML elements reached from the input document. It is a hierarchy discovery; the elements in the output graph are the same as they were in the input document, but the previously unknown containment relationships between them are now known. A second approach is presented in [102]; they use content cues to detect the presence of *concept* components inside a document, whose division into components is known. They divide the input document into contexts that are analysed, searching for the appearance of concepts; these concepts belong to a known collection of input concepts. Morphological cues, some knowledge of relationships present between input document contexts and concepts searched for, as well as concept taxonomies help them to recognise the presence and limits of the searched concepts inside input contexts. The problem dealt with in this thesis, to "find" a document logical structure, can be included in this last group. The comparison with these approaches will be done in section 2.6.

2.2.3 Document classes

What is a document class?

Documents that have a similar structure form a class, to which a generic logical structure can be associated. This structure is defined by a set of rules common to all documents in the class: types of elements that can appear within an instance of the class, which are the allowed containment relationships between these elements and properties that characterize each type of element (content type and -optionally- additional attributes). If there is a document that does not comply with some of the conditions attributed to the document class, it is not an element of the class.

These hierarchy properties, common to a set of structured documents, can be defined by using *context-free* [69, 37] grammars, or by a *tree-like* document model [59]. Generic logical structures describe how the documents of the class are constructed. A class general logical structure contains *definitions* of objects, while a document contains *instances* of these objects. The *tree* representation provides a quick visual idea of what the composition hierarchy between components is.

Example 6. Scientific articles are divided into parts. Besides *title* and *author* information, there is an *abstract* that provides a general idea of the subject to be developed in the article. The *abstract* is followed by an *introduction* about the state of the art in the working area, and the presentation of objectives. Next, there is a description of the *material and methods* used in experiments. Following that, there is an enumeration of *results* obtained with each technique. Interpretation and comments on the results is the part called *conclusion*. Actually, this part is optional, as it can appear as an independent object, or be embedded inside the *results*. *Acknowledgements* and bibliographical *references* close the article.

This generic structure (the matching tree can be seen in figure 2.3) is common to any scientific article, no matter the subject it deals with. Some articles may have one figure, others may have several figures, others only a table; but all of them share the same basic rules governing their structure described above. Consequently, scientific articles are a *class* of documents, characterized by sharing this common basic structure. Variations such as those described (number of elements of a given type, further subdivisions) are

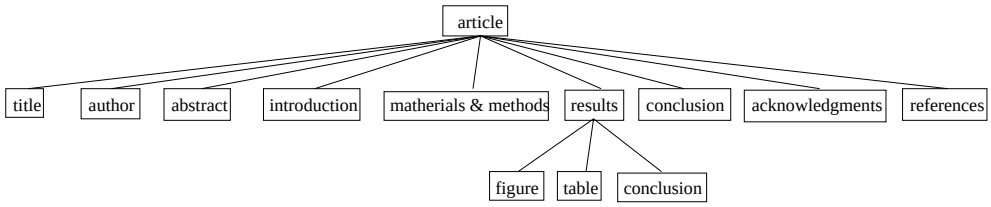


Figure 2.3: General logical structure for the "scientific article" document class. The lowest level (paragraphs) is not represented, to keep the figure clear. The example in figure 2.1 is an instance of this class.

particular to every instance of the class (to every particular article). □

Example 7. Spanish rules have an implicit and accurate grammar that describes any rule as the possible composition of a certain set of components. Hierarchy rules restrict what components may appear inside other components, as not all components are permitted to appear inside another element. These restrictions can be expressed with the grammar in figure 2.4. The tree in figure 2.5 also shows this hierarchy between Spanish rule elements.

All rules may have a first part of text, followed by a sequence of some elements (`libro, titulo, capitulo, seccion, articulo`), whose presence is optional. For every type of element, there is a subset of elements that may appear inside it. For example, an element of type `libro` may be composed of several elements of type `titulo, capitulo, seccion`, or `articulo`. This hierarchy continues to the last level in the hierarchy, where the text is found, organised in paragraphs (elements of type `p`). The most relevant quality of this hierarchy is that it is strictly forbidden for an element of an upper level to appear inside an element in an inferior level (for example, there can never be a `capitulo` inside an `articulo`); this property can clearly be seen in the class tree of figure 2.5. □

$$
\begin{aligned}
< norma > \quad &::= \quad < p >^{*}(< libro > | < titulo > | \\
&\qquad < capitulo > | < seccion > | < articulo >)^{+} \\
&\qquad < disposicion >^{*} \\
< libro > \quad &::= \quad < title >?(< titulo > | < capitulo > | < seccion > | \\
&\qquad < articulo >)^{+} \\
< titulo > \quad &::= \quad < title >?(< capitulo > | < seccion > | < articulo >)^{+} \\
< capitulo > \quad &::= \quad < title >?(< seccion > | < articulo >)^{+} \\
< seccion > \quad &::= \quad < title >? < articulo >^{+} \\
< articulo > \quad &::= \quad < title >? < p >^{+} \\
< disposicion > \quad &::= \quad < title >? < p >^{+}
\end{aligned}
$$

Figure 2.4: Grammar corresponding to Spanish rules.

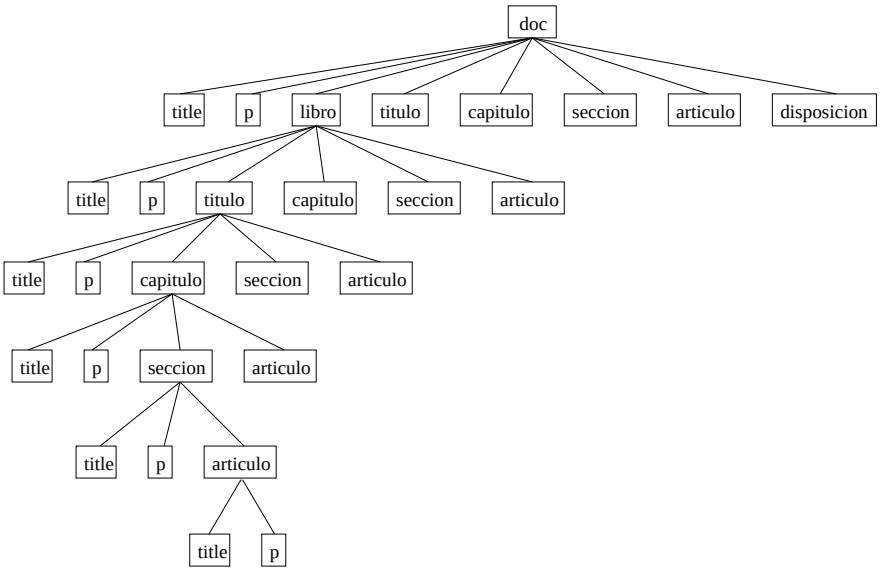


Figure 2.5: Partial representation of the tree for the *Spanish rules* document class. The containment hierarchy between components is mapped to the tree nodes' descendant hierarchy. Wherever a node of a given type `X` appears, the subtree with `X` at the root should appear in a complete tree representation.

Benefits of having the document class generic logical structure

Structured documents are widely appreciated in document manipulation applications like editing, formatting, and document composition. The availability of a generic structure together with documents that conform to it offers the possibility to improve such applications.

Formatters can use a declaration of formatting instructions for each generic element in the document class; there is no need to relate each document to all kinds of formatting directives, since they can be defined once for all the documents in the class. Translators can follow general conversion rules, applicable to all documents in the class, using a transformation grammar [76, 58]. Editors take advantage of the structure not only for the presentation but also to manage the inclusion and elimination of elements by the user [28, 84, 85]. It is also possible to transform a document instance to another one matching a different logical structure[60]; knowledge of input and target classes allows a series of transformation rules to be established between the two classes[3] [28, 84]. Querying documents can be done on structure; the query needs a document to be retrieved only if the search term appears inside a certain element. Documents can be composed [57, 6], assembling pieces extracted from documents, by declaring the rules that express how to assemble those pieces.

[3]When it is concerned with SGML or XML -explained in subsection 2.2.4- this is known as DTD transformation.

2.2.4 Standards for structured documents

A document's *structure* is additional information about it that has been stored in different ways at different times. In general, the approaches are: to store the structure separately from the document content, or it to store with the document content. Initial works with structured documents tended to store it separately, as there was no simple way to do it otherwise [3, 99]. *Markup* languages came with a solution to keep together structure and content, and therefore, ease structured document manipulation. SGML and XML are the most popular markup standards. They also have some valuable properties as data interoperability, semantic data description and legibility.

SGML and XML

The *Standard Generalised Markup Language (SGML)* [75] is an international standard for the definition of device-independent, system-independent methods for representing texts in electronic form. SGML provides the syntax to describe the logical structure of documents. It allows a structured document to be modelled, interspersing *markup* elements within document content. Markup consists of opening and closing tags, that surround pieces of text. In other words, the markup establishes where a document element starts and where it ends. SGML appeared in 1986, and since then its most popular application has been the *HyperText Markup Language (HTML)* [112], widely used in the Internet.

The *Extensible Markup Language (XML)* [117], which is a simplification of SGML, was defined to solve some problems (most of them syntactic ambiguities that an automated application is unable to resolve[4]) detected from the experience with SGML and HTML. XML is supported by the *World Wide Web Consortium (W3C)*, and the current recommendation dates from February 1998.XML documents have more restrictive syntactic rules than SGML documents. The minimal set of characteristics that a document has to comply with to be XML-conforming (that is, to be processed by an XML application) is reduced and simpler.

XML is *extensible*: it allows tag names to be created. Element names in an XML document are chosen by the document creator, permitting tag names to be chosen derived from the semantics in the document, and not from its format. This characteristic is the crucial one of XML, as it is the one behind *descriptive markup*: the tags around a chunk of text do not tell how to format it, or what to do with the document; they just say what it is. This is a major difference between XML and HTML: XML decouples the document from its presentation. While in HTML the name of the element is more related with the aspect the element will have when the document is presented to the user, in XML it is possible to use completely semantic tag names, and describe later formatting rules with stylesheets (CSS [110] and XSL [111]).

XML was created with the aim of being easily understood, easy to use, and to provide interoperability. It frees applications from formatting clues and -yet more important- it is possible to define namespaces [114] to treat semantic interoperability, for example, when exchanging data. Any document claiming to be XML conforming can be treated by an XML processor that will parse it and reject it if this is not the case; the problem

[4]See reference [29] for a comparison of SGML and XML.

of poorly tagged documents, which often occurs when using HTML and may cause an application crash is eliminated.

There are standards associated to XML, ruling how to add format to document classes (CSS and XSL), how to model links between XML documents (XLink [119], XPointer [116], XPath [115]). There are also tools to describe metadata (RDF [118]), and others that can be found in the W3C XML page[5].

The aspect of XML documents

Logical items in a marked-up document are referred to as *elements*. Each element is introduced by a *start-tag* and terminated by an *end-tag*. The content of the element is between the element tags and may contain further tags corresponding to nested elements. Elements may have properties, represented with *attributes*. Attributes are specified in the start tag of an element. Documents may also contain *entities* which reference external files, or that are abbreviations for constant strings.

An XML document is said to be *well-formed* if

- All tags are there.
- The begin and end tags match (with the possible exception of empty elements).
- All the attribute values are quoted.
- All the entities are declared.

An XML document is *valid* if it is well-formed and there is a document class general logical structure[6] it conforms with. This structure is described by means of a *Document Type Definition (DTD)*. A *well-formed* XML document can be treated by an XML processor[7].

The Document Type Definition

Both SGML and XML allow a DTD to be attached to a document. DTDs describe which elements are allowed in a document class, and the permitted inclusion relationships among them. DTDs are the formalism provided by SGML (and XML) to model document class grammar rules in such a way that they can be treated by an application: all documents in a class conform to the same DTD. Attributes and entities must be defined in the DTD, which may be included in the same file as the document instance, or reside in a separate file.

Example 8. Figure 2.6 is a fragment of the DTD for Spanish rules. It includes an entity definition: `libro.mdl` that is used later in the DTD, when declaring the element `libro`. If the content of the entity is replaced in the element declaration, the result is an element of type `libro` composed of an optional element `title`, followed by a sequence of one or more elements of any of the types `articulo`, `capitulo`, `seccion`, `titulo`.

[5] http://www.w3c.org/xml

[6] The general logical structure specifies which are the elements allowed and the allowed relationships between them.

[7] This is a main difference with SGML, where all documents must conform to some DTD.

The only attribute of an element `libro` is an identifier that uniquely distinguishes the element from all other elements in the document. The attribute type is declared to be `IMPLIED`; that is, the attribute is of optional appearance in elements and there is not default value for it. □

```
<!ENTITY  % libro.mdl
        "articulo|capitulo|seccion|titulo">

<!ELEMENT libro       (title?, (%libro.mdl;)+)>
<!ATTLIST libro       id   ID  #IMPLIED>
```

Figure 2.6: A DTD fragment extracted from the "Spanish rule" class DTD.

The most popular DTD in SGML is HTML. HTML has been widely used to create Web pages. This has contributed to SGML popularity, but at the same time, the flexibility in HTML has given rise to lots of poorly structured pages, from which it is difficult -or even impossible- to extract semantic information. For example, searches of document abstracts, when these documents are HTML pages, are not evident as there is no tag in the DTD HTML to express such semantic concepts. That is, with HTML, documents are modelled according to formatting criteria. Now, there are proposals for HTML conforming to XML, called XHTML [113], that will facilitate the syntactic treatment of these documents, even if the semantic problem persists.

There are also DTDs whose purpose is to describe the general structure of documents. *DocBook* [120] is an SGML DTD maintained by the *DocBook Technical Committee* of *OASIS*. It is mainly oriented for use with books and papers about computer hardware and software. One more DTD in the SGML environment is the TEI [103]. *The Text Encoding Initiative (TEI)* is "*an international project to develop guidelines for the preparation and interchange of electronic texts for scholarly research*". Work on this DTD began in 1987, when XML had not even been proposed. So, in its origin, it is an SGML DTD, while there is now an XML version. Its aim is to be general enough to allow modelling as many types of documents as possible, while permitting elements in the document to be characterized with attributes giving semantic information. This DTD has lent some important things to XML -such as *XPointer*s-, which, in part, explains why activity around this DTD has declined while XML has gained acceptance.

2.3 Documents in the legal domain

2.3.1 Abstract document, document copies and document versions

Legal documents exist independently of the way they are stored in a computer system [3]. For example, when talking about a given rule, everybody identifies the rule immediately despite whether the document is presented in one system or another, or

whether it is possible to access the complete text.

There may be several copies of a document with different formats. This is common to any document that is digitalized. However, concerning "official" documents in particular (laws, decrees, etc.), these documents are of a type that suffers several modifications during their life, resulting in different *versions* of the document.

2.3.2 Structured documents

Legal documents, such as rules and jurisprudence, are very well structured documents. They have a semantic structure that is fixed and strictly kept in every new document produced. That structure is useful for professionals to locate internal parts in large documents (for example, some rules take up tens of pages), and to find the portion of the document of interest at any moment. Indeed, references inside documents to other legal documents are in many cases references to internal fragments. Moreover, as they are used to take advantage of such structures, jurists, being conservative, tend to maintain these structures.

So, this is a domain where it seems an interesting thing to have a document *representative* where the semantic structure of the abstract document is somehow reflected [14, 56]. This would permit some manipulation of legal information to be automated, give jurists the possibility of profitting from utilities that help them in their daily work, and also ease access to these -sometimes difficult- texts for common people. Given that legal documents are of a certain *class* (rules, jurisprudence, ...), the structure of a given document and all its copies conform to the same set of rules (general logical structure). Actually, even if the document is modified, there will be little difference between the structure of two versions of the same document. The tree structure will be very similar.

Among the advantages of structured documents in general, it is of note that the structure of legal documents is intensively exploited in document references. It is also of great usefulness for composing documents [14] and obtaining navigation hypertext [3].

2.3.3 Standards for structured documents in the legal domain

The way to store the structure of documents in the legal domain has evolved with the appearance of new standards applicable to structured documents. Those approaches where the structure is kept separately from the document content [38, 3], have given way to those where the structure forms part of the text of the document [56, 12, 65, 55]. XML allows a document to be tagged according to its semantic structure, and provides additional standards and utilities to access (XPath) and manipulate document components in XML documents (XSLT).

The standards used for structured documents have been applied in the legal domain. SGML has been the object of several propositions and implementations. Some propositions consist of DTDs designed on formatting criteria [86, 51]. The clear disadvantage of such proposals is that it is not possible to automate access to the semantic structure of the document.

There are, however, more DTDs for legal documents, whose aim is to be as general as possible. That is, the grammar implicit in the DTD has to describe as many

different types of legal documents as possible [101]. The intention here is to provide access to documentation coming from several European Union countries, and to facilitate comparison between those documents. With this goal in mind, the *Legis* project [65] proposes to include the metainformation about the legal document in the digital version. Of course, to achieve such generality, the description of the structure has to stop at a high abstraction level (top levels in the class tree); more precision about the components of every document depends on the document class at its origin (the internal structure of one with Spanish rules is not the same as a French one, despite similarities). Another attempt -in the development phase- to provide common structure to legislation in the European Union is the *Eulegis* [55] European project.

A different option is the one taken by some people who have tried to apply standard DTDs to legislative documents. D. Finke [56] proposes extensions to the *TEI* DTD for legal documents. The extensions include the addition of new attributes and new elements. These new elements should reflect the particular structure of legal documents. Legal information has a good semantic structure where components are of a given *type*, defined in the legal domain. Without such extensions, the structure of legal information would have to be simulated with element attributes (for example, with `type` attributes). This solution, being a little artificial, is, in addition, not appreciated by legal specialists, who prefer to work with documents where they can directly recognise the structure with a first glance at the document (it is easier for a reader to recognise tag names than tag attributes).

Example 9. The following example shows what has been said. In both cases the tag marks the beginning of a semantic element of type *articulo*. The first tag corresponds to a modelisation of a Spanish rule with the TEI DTD. The tag name `div2` has been chosen among the available ones in this DTD, and the semantic type of the element is modelled in the `type` attribute (it is by no means more relevant than other element attributes, even if it contains the main semantics of the element). In the second case, the DTD has been defined expressly for this class of documents; the natural consequence is that the tag name is semantically expressive, which guarantees that it will emerge naturally over the element attributes.

```
<div1 id="14/198722" type="articulo">   (1)
<articulo id="14/198722">               (2)
```

□

2.4 Proposal for content-based semantic logic structure capture

The conversion algorithm presented in this section generates the semantically-tagged representative digital copy of a markup input document from another document copy tagged with different criteria. It obtains the logical structure of a document from its text, thereby showing that it is possible to *translate* a document from one copy where the semantic structure is hidden, to another document copy where the semantic structure is reflected in the document logical structure. This process is taken on by

the *document translator* component in part 4.2.6 of chapter 4. The resulting copy is thereby normalised, and can be used by any system component. To have such a representative will be crucial to be able to exploit relations between internal document fragments; this aspect is developed in chapter 3 about relationships.

The semantic structure of the document can be obtained from its content. Well-structured documents present a set of keywords inside their text that help the reader to recognise the start of document components. The presence of a keyword in the text marks the start of a semantic component. These components are the ones referred to as "semantic", as they come from content criteria, with no presentation aspects involved in their definition.

The input document to the algorithm is a tagged document copy, whose logical structure does not correspond to the abstract document semantic structure. The input document copy structure is, in most cases, based on formatting aspects (as is the well known case of HTML documents), hiding the semantic structure that is lost inside the input components content.

Structure recognition that allows the document copy to be obtained with a logical structure that matches the semantic document structure is content-based, guided by the target DTD; aspects present in the source copy structure and common to the abstract document, that cannot be discovered from the document content, are kept in the output copy. Decisions to create elements in the output are made from the content when possible. When the text does not provide information about logical divisions (lowest levels of divisions, that are indeed layout divisions, as for example paragraphs), they will be transferred from the source document, *trusting* its correctness as far as formatting aspects are concerned.

The output is a structured copy of the input document, with the same content, but with a different logical structure: the semantic one. This structure is reflected in the representative markup (the output is an XML document). While main divisions in the output document are created from input content, the highest granularity level -paragraphs and similar divisions that cannot be inferred from content- are preserved, as they come in the input document.

Considering the target document with a tree and focusing on elements (abstracting from text and attributes), it can be said that internal nodes are generated from document content, while leaves either come from the input document structure, or are generated using information from content and input markup.

The example in section 2.4.3 illustrates the output document obtained from a layout-oriented tagged document.

2.4.1 Inputs to the algorithm

In every application of the algorithm there are three inputs: a document d, a hierarchy h, and a vocabulary mapping v.

The input document

The input document, d, is a tagged copy of a document, with a logical structure that does not reflect the semantic document structure.

Every input document d to the algorithm complies with some assumptions (conditions), listed below:

- The input document is text, where there is content and markup.
- All text that is not markup is relevant. That is, it is also in the abstract document. There are no pieces of content dependent on the copy used as input.
- The text has keywords (in v) inside it that can guide the generation of elements in the output. The presence of every keyword marks the start of a semantic element.
- The start of one of these *semantic* elements, signals the end of all semantic elements, of inferior or equal level -as defined in the semantic components inclusion hierarchy-. For example, the start of a *chapter* in a book, implies that previous chapters are finished.
- Markup can be of any type. But there is a guarantee that all open tags are closed, and all closed tags were previously opened. This condition guarantees that presentation markup recovered from the input is reflected in convenient presentation elements in the output (there is no other way to recognise the limits of these elements).
- Lowest divisions in the output document tree cannot be characterized from text alone. They are layout divisions that also appear in the abstract document, or layout elements whose presence is always associated to a semantic element (semi-formatting elements).
- Granularity in the source document markup reaches the minimum paragraph level. This condition guarantees the presence in the output of layout divisions mentioned in previous conditions.
- Nesting in the input markup is limited to presentation elements. This condition guarantees the correct transfer of closing presentation tags. Nesting interspersed semantic elements inside presentation elements would break the granularity condition and obstruct the acquisition of element titles (semi-formatting elements).

The vocabulary mapping

A vocabulary mapping (v) guides the generation of markup in the output document. v is the equivalence between keywords that can be found in the document content, and elements in the target DTD. It is defined according to the target document class (semantic structure).

The target components inclusion hierarchy

h is information about inclusion rules in the target DTD. These inclusion rules are expressed as a level hierarchy of elements in the target DTD, such that if an element type *e1* is in a lower level than another element type *e2*, elements of type *e1* may contain elements of type *e2*, but elements of type *e2* cannot contain elements of type *e1*. If two element types are at the same level in the hierarchy, inclusions between them -in any

direction- are forbidden. This hierarchy concerns *semantic* elements in the target DTD: elements that are created from the vocabulary in the source content.

2.4.2 Output

The output of the algorithm is a tagged document *str* with the following characteristics:

- *str* content is the same as the content in the input document d
- *str* is the semantic representative (copy) of d, tagged according to the abstract document structure
- *str* is XML well-formed
- *str* is tagged according to rules given in h and v
- The nesting of elements in *str* conforms to inclusion rules in h
- Internal nodes in the associated tree are semantic elements, while leaves are created on presentation criteria.

2.4.3 An example

Input document

The document in figure 2.7 is a simplified fragment extracted from a Spanish rule.

```
<doc>
<p>Ley 1.</p>
<h4><a>CAPÍTULO I.</a> DEL REFERENDUM.</h4>
<p><a>Artículo Primero.</a></p>
<p> Texto del artículo primero.</p>
<p><a>Artículo Segundo.</a></p>
<p>Texto del artículo segundo.</p>
<p><b>Artículo Tercero.</b></p>
<p>Texto del artículo tercero.</p>
</doc>
```

Figure 2.7: An input document to the extraction algorithm

A reading of the text -ignoring the markup- shows that the document has a chapter (`Capítulo`), which in turn contains three articles (`Artículo`). A second look considering markup, refines the information in the following way:

- There is a paragraph (`Ley 1.`), which is not inside any document subdivision.
- The chapter has a title: `CAPÍTULO I. DEL REFERENDUM`.
- The text inside the first `Artículo` is a paragraph: `Texto del artículo primero.`

- The text inside the second `Artículo` continues until the beginning of the next `Artículo`.
- The text of the third `Artículo` expands to the end of the document.

Output document

The output document from the algorithm is in figure 2.8 and its associated tree in figure 2.9. Markup in the output reflects the division recognised when reading the input content, by contrast with the input document, where the markup did not correspond to it. There are three `articulo` elements placed inside a `capitulo` element. These are internal nodes in the document tree. Element titles have been recognised with the help of input markup. Presentation elements (paragraphs in the figure) are preserved from the input document. They are leaves.

```
<doc>
<p>Ley 1.</p>
<capitulo><title>CAPÍTULO I. DEL REFERENDUM.</title>
<articulo><title>Artículo Primero.</title>
<p> Texto del artículo primero.</p>
</articulo>
<articulo><title>Artículo Segundo.</title>
<p>Texto del artículo segundo.</p>
</articulo>
<articulo><title>Artículo Tercero.</title>
<p>Texto del artículo tercero.</p>
</articulo>
</capitulo>
</doc>
```

Figure 2.8: Output document resulting from the application of the extraction algorithm to the document in figure 2.7.

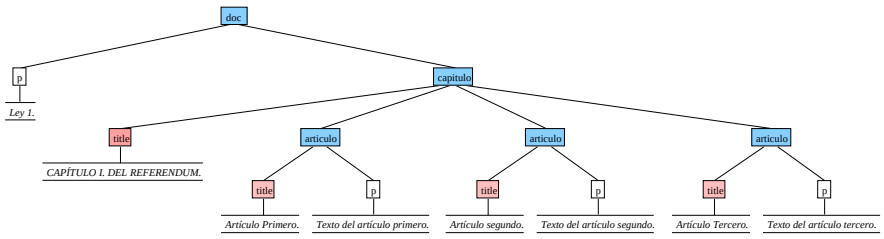


Figure 2.9: Tree for the output document when transforming the document in figure 2.7. Internal nodes are semantic elements; couloured leaves are obtained by using information from input content and markup, and uncoloured leaves are formatting elements transferred from the input document.

Hierarchy

Information about the target DTD (`Rules` DTD) needed by the algorithm is the hierarchy in figure 2.10. The tree expresses inclusion permitted between semantic elements in the target DTD grammar explained in the example in section 2.2.3. The tree is completely expanded in its leftmost branch. The subtree formed by an element and their descendants is always the same, wherever the element appears in the tree. For example, a `capitulo` element can contain elements of type `seccion` or `articulo` as children or descendants. But an element of these two types could never have a `capitulo` element inside.

Every element type has an associated *level* in the hierarchy, which is the lowest tree level where the element can appear in a top-down tree traversal (the depth of the node). A `capitulo` element is therefore from level *3*, a `seccion` element from level *4*, and elements of `articulo` type are from level *5*; elements of level *3* may contain elements from levels *4* and higher, but the inverse inclusion is not possible.

During the generation of the output document in figure 2.8, the hierarchy is queried to close semantic elements of equal or higher levels before opening new ones. Thus, when a new `capitulo` begins, all `articulo`, `seccion` and `capitulo` elements open up to this point, are closed before opening this new `capitulo`.

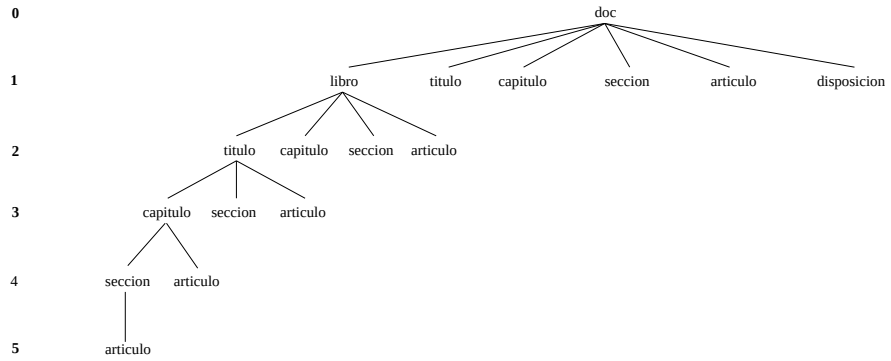


Figure 2.10: Inclusion hierarchy between semantic elements in a Spanish rule. Partial representation (the tree is completely expanded in its leftmost branch).

Vocabulary mapping

The fragment of the vocabulary mapping used in this example is in table 2.1. The semantic elements created are `capitulo` and `articulo`. A `capitulo` element starts where the string *Capítulo* appears in the input document, while `articulo` elements are recognised by the presence of the *Artículo* string.

2.4.4 The extraction algorithm

The algorithm behaviour simulates the way a human being recognises a document logical structure during reading. Implicit knowledge about the desired structure enables a reader to extract a document structure from its text, while reading. Some vocabulary

Text vocabulary	*DTD element*
`Capítulo`	capitulo
`Artículo`	articulo

Table 2.1: Fragment extracted from the vocabulary mapping of Spanish rules, used during the application of the transformation algorithm to the input document in figure 2.7.

in the text indicates when a new element begins (thus, ending another one). For example, a reader knows a chapter is finished because there is a new chapter that starts. That allows the reader to construct in his/her mind a logical structure of the document (how many chapters the book has, how many sections there are in every chapter, etc.). Linear traversal of the source preserves the order of the text in the output document.

There are two information elements that give information about the target document class logical structure and how to apply it:

1. The mapping of vocabulary in source content to markup in the output representative (mapping of document content domain ontology to system ontology), v.

2. The inclusion hierarchy between semantic components (grammar expressing the semantic component inclusions), h.

Definition 2.1 *The extraction algorithm is a function* $\delta : D \times V \times H \longmapsto STR$, *where*

- D: *set of XML well-formed documents*
- V: *set of vocabulary-mappings*
- H: *set of hierarchies*
- STR: *set of XML semantically-tagged documents.*

Each application of the algorithm is given by $\delta(d, v, h) = str$, *where:*

- d: *a document from* D, *where terms from the vocabulary* v *are found*
- v: *the vocabulary-mapping between keywords in* d *and markup in the output document*
- h: *the containment hierarchy between elements in the output document*
- str: *the output document, marked according to* d, v *and* h.

Conjecture 2.1 *In each application of the algorithm,* d, v *and* h *are matching entries in the sense that: vocabulary in* d *and* v *are the same, and element types in* v *and* h *are the same.*

A mismatching between d and v would cause the algorithm to transfer the source document as it is to the output. A mismatch between v and h would cause the algorithm to end in an abnormal state.

Creation of target tree

Types of nodes

The output document copy is well-structured and can be represented by a tree, as was explained in section 2.2.1.

There are two types of nodes in the output document tree:

- *Semantic nodes (or elements).* These are nodes created from content. They are internal nodes in the tree.

- *Formatting nodes.* These nodes are created using source markup. They are leaves in the document tree.
  If the source document has redundant markup, it can still be said that these nodes are either leaves, or placed at the bottom levels of the tree in such a way that any ancestor of a semantic element can be a formatting node. In a top-down look at the output tree, the *lowest* levels in the output tree come from the input markup and the top level elements are generated from the document content. From now on, an optimistic view will be taken and they will be referred to as *leaves*[8]

Creation of nodes

The generation of *internal* nodes is done from vocabulary in the input document. The generation of *leaves* in the target tree is done according to the following criteria:

- *Leaves generated by a combination of knowledge from the content and markup.* These are in most cases element titles. Each text fragment, that will be placed in a leaf in the output, is enclosed by a non delimited number of open and close tags. Open tags precede the text, forming a sequence of consecutive tags. By contrast, matching close tags can be found interspersed with text; the last of them marks the end of the output element.

- *Leaves imported from the origin.* These nodes have text where there is no keyword that could guide the element generation. We can therefore presume that the tagging enclosing such text was placed there on a *presentation* criteria, and that it could be worth preserving this lowest level of fragmentation. So, it is written with no modification to the output.

Example 10. Figure 2.9 is the output tree obtained by the application of the algorithm to the example in part 2.4.3. Internal nodes (in blue in the figure) are semantic elements; leaves coloured in red are obtained by using information from the input content and markup, and uncoloured leaves are formatting elements transferred from the input document. □

[8] In an ideal situation, there is no redundant markup in the input document. That is, there are no fragments of content -with no keywords to guide element generation- surrounded by several levels of tags. For example, there are no paragraphs with the aspect `<p><p><p> This is a paragraph.</p></p></p>`.

Creation of the target text

The target document is a structured document. Document content is transferred as it comes from the source. Concerning markup, the algorithm **opens** a new element in the target document in the following cases:

- When it encounters a keyword from v in the input content. The equivalent element is open in the output document. In this case, it creates a *semantic* element. For example, a new `capitulo` element is created in the example in part 2.4.3 when the keyword *Capítulo* is found in the text.

input	*vocabulary mapping*	*output*
<any-tag> keyword1 text ...	keyword1 T1	<T1> keyword1 text ...

- When it finds a piece of text in the input after an input tag, and there is no keyword in the text. This is a *presentation* element, that will be transferred untouched to the output. This is, for example, the case of paragraphs.

input	*output*
<any-tag> text	<any-tag> text ...

Elements are **closed** in the output document when:

- A new semantic element is open. For example, the start of a new chapter implies the end of the previous one. The closing tag will precede the input tag of the new element.

input	*vocabulary mapping*	*output*
<any-tag> keyword1 text ...	keyword1 T1	**</T1>**<T1> keyword1 text ...

- The closing tag of a *presentation* element is found.

input	*output*
...</any-tag>	...</any-tag>

Algorithm functioning

The algorithm (in figure 2.11) is designed to work on top of another application (an XML parser in the implementation) that provides it with inputs, containing pieces of text from the source document. As there is no direct control of the algorithm on how the words sent to it are created, the algorithm tries to generalise and considers the possibility of receiving a content keyword fragmented in several pieces.

Inputs to the algorithm come from the application it is implemented on top of. The input alphabet received from the application is a collection of events: STARTDOC, ENDDOC, STARTTAG, ENDTAG, TEXT.

STARTTAG, ENDTAG, TEXT events are accompanied by a string that is the element name or text recognised by the application. STARTDOC and ENDDOC are events that come alone, without additional information.

Reactions to inputs are explained below:

- STARTDOC: A STARTDOC in the source document provokes the opening of the target document. It can be a simple action like that, or any kind of action to be done every time a new document is created (for example, inserting a header at the start of the document).

- ENDDOC: The end of the source document advises the algorithm that it has to close the target document. Before closing the file, all open elements in the output not yet closed, are closed in an ordered fashion. This guarantees that the algorithm ends correctly and that the file it creates is also correct[9].

- STARTTAG: An open tag in the source document is not by itself sufficient to take a decision on what to write on the target document. Actually, the source tag may be a tag to preserve in the target (it opens a "formatting" element) or a tag to be ignored when writing to the target (the element in the source open with this tag contains keywords in its text, that will actually guide the markup in the output). It is not possible to know what kind of tag we are on at the moment, until we advance to its enclosed text. So, the tag is kept as a *candidate* to be preserved or to be ignored.

- ENDTAG: When we get to a close tag in the source, what to do with it depends on what we have previously done with its corresponding open tag. If we have written its partner tag to the output, we will also write this closing one. If we have ignored the open tag, the closing tag is also ignored.

- The text (TEXT) is the crucial information to decide how to markup the target document. Every time a piece of text is obtained from the source document, a decision is taken by examining a portion of input text. This text can be the string received, or a string composed by concatenating this one with other text fragments received before. At the moment, it is enough to continue with the explanation to know that there is a text fragment to be analysed; the selection of text to be examined will be explained later. There are three possibilities: the input has the start of some keyword, the input contains a complete keyword and the input does not have any keyword or any start of keyword.

  - If the current item is the *start of some keyword* (a keyword in v does not solely have to map one of the tokens returned by the parser to the application), this fragment is kept, to verify, in the next event, if it completes the keyword or not.

[9] XML well-formed and "semantically" valid.

```
switch input do
STARTDOC:  opens the output document
ENDDOC:    ends the output document
STARTAG:   keeps the tag until text is found
ENDTAG:    if the tag closes an element with no content
             then ignores it
           or else
             if it closes a formatting element
               then closes the element in the output
               or else ignores the tag
             fi
TEXT:      s = string to analyse[a]
           switch s do
             start of keyword:    keeps it till the next input comes[b];
             complete keyword: opens a semantic element in the output;
             no-keyword:       if it is at the start of a formatting element
                               then
                                  opens the element in the output
                               or else /* inside text of an element */
                                  writes s to the output
                               fi
end
```

[a] s can be the string received with the input, or be the result of concatenating the input string with a start of keyword kept from the previous iteration.

[b] If the next input is also text, it will be concatenated with this string to obtain the string s to be analysed in that iteration.

Figure 2.11: Structure extraction algorithm evolution at every input event.

- If the input contains a *complete keyword*, it marks the beginning of a *semantic* element which is open in the target and the input text is written. If there are elements of equal or lower levels than the one just opened in the target document, they are closed before starting this one.
- The last possibility is a text fragment that does not contain any keyword, and nor is it possible to start one. This may be due to one of two exclusive reasons: it is at the beginning of an element to transfer to the target document as it is, or the text is in the middle of one element content. If it is the beginning of a *formatting* element, the element is opened and the content is written. If it is text in the middle of the element, it is just written to the target.

The string to be examined in every iteration is either the input text, or formed by the concatenation of the input text to some start of keyword kept from previous iterations.

Proposition 2.1 *The scanning algorithm always ends.*

Proof: The sequential traversal of the input document guarantees that its end will always be reached.

Proposition 2.2 *The output document, h, is well-formed and its logical structure is semantically correct.*

Proof: Well-formedness is obtained by construction: semantic elements are guaranteed to be opened and closed in the correct order, since the start of a semantic element is always preceded by the closing of all semantic elements of equal or lower level. Correct nesting is guaranteed because the track of open elements in the target is done using a LIFO structure (stack). At the end of the execution, all elements not yet closed, are closed.

Lemma 2.1 *At the end of the document, all elements will be closed.*

Proof: Elements not yet closed in the target document when the end of the source document is found are kept in the open (target) elements stack. The algorithm does not end until this stack is empty.

Proposition 2.3 *Non-semantic elements are always placed at the bottom tree levels.*

Proof: For elements directly imported from the source, it is self-proving: such elements are opened and closed as they appear in the source. Given that the source is well-formed, there is a guarantee that they will be closed before another element starts. For elements obtained from the content and source markup, the argument is: an element of this kind starts with a sequence of open tags in the source document. Well-formedness in the source ensures that the end of all elements is found; thus, the end of the element created in the target is found when the outer tag in the source is closed. Moreover, as this is the last element open in the target (the last one entering the open elements stack), it will be correctly closed.

A detailed version of the algorithm

The version of the algorithm, where complex sentences are expanded, is on pages 35 to 36. Here, concepts such as *keeping* a variable mean that data structures are used.

The main algorithm loop continues until the end of the input document is found. At this moment, the output document is closed in a correct manner (closing all input tags) and the algorithm ends. The code shows action in each iteration.

Two data structures are used to keep the memory in evolution. A stack ORIGEN tracks what has been done with open tags in the source. This is useful to know what to do with matching closing tags. If the open tag was ignored, it will be ignored. If the matching open tag was translated -because it was associated to a presentation element-, the closing tag will also be closed in the target document.

A stack ABIERTOS keeps track of open elements in the target document. This stack can be inspected when a semantic element is to be opened, to know if there are elements to close before opening the current one. Every time a new semantic element starts in the output, all elements of equal or inferior level are popped from the stack and written to the output. Correct nesting in the output is guaranteed by the way of introducing and taking elements from the stack.

Algorithm 1 Structure extraction algorithm.
INPUTS: SourceDoc: D, DTDmapping: V, DTDhierarchy: H
OUTPUTS: TargetDoc: STR

```
while there are inputs from SourceDoc do
switch input do
 BEGINDOC:
   open(target)
   write-header(target)
   searching-keyword ← false
 ENDDOC:
   while not empty(ABIERTOS) do
     ele ← pop(ABIERTOS)
     close(ele)
   end while
 STARTTAG:
   push(ORIGEN, 'candidata')
 ENDTAG:
   if top(ORIGEN) = 'candidata' then {empty element}
     ele ← pop(ORIGEN) {ignore empty elements}
   else
     if top(ORIGEN) ≠ 'ignorada' then {it closes a formatting element}
       close(pop(ABIERTOS)) {close the element in the output}
     else {top(ORIGEN) = 'ignorada'}
       ele ← pop(ORIGEN) {ignore the closing tag}
       if top(ORIGEN) ≠ 'ignorada' then {it marks the end of a semi-semantic (title) element}
         write(pop(ABIERTOS))
       end if
     end if
   end if
 TEXT:
   if searching-keyword then
     s ← concat(keycandidate,text)
   else
     s ← text
   end if
   if startkeyword(s) then
     searching-keyword ← true
     split(s,s1,keycandidate) {keycandidate = start of some keyword}
     if s1 is not null then
       write(s1) {it is certain that the text fragment preceeding the possible start of the semantic element
       belongs to a previous element}
     end if
   end if
   if completekeyword(s) then
     split(s,s1,keycandidate)
     if s1 then
       write(s1) {fragment text before the candidate start of semantic element belongs to some other
       element with certainty}
     end if{replace the top sequence of 'candidata' in ORIGEN by 'ignorada'}
```

```
    while top(ORIGEN)='candidata' do
      ele ← pop(ORIGEN)
      push(AUXILIAR, 'ignorada')
    end while
    while not(empty(AUXILIAR) do
      push(ORIGEN, pop(AUXILIAR))
    end while{candidate tags have been ignored}
    outputtag ← eqtag(keycandidate,DTDmapping) {obtain the equivalent tag from the vocabulary mapping}
    if in(outputtag,ABIERTOS) then
      while top(ABIERTOS) ≤ outputtag do {close hierarchically lower or equal elements}
        ele ← pop(ABIERTOS)
        close(ele)
      end while
    end if
    open(outputtag)
    push(ABIERTOS, outputtag)
    write(keycandidate)
    searching-keyword ← false
    keycandidate ← null
  end if
  if notkeyword(s) then
    if top(ORIGEN)='candidata' then {start of a formatting element}
      while top(ORIGEN) = 'candidata' do
        push(AUXILIAR, pop(ORIGEN))
      end while
      while not(empty(AUXILIAR)) do
        ele ← pop(AUXILIAR)
        push(ABIERTOS, ele.tag)
        open(ele.tag)
      end while{candidate tags are open in the target document in an ordered manner: text can be written}
      write(s)
      searching-keyword ← false
      keycandidate ← null
    else {in the middle of the text of some element}
      write(s)
    end if
  end if

 end switch
end while
```

Algorithm evolution on an example

The document in figure 2.12 is a very simple document extracted from a document in the prototype that will be presented in chapter 5, that has been input to the algorithm (the example has been simplified to be included here). The evolution of the creation of the target document copy and stacks is shown. Every state shows the output document, the content of the stack that keeps track of the tags open in the output (ABIERTOS) and the content of the stack that keeps tags found in the input until document content allows a decision to be taken to ignore or to transfer them (ORIGEN).

Transitions through states are labelled with the input event and the associated string received from the parser. Some transitions have a note on the right (in bold italic) to

indicate decision factors that complement information in the input event and text. The evolution can be followed in annexe A.

```
<doc>
<p>Ley 1.</p>
<h4><a>CAPÍTULO I.</a> DEL REFERENDUM.</h4>
<p><a>Artículo Primero.</a></p>
<p> Texto del artículo primero.</p>
<p><a>Artículo Segundo.</a></p>
<p>Texto del artículo segundo, previo a una disposición</p>
<p><b>Disposición final.</b></p>
<p>Texto de esta disposición.</p>
</doc>
```

Figure 2.12: Source document for the algorithm evolution example.

2.5 Application of the content-based semantic structure extraction to legal documents

Documents in the digital library used in this prototype are Spanish rules, jurisprudence, and other documents that complete this information. The grammars of these classes can be found in the chapter that describes the prototype (chapter 5). The extraction algorithm is used here to obtain the semantic representative of official documents, coming from public Web servers. The obtained document representative are XML documents, where the structure (and, therefore, the DTD) agrees with the semantic structure of the documents. The example in subsection 2.4.3 shows the application of the algorithm to a Spanish rule. The grammar for Spanish rules is in figure 2.4 and the containment hierarchy derived from this grammar is the one in figure 2.10.

2.6 Discussion

The majority of work related with documents logical structure is concerned with discovering the grammar associated to a class of documents. This is a different problem from the one of discovering the logical structure of an instance of the class. Two works that can be included in this last category are those from Smith and Lopez [102] and Lim and Ng. [83], commented in subsection 2.2.2. The utility proposed in [83] discovers the containment hierarchy in HTML elements. Smith and Lopez use content cues to infer the existence of "concepts" inside a structured document; these concepts represent semantic fragments in the document. The important goal in this case is to discover the concepts, as opposed to Lim and Ng., who already know what the elements are and focus on the hierarchy.

The algorithm presented in this chapter extracts the semantic logical structure of

a document instance, using the target DTD to do so; it is not the discovery of a class grammar. The knowledge of the input document copy is that it has a logical structure, that nesting levels in markup are correct, and that there are content keywords that signal the start of semantic elements. There is no knowledge of the types of elements in the input: if they can be ignored, they are; and when they cannot be ignored they are simply transmitted to the output with no further analysis of these tags. This is a difference from [83], where knowledge of the input DTD guides the algorithm evolution. On the other hand, the semantic element limits in this algorithm are determined by the start of a new semantic element, which can be done on the basis of the presence of some vocabulary in the content. The vocabulary is well-defined and the abstract documents treated are well-structured: semantic element limits are accurately determined by the presence of this vocabulary. This determines a completely different way of working from [102]: first, there is no knowledge of divisions in the input document copy (as in [83]), and second, there is no need to explore inside input regions that have a good guiding vocabulary.

The algorithm in section 2.4 simulates the way a human being recognises the structure in this type of documents: during a sequential reading, a document part ends where another part begins. It aims to extract the semantic logical structure of a document instance. The input is a tagged document. No attempt is made to obtain the grammar of a class, as there is one already. Neither is it the aim of this thesis to transform DTDs; in this case,the input DTD should be available, and that is not the case either. The advantage of the proposal here is that knowledge of the input document structure is minimal: basically, that it is a well-nested tagged document. The algorithm profits from knowledge of the abstract document class general structure to guide the extraction of the instance logical structure: the output grammar inclusion rules serve to determine the inclusion hierarchy between semantic elements, and the vocabulary mapping of the document class allows their limits to be recognised.

One of the advantages of having a semantically tagged document copy is to be able to dissociate the presentation from the document content, associating a stylesheet to the document class that will be applied to the document at the very last moment of interaction with the user (*Interface* service in subsection 4.2.3 of chapter 4).

It is not possible to obtain a general algorithm that extracts the semantic structure of any document. Lack of knowledge of tagging and inclusion hierarchy in document copies input to the algorithm forces some decisions to be taken that restrict the algorithm's generality; decisions taken have been influenced by the prototype domain and the origin of documents used in the prototype. The fact that the initial data to the prototype implementation were HTML pages from different servers was considered at several points. Semi-formatting elements are the consequence of realising that semantic element titles were always formatted differently from the rest of the element. The nesting of semantic elements has not been considered. There were two exclusive possibilities:

- To allow it: to recognise semantic elements that start inside another one, with no sorrounding tagging; this would suppose a risk of confusing a citation of another document element with the start of a semantic element inside the current document, which would be an error.

- Not to recognise semantic elements that start inside an element content (not tagged around).

Documents in the prototype frequently use the same vocabulary to cite a document element and to mark the start of a document element inside the current document; this fact was decisive for choosing the first possibility.

Another consideration is that part of the format tagging in input documents should also appear in the output documents. This is the case of paragraph subdivisions. This means that the prior elimination of all input document markup cannot be considered before applying the algorithm. In that case, the paragraphing divisions would have been lost, as they are not semantic. Moreover, this level of division is frequently used in citations (indeed, it is widely used in legal documents). Tables and figures -that are not considered in this thesis, but should be in future work- are also the same case as paragraphs; their divisioning cannot be extracted by text semantic recognition algorithms, which makes them better for preserving as they are in the inputs.

Possibilities to expand the algorithm's range of application are in documents with more flexible language variations to mark the start of semantic elements and in documents not tagged at all. An increase in language flexibility means a need for an elaborated analysis of document content. These processes should be used at the moment the *vocabulary* mapping is done, and should adapt, as is already the case, to the target document class vocabulary peculiarities.

The case of documents with no tagging is a simplification with respect to documents now considered: it is enough to recognise the start of elements, without worrying about preserving tags or not. Considerations to be made about it are the same as those explained some paragraphs before when justifying the decision taken not to consider an elimination of all markup in input documents before application of the algorithm.

The main advantage of the proposed algorithm is that it allows a document digital copy to be obtained that is "formatted" in such a way that its logical structure is an exact digital copy of the abstract document logical structure. The semantics inherent in the abstract document entity is also in the digital copy. But preserving semantics is not only important for itself. The semantic structure is used in citations; therefore, it is extremely beneficial to have this structure available in the digital copy to address the internal document fragments cited, which, in some cases, are also the fragments affected by the modifications presented in the next chapter.

3

Relationships between documents

Contents

This chapter deals with relationships between documents and how these relations can be exploited to obtain advanced digital library functionalities. The objectives of the chapter are: to show how it is possible to model citations and modifications between documents as links, and to show how to exploit these relations. Two ways to exploit them are devised: to query the links in order to ask user questions about relations, and to generate document versions due to modifications that update document content.

Relationships between documents can have varied causes and meanings: they can be semantic and/or, for example, they can be explicit links. The relationships considered in this thesis derive from references in documents to other documents. In all cases, relationships can be represented by a link graph, where the resources are the related items and the links represent the relationships. If there are heterogeneous relationships, links can have *type* that represent the nature of the relationship [108]. The more general use of this graph has been the creation of hypertext that users can navigate through [39]. But the graph can be used with more purposes; for example, relationships can be queried [49], as is done in this thesis. Moreover, when working with structured documents, resources in the link can be document fragments, which allows relationships to be represented accurately to indicate solely the fragment that is really affected by the relationship.

Besides just querying relationships, there is another interesting problem related to working with documents: *versions* of documents. Document versions are variations of the same abstract document, which may affect the document logical structure (this concept was presented in chapter 2). Versions of the same document are obviously related [93, 121], which leads to the problem of expressing these relationships and maintaining them [34, 38, 101]. In this thesis the problem is considered in a different manner: instead of maintaining versions, or discovering the relationships between them (comparing documents to find if they are the same copy of a document) [36], they are generated. The idea is that versions are due, in many cases, to modifications made to previous versions (they are called *historical versions*), which in the end is just another type of relationship. So, if it is possible to query relationships and to work with the associated link graph, it must be possible to generate versions with a traversal of this graph. The algorithm that does this traversal, generating new versions, is presented in section 3.7.

Finally, the manner to store the link graph is presented: a link database (which is one of the architectural components presented in chapter 4). Associated XML standards, XLink [119] and XPointer [116], provide the tools to model a link graph with flexibility: multidirectionality, the degree of the graph, etc. and also seems the most convenient way to do so when documents are modelled with XML, as this continues to guarantee the interoperability inherent in XML data using links.

3.1 Classes of relationships between documents in digital libraries

Documents may be related for many reasons: two or more documents may be created by the same author, they may have the same subject, a document may cite another document or documents, a document contains an explicit link to another document,

etc. These relationships constitute an important piece of information that may in some cases be as important as the documents themselves [52].

Taxonomies to classify relationships are various, depending on the author's interest [108]. Most of them come from the area of hypertext [92, 47, 4]. From this work point of view, it is interesting to differentiate relationships because of the nature of the relationship and how it can be detected. Thus, two or more documents can be related:

- Because there are *semantic* relationships between them. This is the case when documents share some metainformation (author, subject, etc.), or when they are included in the same catalogues, with keyword linking [68], etc. This type of relationship may not be evident even for humans (for example, the criteria used during cataloguing may be as various and complicated as cataloguers themselves) and discovering them can be an arduous process, usually based on some kind of document classification process [67, 68, 108, 71, 46, 23, 21].

- Because there are *references* in a document to another document (the first one *cites* the second one). The relationship is clear for a human being -who directly detects it during document reading-, but not for software applications, which -at least at the moment- are not able to process such references correctly. They can be as difficult to detect as natural language expressions can be complex. This kind of relationships has been studied mostly in the *Reference linking* domain [67, 68, 31, 20], an area that deals with links between documents derived from citations between them, mostly citation linking between electronic journals [31, 68]; these links improve electronic journals, for example, by sharing objects (such as figures) from different sources. Another good example of documents where citations are very frequent and may be the crucial information to access documents whose reading allows a correct interpretation of the citing text are legal documents [121]; in this environment, it is impossible to understand a (tribunal) sentence if the text of the rule (or rules) and jurisprudence that justify decisions in the sentence is not available. Citations give the clues to obtain them and complete the reader information, who obtains a semantically *complete* document by assembling all document contents (his/her information about the sentence is complete).

- Because there are *explicit* links from one document to another, embedded in the document by its author during document creation. This is the case of HTML link tags, that are included inside the document. They may represent semantic relationships, but the difference with the first group is that here the document comes with relationships explicited inside, in contrast to previous cases, where it is necessary to analyse the document to detect the existence of such relationships. The semantics of the relationship that the author took into consideration when inserting the link has been lost, thus only the information about the fact that the two documents are related is all that can be seen from the document, but not *why* they are related.

There is a fourth group of relationships that completes this classification: structural relationships [108]. They relate pieces of content whose aggregation results in a structured or composite document [109, 68]. The aggregated pieces can be complete documents or elements taken from structured documents; in this last case the relationship

is the containment hierarchy reflected in the document tree (see section 2.2 of chapter 2).

All these relationships can be exploited in several ways. One of the most popular is the creation of navigational hypertext, where related documents or objects are linked to make a new document that can be read in a discontinual manner [39, 26].

3.2 Links

Relationships between documents can be modeled as *links*. The most popular links between documents are those present in hypertext, created for navigational purposes: the user starts his/her navigation in a document, from which he/she can pass to other document following links in the current one, and this process can go on following links in documents. In hypertext, a link has the following properties [108]:

- It has (specifies) a *source* and *destination*.
- It is used to activate (specify) a navigation action at the source which consequently reaches the target.
- It represents some relationship (semantics) between the source and destination.

3.2.1 Link graphs

The set of relationships between documents modeled as links results in a *link graph*, where the links can be directed, and also *typed*[1]. Vertices in the link graph are the documents linked and the edges or arcs are the links [39].

Citation and explicit links are *directed*: they have an *origin* (the document that cites or holds the link) and a *target* (the document cited or pointed to by the link). Semantic links can be directed or not, depending on the nature of the relationship. Vertices in the matching graph are connected by arcs that go from the origin of the link to the target of the link. The type of a link is the *label* of the associated arc in the graph. Hypertext graphs [39] are the most popular (there is an example in figure 3.1).

Selecting a subset of links of a certain type results in a *partial graph* with the same nodes and only a subset of the original arcs. The degree of a link vertex is the degree of the associated node in the graph: the number of arcs (links) that have the node as an endpoint.

Example 11. Revision links [93, 47] can be arcs from a document to its revision (version), or from the revision to the original document; directed arcs between the two vertex documents show simultaneously the relationship and the time ordering sequence of document versions (time ordering is implicit in the direction of the typed arc). □

[1] The *type* of a link is a label that indicates the nature of the relation between the linked nodes.

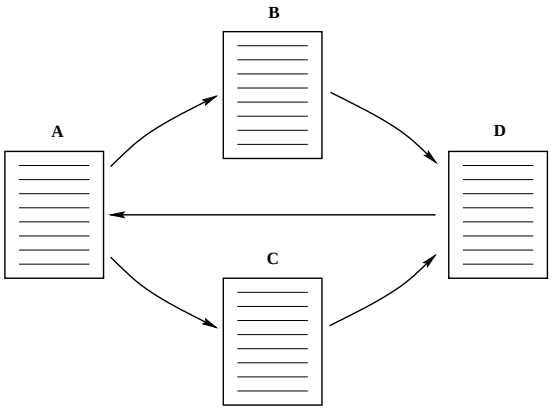


Figure 3.1: An example of a navigational graph in hypertext. Document A links to documents B and C. These two documents have links to D, which links in turn to A. A navigation starting at A could traverse B (or C), pass through D and finish by returning to A.

Linking structured documents: granularity in the link graph

Structured documents introduce a vision of a document that is not the vision of single, indivisible units; a structured document is a set of nodes (document fragments) that are structured in an inclusion hierarchy represented by a tree of nodes (see chapter 2). Structured documents have the advantage that linking internal document fragments is facilitated by the availability of the document structure.

This vision of the document gives rise to a link graph with more granularity than when referring just to a complete document [53, 29] -as the number of nodes in the graph is greater than in one where the documents are considered to be the minimun unit-, and where the hierarchy allows different subsets of the same document tree, made up of the set of nodes that form a document portion, to be selected by using query language expressions [40, 115]. Document trees that represent the inclusion hierarchy in structured documents are partial graphs that contribute to the typed link graph with structural links. Links can be queried [49] and manipulated to obtain composite documents [109, 103]. Moreover, the hierarchy in structured documents can also be queried [1, 2, 49, 50].

Example 12. Annotations are typical examples of links that affect document fragments. For example, annotations made to pieces of theatre are commonly related to some concrete scene or act and not to the whole work (in which case they would not be considered annotations, but comments on the work). □

The following definition for link graphs, to which the algorithms presented thereafter apply, takes into account the considerations about granularity in structured documents.

Definition 3.1 *In the context of structured documents, and restricted to relationships that result in directed links, a* link graph *is a labelled directed graph $G = (N, E)$ composed of two finite sets: a set of nodes N and a set of arcs E. Nodes in N are document fragments. Each arc $u = ((i, j), t) \in E$ is a tuple where:*

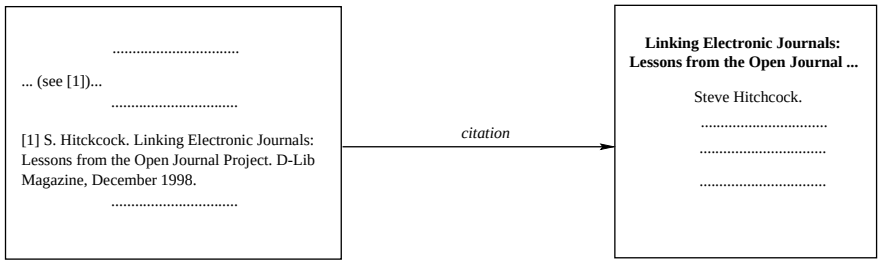


Figure 3.2: Citation linking: document on the left *cites* the one on the right. The direction of the arcs illustrates the direction of citations.

- (i, j) *is an ordered pair of nodes where* i *is the* origin *of the link and* j *is the* target *of the link,*
- t*, the label of the arc, is the* type *of the link.*

Each arc u *corresponds to a typed link.*

Example 13. Citation linking is mostly concerned with linking on-line journal articles. In this domain, bibliographic citations are the most important type of link worth managing. Figure 3.2 shows a simple link graph where each node represents a document. The *arc* between them has the following semantics: two documents are linked if one of them *cites* the second one (thus the *citation* label). The arc is *directed*, meaning that the document pointed to by the arc (the link's *target*) is the one cited, while the other (the link's *origin*) is the one that cites. □

3.3 Linking with standards for structured documents

XML has associated standards that allow relationships between documents to be modelled. Relationships are modelled as links, represented as *XML* documents. These standards have capabilities that supersede traditional hypertext links (namely, HTML links).

Linking with *XML* includes rules to link resources (XLink) and to address internal fragments inside linking resources (XPointer)[2].

The XML Linking Language (*XLink*) [119] allows elements to be inserted into XML documents to create and describe links between resources. In *XLink* , a *link* is a relationship between two or more resources or portions of resources, made explicit by an *XLink* linking element. This does not map with the notion of *arc* in the linking graph introduced in section 3.2. Indeed, an "XML link" (*xlink*) is the union of one or several arcs that have "something" (the semantics of the relationship) in common.

XLink allows traditional unidirectional links embedded inside a document, but also more complex links. This means that with *XLink* it is possible:

[2]They are not yet stable *W3C* recommendations.

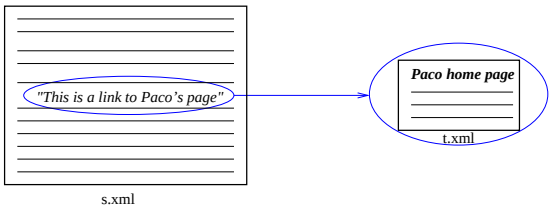


Figure 3.3: A simple link.

- To assert linking relationships between more than two resources (n-ary links)
- To associate metadata with a link
- To create link databases that reside in a location separate from the linked resources.

Xlinks are a means to model graph information: vertices are resources (documents, images, etc.), arcs in the graph are arcs inside an xlink, arc labels are modelled with `role` attributes and other metadata about arcs or vertices can be joined as (resources or arc) attributes.

3.3.1 XLink

The version of *XLink* referred to in this section is the one used for the implementation of this thesis prototype: the working draft is dated February, 21st, 2000 [119].

There are two main types of xlinks: *simple* links and *extended* links. **Simple** links offer a short form for a common kind of link: the two-ended inline link. The source link vertex is in the document where the link resides, while the target vertex is a remote resource. They are similar to the well-known HTML link elements (`a` elements).

Example 14. The simple link below corresponds to the link in figure 3.3. The source link content is the content of the linking element, and the link target is designated by the `xlink:href` attribute value.

```
<simplelink xlink:type="simple" xlink:href="http://www.bla.bla/paco.html">
            This is a link to Paco's page
</simplelink>
```

□

Extended links offer full *XLink* functionality, such as out-of-line, multidirectional links and links that have more than two participating resources (n-ary links). They can be inline or out-of-line. They are the only ones that can be used with n-ary links and to link resources found in external documents (for example, resources found in documents that cannot be written to include the inline link). Extended links are the most interesting ones, as they allow linking graph structures to be expressed in as complex a manner as desired (which cannot be done with simple links or HTML links) and to traverse the link in whatever direction is desired in each link resolution.

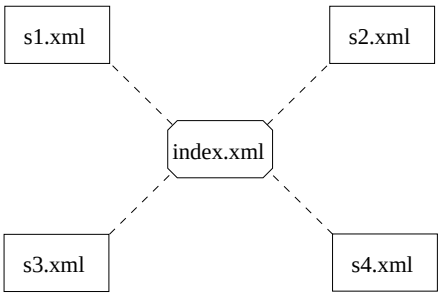


Figure 3.4: An extended link.

Xlink element attributes can come from a set of attributes specified in the standard or they may be created by link authors. *XLink* provides attributes to indicate the `type` of an xlink element, its `role` in the relationship and to address participating resources (`href`)[3]. For extended links -that can associate N resources and an arbitrary number of arcs (see figure 3.4)-, there is a special type of element to represent an arc between two given nodes. Elements inside an extended link are merely the nodes of the graph; arcs between these elements have to be represented to completely describe the links graph. These arcs are represented with elements that present the value *arc* in the *type* attribute, thus indicating that the element content is not a node but an arc.

3.3.2 Addressing internal document fragments: XPointer, XPath.

While *XLink* addresses documents, XPointer [116] addresses internal fragments (a point, a set of nodes, or an interval) of XML documents. That is, internal document locators for portions of documents are constructed with XPointer. Inside an `xlink:href` attribute, the XPointer is the argument to the `xpointer()` function, always placed after the document URI. The character '#' marks the separation between the document URI and the XPointer.

XPointer is built on top of the XML Path Language (XPath). Many XPointers are *location paths*, built from location steps. Each location step specifies a point in the targeted document, generally relative to some other point, such as the start of the document or another location step. This reference point is called the *context node*. In general, a location step has three parts: the axis, the node test, and an optional predicate.

axis::node-test[predicate]

The *axis* tells us in what direction to search from the context node. The *node-test* tells us which nodes to consider along the axis. The *predicate* is a boolean expression that tests each node in the node-set. Only nodes that comply with the three conditions at the same time are selected.

[3]More attributes can be found in the XLink specification.

Example 15. The addressing of document elements can be done with XPath location paths, as in the example in figure 3.5, where the XPointer addresses the first element `articulo` inside the first element `disposicion` of document `l13-1986.xml`. There are two location steps in this example:

1. `descendant::disposicion[1]`

   The `axis` in the example selects all descendant nodes of the context node. The node-set indicates that only descendants of type `disposicion` should be considered. The predicate tells us to select the first element `disposicion` from all descendants.

2. `articulo[1]`

   In this location step there is no axis; by default, all children of the current context node (the first element `disposicion` from all descendants) are considered. Among them, elements of type `articulo` are considered. Again, the first one is selected.

□

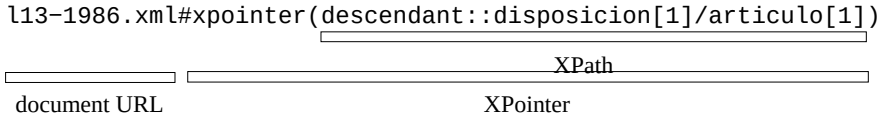


Figure 3.5: Value of an *xlink:href* attribute. Internal locator is an XPointer that has an XPath inside. The path addresses the first element *articulo* inside the first element *disposicion* of the document *l13-1986.xml*.

3.4 Document versions

Document versions are presented as follows: given an abstract document, D, there can be several versions of the document, corresponding to modifications applied to D. All versions of D have a high degree of content similarity, but never one hundred per cent similarity. The document version problem could be stated as follows: "given an abstract document D and the collection of versions of D (also abstract entities), is it possible to access any version of D, allowing at the same time for users to ignore (or not) the existence of several versions of D depending on their preferences?".

DeRose [47] distinguishes two kinds of versioning:

- *Historical versions*, which correspond to modifications to document content in time. Historical versions may also suffer variations in the logical structure, but share a significant percentage of document content (and logical structure).
- *Translation versions*, that share the logical structure, but have significantly different content.

Historical versions pose some interesting questions: Is it suitable to keep all versions? How to model the modifications? How to detect them? The answer to these questions

depends greatly on the objective of the system implemented in the context in which it works. Most of the effort related to document versioning is concerned with the simultaneous maintenance of versions in the document database. Three approaches have been used for this purpose:

1. To link related versions. These links were named *revision links* by Parunak in 1990 [93]. Two databases are kept simultaneously: the document database and the link database. This approach has been followed by Wilson [121] and Choquette et al. [38]. The main problem with this approach is to keep the revision links database [34, 38, 101] up to date.

2. To consider different stamps of the database and to compare them in order to detect changes that reflect the fact that an object has been versioned [34]. This solution is used with object databases, and therefore can be considered when modelling documents as objects. In this approach the link database disappears and document changes are represented indirectly as the difference between two database states [36]. It is applicable to historical versions, but not to translation versions.

3. A third approach comes from the area of semistructured data. Chawathe et al. [36, 35] model changes to hierarchically structured data (which is the case of structured documents) as changes to nodes in the document tree. They represent changes as annotations (attributes) to the affected nodes, facilitating queries about its "history". The detection of versions is done by tree comparisons. In contrast to the previous ones, this is the first solution where document structure is considered, thereby associating changes to document fragments instead of to whole documents.

   This idea of annotating document nodes with attributes that contain information about these changes can be found in some public servers [79], where document elements are qualified with attributes that indicate they have been modified later.

3.5 Linking and versioning in the legal domain

Legal information has some special characteristics that makes it well-suited for illustrating the importance that linking between documents may have. It is also a domain where document versioning emerges as a crucial problem; good access to document versions facilitates specialists' work, while no access to these versions can be the obstacle that makes such work impossible.

3.5.1 Relationships

Legal documents are intensely related. Besides semantic relationships (documents of the same category, with the same court provenance, etc.), there are many citations between documents. Any document may contain several citations to previous rules, jurisprudence, etc. and, in the other direction, it can be referenced in many other documents. For example, some rules are indeed a collection of amendments to previous rules; every amendment is a reference to the modified rule.

Access to the information that provides these relationships can be valuable for any lawyer [121], for example during the preparation of a case. To interpret a tribunal sentence it is necessary, in addition, to have the text of the rules that guided the decision cited in the sentence. It is worth noting that exploiting these relationships in both senses is a requisite: it is as important to know about the jurisprudence related to a certain rule as to know which rules are related to a given sentence. That is, it is a requisite for the legal information graph to be traversable in both path directions.

Modelling relationships

The evolution of the manner of modelling relationships for legal documents is parallel to the evolution of standards related to structured documents. Solutions prior to the emergence of XML kept the content, structure and relationships in separate databases. Document structure was modelled as *structural links* [3, 38]. Solutions based on SGML or XML do not need to look at structural links, which are implicit in the document tagging. Most of these solutions choose to model relationships within one of the documents involved [65, 56]; this choice is influenced, if not in all in most cases, by the use of HTML hypertext, where links are explicitly typed [79].

Hypertext has been generally accepted as an ideal model for these documents [3, 62], where relationships are represented by hyperlinks. The user navigates through these hyperlinks to obtain related documents in each navigation step.

3.5.2 Versioning

Another particularity of legal documents is that these documents (for example, rules) are the object of several partial historical modifications. That is, a given rule can be the target of a temporal sequence of amendments that result in different versions of the document valid during the period of time between two modifications. Once more, to be able to access the version of a document as it was at a certain moment is crucial for analysing some other documents [121]. For example, to analyse a sentence, it is necessary to read the rules that justify it as they were at the moment the sentence was made; niether previous versions of the rule, nor later versions can be used.

The document versioning problem has been treated in legislative digital libraries in three ways:

1. Maintaining simultaneously all versions as individual documents in system databases [38, 101]. Versions of the same document are linked by revision links. Its main difficulty is to keep these links up to date, as well as links that affect documents (which may, in consequence, affect all the versions of the involved document).

2. Modelling modifications as attributes. This solution is, however, compatible with the previous solution. These attributes are considered as "links", whose resolution (obtaining the link's target) is left to the user [56]. Its main aim is to facilitate queries about the "history" of a document (changes made to it): the user can know the document has been changed and where to find the changes; however, it is up to the user to obtain the versions if this is his/her wish.

3. Keeping the rules that allow the generation of versions of documents [13]. For every version there are associated rules that allow it to be generated automatically at the user's request. This approach coincides with the one in this thesis as far as the goal is concerned, but differs in the way versions are obtained; a particularised set of rules for each version has the advantage of its precision and efficiency, but the disadvantage that every version's set of rules has to be specified independently. Versions cannot be generated unless the rules have already been specified, even if all the necessary pieces are in the library. A more general method, such as the one presented in section 3.7, to automatically infer the rules for generation of new versions allows this to be done, taking advantage of the relationship graph.

3.6 Modelling of citations and modifications with typed links

3.6.1 The relationships modelled

There are two types of relationships between documents that are dealt with in this thesis: those that do not modify any of the documents that participate in the relationship, and those that do modify some of the participating documents. A precision follows:

- *Citations* between documents are a type of relationship due to the fact that a document "cites" another. This type of relation between two documents does not modify any of the linked documents.
- *Modifications* between documents relate a document that is *modified* with a second document that contains the modification. The application of modifications to a document result in a new version of the document. Modifications to a document can consist of additions of new text, substitutions of some text by another text or the deletion of some content. It can all be reduced to substitutions: an addition is the replacement of an empty text fragment by the text to be added, a deletion is the replacement of a non-empty text fragment by an empty text. They will all be referred to from now on without distinction as "modifications".

Citation relationships can be recognised by the presence in a part of the document text of a citation or reference in the text of a document `A` to some portion of some other document `B`. `A` and `B` are related documents. The reference can be merely a citation in `A` to some portion of `B`, or a reference to some portion of `B` indicating how to *modify* the portion referenced. These cases where modifications are extracted from document content have the property that modifications are always "geographically" close to a citation that designates the target of the modification that follows. There are two heterogeneous links (a citation and a modification) that share the target, but which have different origins.

Moreover, the citation to a document element implies that the element and all subelements included inside it are affected by the relationship (for example, a citation to

the first section of this chapter includes all subelements of the first section: paragraphs, figures, tables, ...). Access to document fragments is a necessity for the link graph obtained from these relationships to have the same precision (granularity) as in the citation texts. In the link graph, the implied documents (or fragments) are the nodes, while relationships between them are typed links. The graph structure is explained in the next subsection.

3.6.2 The resulting link graph

Links

A *link* is a directed arc from an **origin** (or **source**) vertex to a **target** vertex. A *typed link* is a labelled directed arc from an origin vertex to a target vertex. The origin is the document (or document fragment) that contains the citation or modification, and the target is the document (or fragment) which is cited or modified. An example can be seen in figure 3.6. The link type expresses the nature of the relationship (citation, modification) and is represented in the arc label.

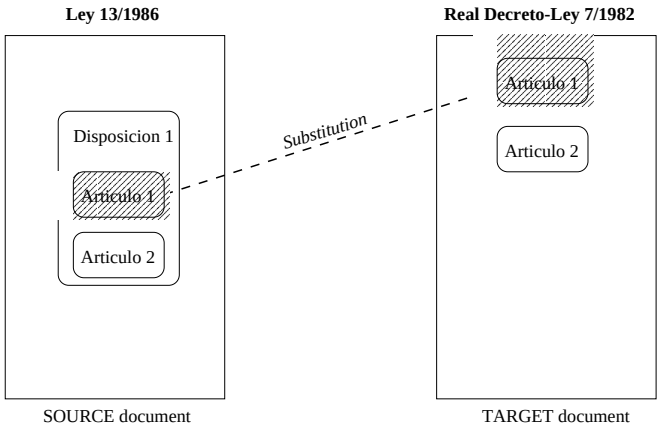


Figure 3.6: An example of typed link. The source document (*Ley 13/1986*) has an (*disposición 1/artículo 1*) element that replaces the first *artículo* element in the target document *Real Decreto-Ley 7/1982*. Link vertices are document fragments (elements of type *artículo*) instead of complete documents.

With this consideration, a *typed link* is considered a tuple $< v_o, v_t, t >$ where:

- v_o is the node set at the origin of the link
- v_t is the target node set of the link
- t is the type of the link

Link vertices

The relationships considered (citations and modifications) are more granular than relating two documents. This is specially important with modifications, where the relationship, in general, only affects a document fragment and not the document as a whole (the example in figure 3.6 relates document fragments).

In structured documents, the logical structure reflects the division of the document in fragments; the portion of the document inside which a citation is located can be referred to by its position relative to the logical document structure, and the same statement can be made for the cited (or modified) document fragment.

However, it is also true that, taking the document logical structure into consideration, the majority of links involve several nodes in every vertex of the link. Citations and modifications may affect one or more nodes in the document tree, thus affecting a node-set. For example, a citation to a definition can involve the `definition` element itself and all the paragraphs that together form the definition. There is a mixed vision of the document as a tree of nodes related by hierarchical relationships and a set of nodes. In this way, link vertices are node-sets that can be located by their position in the document tree. In general, node sets can be represented by the root node in the subtree formed by each node set.

The graph

Combining document trees with the set of typed links between document fragments (node sets) -with the consideration that node sets are subtrees that are unambiguously determined by the position of their root node- result in a *labelled* graph G where the vertices are document nodes, and arcs are labelled depending on the semantics of the relationship between them:

- Arcs from the document tree represent the hierarchical relationships between nodes inherent to the document logical structure. These are *structural* arcs and are not labelled in the examples shown below, but are drown in bold.
- Arcs that represent a citation in the origin to the target have the label *citation*. They represent citation links (they are drawn with dash-dotted lines in figures).
- Arcs associated to a modification link are labelled *modification*.

Graph cardinality

Vertices can be simultaneously the source and target of several links. It is normal that a document participates in several heterogeneous links -either because it contains modifications to several documents or different fragments of a document, or because it is cited from several points, or it is the target of various modifications, etc-, and that these links relate it to various other documents. This is also true for document fragments: a document fragment can be cited several times or it can be modified from several sources. Figure 3.7 shows a document (or fragment) `A` that participates in three heterogeneous links, being the target of the three: `B` and `C` are supposed to replace `A` in new versions, and `D` merely cites it. Figure 3.18 shows another example of multiple cardinality, where the target node-set is affected by various modification links. This figure corresponds to a portion of the graph shown in figure 3.8.

Example 16. Articles from the most important Spanish rule ('*La Constitución*') are cited in hundreds of documents. They are the target of as many citation relationships as citations to them can be found. □

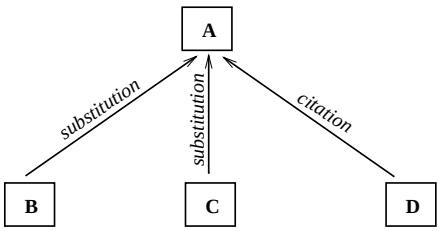


Figure 3.7: A link graph with 3 heterogeneous links. One of the vertices, A, is endpoint of three different links.

Example 17. Figure 3.8 shows a graph with all three types of relationships. There are four documents with root in D, P, M and N respectively. The tree structure corresponds to structural links (not labelled). The subtree of D with root in d_2 is cited (citation link c_1) in M and modified by the subtree with root in m_2 (modification link m_1). It is also cited in P (citation link c_3) and modified (link m_3). Element d_3 in D is cited (citation link c_2) inside p_2 content. The subelement d_21 of d_2 in D is also the target of a modification (m_2) and citation (c_2) that only affects this node. d_{21} is affected by its own modification (m_2) and by modifications m_1 and m_3. □

Partial graphs

Three partial graphs can be obtained (see figure 3.9), depending on the type of link (operation) considered:

- The forest formed by document trees, where hierarchical relationships between document fragments (logical structure) are represented. Paths in these trees allow individual fragments (node-sets) to be addressed. This is the *structural graph*.
- The *citations graph* obtained by considering node-sets and relationships that do not modify any of them. This graph can be used for navigational purposes.
- The *modifications graph* obtained by considering document nodes, structural arcs (arcs in the document tree) and modification arcs.

  A modification to a document fragment can be expressed in terms of a reference to the root node of the affected fragment. Considering the hierarchical relationship between nodes expressed in the document tree, it can be said that a change to a node is indeed a change to the element and all its descendants (including the leaves that hold the document content). This property is used in the version generation algorithm explained in subsection 3.7.3, that extracts the subgraph needed for each document version generation from the modification graph.

3.7 A proposal to generate document versions using links

This section presents a proposal to generate historical document versions, using the modifications graph obtained when modelling modifications together with document

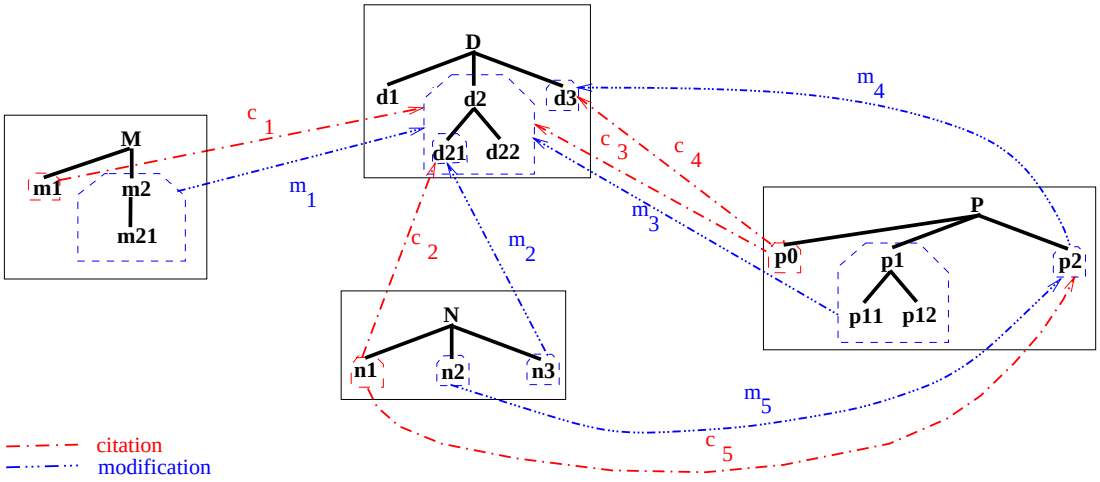
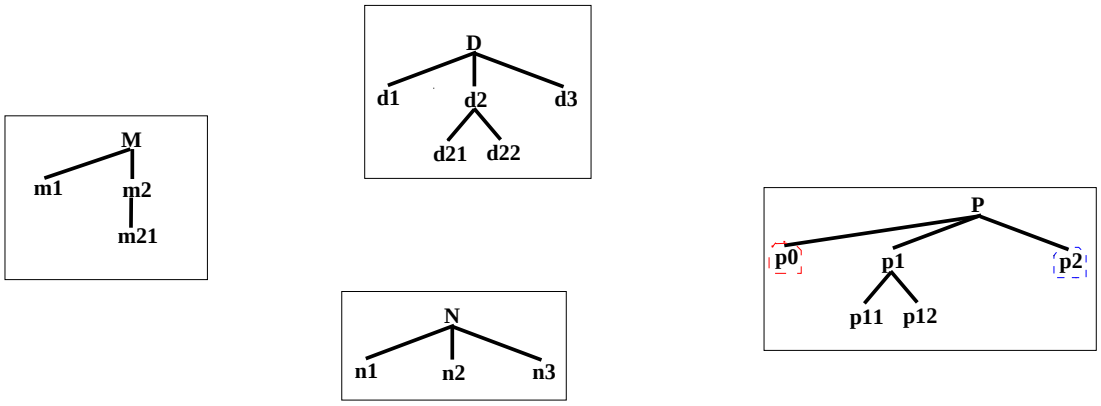
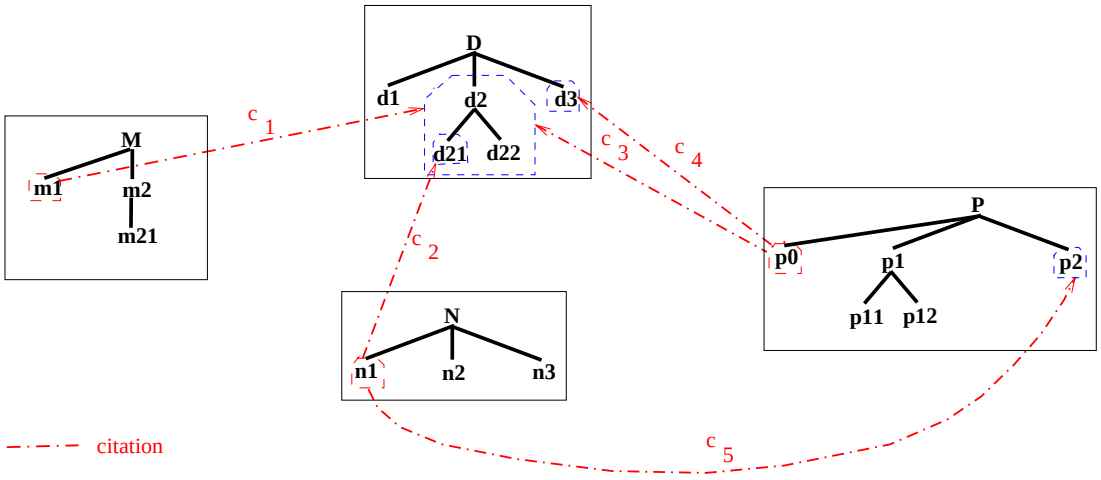


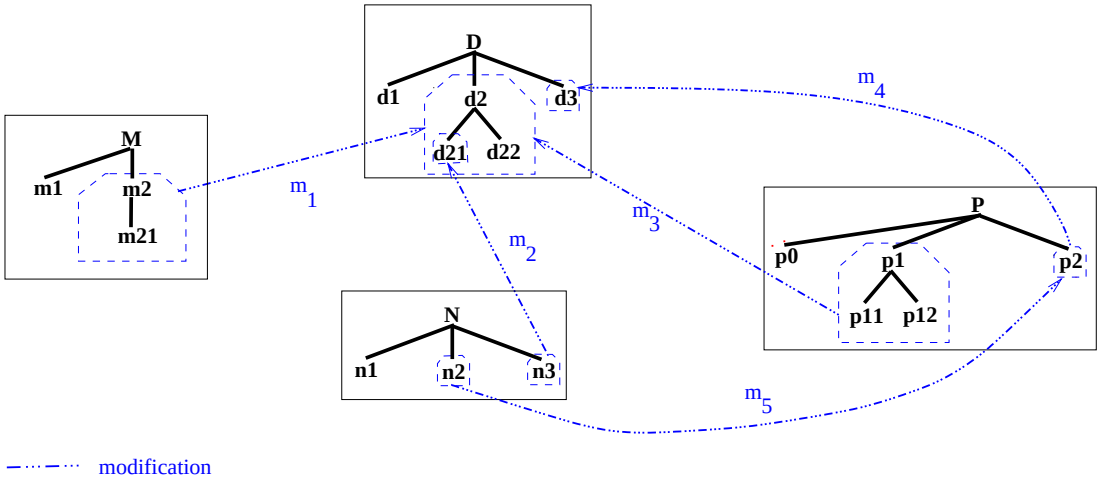
Figure 3.8: Link graph with heterogeneous links: structural, citation, modification.



(a) Structural graph.



(b) Citation graph.



(c) Modification graph.

Figure 3.9: Partial graphs obtained selecting links by type criteria from the graph in figure 3.8.

structure as typed links. A subgraph will be used for the generation of each individual document version, called the *versioning graph*.

There are three main aspects related to document versions that can be considered: detecting changes, representing changes and querying changes. Detecting changes can be done by comparing documents [36] or from citations inside documents [121, 68]; the approach used in this thesis is the second one: versions are due to modifications detectable from citations. There are two possibilities to represent changes aside from the one chosen in this thesis: storing all versions caused by a change and linking them [38] and representing changes as annotations [36, 35]. The problems caused by storing all versions have been commented in section 3.4, and moreover, this approach presupposes that someone has taken care of generating the different version. This fact is not always guaranteed, which is the case when modifications that cause different versions to appear come from citations: the documents that should be used to obtain every version are available, but the different versions must be composed by users following the modifications and applying them. As for representing changes as annotations, this facilitates queries about the history of a document, but it is not the most appropiate choice to facilitate version generation. The information about relationships does not appear as a link that can make part of a graph, but as attributes of nodes. In the end, these attributes express a relationship between two nodes. So, the decision in this thesis has been to model this type of relationship as any other type of relationship: with typed links, and to generate versions exploiting the relationship graph.

So, the aim of the proposal presented in this section is to be able to offer a copy of a document in its state at a certain date. Every version of a document should show the resulting document version after applying all modifications made to the document in the time interval from document creation and requested version date to the original version. The presumption is that the initial version of the document is available, as well as documents that hold modifications to it. Also, it is proposed to "store" modifications in a link database that contains the information about the relationships graph that involves all documents needed. Obtaining document versions from modifications should be done on demand, by applying modifications expressed in links to the original versions.

The proposed algorithm -subsection 3.7.3- works on structured documents, in which modifications concern document fragments whose limits can be specified in terms of the document structure. Another characteristic is that modifications are commonly included in some other document, but are not available as complete documents. That is, a document is modified by the replacement of some of its fragments by fragments coming from other documents. With structured documents, it is feasible to have a modifications graph whose vertices are node sets -as explained in section 3.6.2- that allows the version generation to be tackled as a graph problem. Modifications to be applied are filtered from the total set of modifications using request parameters (for historical versions the filter criteria is the date).

From the architectural point of view, the algorithm explained here is implemented as a method of the document management service, and the scenario where it fits is the version generation scenario in chapter 4. Figure 3.10 shows documents that participate in this process. At the user's request, documents and links are combined to create a virtual document: the required version. This one is obtained by applying modifications to a document copy already in the collection: the *source* document, D. Modifications to

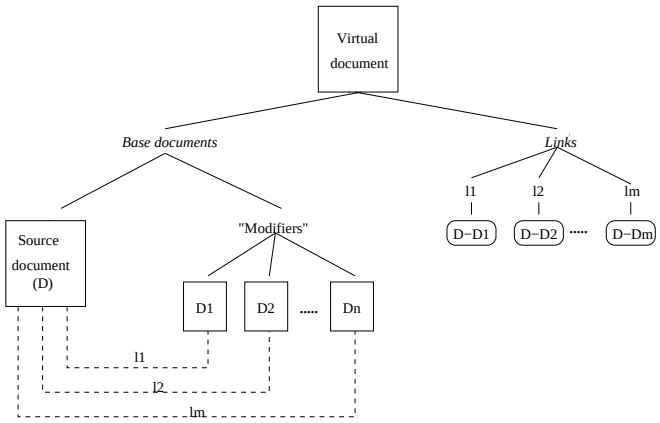


Figure 3.10: Documents in the version generation process. Inputs documents are a document to modify and modifier documents.

D are extracted from some other documents: the *modifiers*.

3.7.1 The output version tree

A document is considered as a tree with elements. The output document, produced as a result of the update process, belongs to the same document class as the original version, but may have a somewhat different logical structure.

The tree is basically the same as the initial one, where some nodes have been replaced, deleted or inserted. Nodes in the source document not affected by any modification (nodes that are not the target of a link) remain untouched in the output version. Structural links and modification links are used to produce the new graph.

Example 18. An example extracted from the legal documents library illustrates the problem.

Input documents to the updating process are the rules shown in figure 3.11:

- A source document, `lo2-1980.xml`, whose first element *articulo* has to be replaced. Its source text is in figure 3.11(a).
- A modifier document, `113-1986.xml`, which contains the element that will replace the first element *articulo* in `lo2-2980.xml`. The replacing element is the first *articulo* inside the first element *disposicion*. The source text 3.11(b).

The output document of the modification process -which can be seen in figure 3.11(c)- is a new version of the source document, where the first node-set that constitutes the element *articulo* has been replaced by the first element *articulo* inside the first element *disposicion* in document `113-1986.xml` and the rest is unchanged. Figure 3.12 shows the effect of the process in the document tree. The link that models the modification can be seen in figure 3.13. □

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<doc>
<articulo id="a1"><title>Artículo Primero. </title>
<p>El referendum en sus distintas modalidades, se celebrará de
acuerdo con las condiciones y procedimientos regulados en la
presente Ley Orgánica.</p>
</articulo>
<articulo id="a2"><title>Artículo Segundo. </title>
<p>Uno. La autorización para la convocatoria de consultas populares
por vía de referendum en cualquiera de sus modalidades, es competencia
exclusiva del Estado.</p>
</articulo>
</doc>
```

(a) Source document for the example: *lo2-1980.xml*

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<doc>
<p>Ley 13/1986, de 14 de Abril de 1986, de Fomento y Coordinación
General de la Investigación Científica y Técnica</p>
<p>Don Juan Carlos I,Rey de España.</p>
<disposicion id="da"><title>DISPOSICIONES ADICIONALES. </title>
<p><a>Undécima.</a>
 1. Quedan modificados los artículos 1.°, 4.° y 8.° de la Ley
Orgánica 2/1980, de 30 de abril , que quedarán redactados en la
forma siguiente :</p>
<articulo id="da111"><title>Artículo 1.</title>
<p>Con la denominación de Instituto de Astrofísica de Canarias se crea
un Consorcio Público de Gestión , cuya finalidad es la investigación
astrofísica .</p>
<p>El Instituto de Astrofísica de Canarias estará integrado por la
Administración del Estado, la Comunidad Autónoma de Canarias la
Universidad de La Laguna y el Consejo Superior de Investigaciones
Científicas .</p>
</articulo>
<articulo id="da112"><title>Artículo 4.</title>
<p>El Consejo Rector estará integrado por el Ministro de Educación y
Ciencia , que actuará como Presidente; un Vocal en representación de la
Administración del Estado, que será nombrado a propuesta del
Ministerio de la Presidencia , y tres Vocales más en representación de
cada una de las restantes Administraciones públicas y Organismos que
se relacionan en el artículo 1.° Formará parte del Consejo Rector,
asimismo, el Director del Instituto , que será miembro nato.</p>
</articulo>
</disposicion>
</doc>
```

(b) Modifier document for the example: *l13-1986.xml*

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<doc>
<articulo id="da111"><title>Artículo 1.</title>
<p>Con la denominación de Instituto de Astrofísica de Canarias se crea
un Consorcio Público de Gestión, cuya finalidad es la investigación
astrofísica .</p>
<p>El Instituto de Astrofísica de Canarias estará integrado por la
Administración del Estado, la Comunidad Autónoma de Canarias la
Universidad de La Laguna y el Consejo Superior de Investigaciones
Científicas .</p>
</articulo>
<articulo id="a2"><title>Artículo Segundo. </title>
<p>Uno. La autorización para la convocatoria de consultas populares
por vía de referendum en cualquiera de sus modalidades, es competencia
exclusiva del Estado.</p>
</articulo>
</doc>
```

(c) Modified document for the example: new version of *lo2-1980.xml*

Figure 3.11: Version generation. Input and output documents.

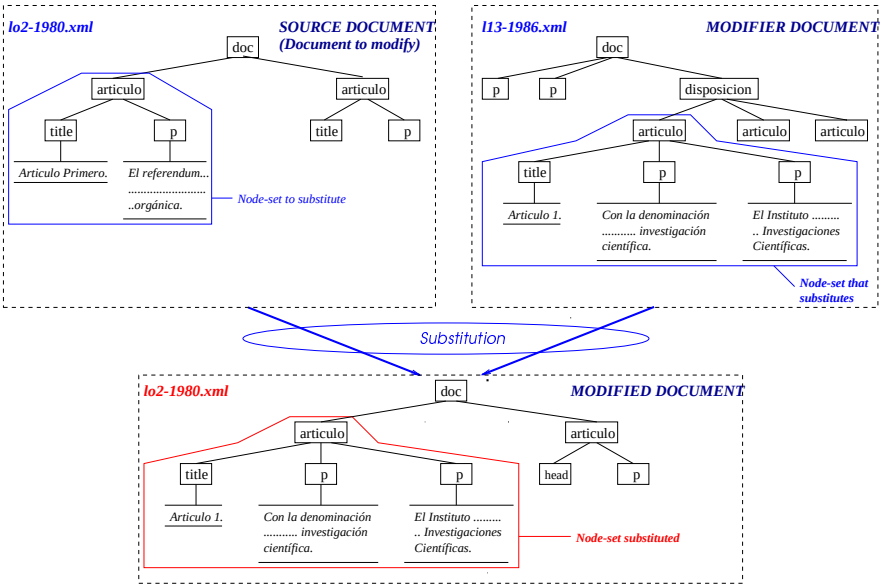


Figure 3.12: Element substitution, based on links. The first element *articulo* in the source (*lo2-1980.xml*) document is substituted by the first element *articulo* inside first element *disposicion* of the modifier document. The result is a new version of document *lo2-1980.xml*.

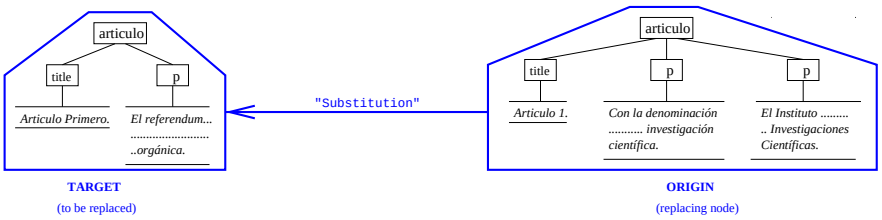


Figure 3.13: "Substitution" link. The ORIGIN will replace the TARGET when generating a new source document version. The ORIGIN is a subtree of the source document made up of the element *articulo* and all its descendants. The TARGET is the subtree in the modifier document tree whose root is the first element *articulo* inside the first *disposicion*.

3.7.2 Versioning graphs

Every document version is the result of resolving a versioning graph obtained from the modification graph. There are no loops in the modifications graph: no document can modify a later document, and in consequence there will be no loops in the versioning graph.

The versioning graph associated to the requested document version is obtained with:

1. The tree, T, of the document version available in the document database. This version is the source document for the versioning process.
2. The set of modification links, M, that reach some node in T.
3. Modification links that reach some source node of links in M should also join M.

Modification links in the versioning graph have a priority attribute: the date of the link (the date when the modification was stated). This priority attribute will be used to resolve conflicts such as that found when a node is affected by more than one modification.

3.7.3 The document version generation process

The document version generation algorithm operates by resolving links during the versioning graph traversal. The algorithm 2 generates the version of a document D, applying modifications to it that match the filter criteria expressed by the priority argument -the date-. Only modifications previous to that *date* are applied to D.

The document version generation algorithm deals with the source document tree in a recursive manner, beginning at the root node and treating its descendants in the same manner till there are no more nodes to consider. Every new call to recursion supposes a descent on the document tree (more granularity in the document fragments considered).

When the algorithm reaches a node, this can be in one of two possible categories:

- It is a node affected by a substitution. That is, it is the root of some modification link's target. A modification to a given node in a structured document affects the node and all its descendants (for example, figure 3.12 shows that to substitute the first element `articulo` of document lo2-1980.xml by a new one, means to replace the whole subtree with the root in this element).

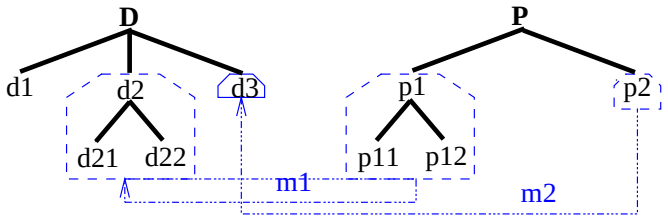


Figure 3.14: Versioning graph.

- It is not affected by modifications. There are no modification links that reach it.

The recursion stops when:

1. The node to be dealt with is empty: all document nodes have already been visited (including document content). This case is reached when the recursion path to the current node includes nodes that are not modified.

2. The node is modified. With modification, the process finishes, as all its descendants have been modified with it.

The recursive call takes place when the node is neither empty, nor a link target. Such nodes must be copied as they are in the source and recursion must continue on their descendants, as they could be affected by some modification.

Example 19. Document D in figure 3.14 is composed of three elements (d_1, d_2,d_3); element d_2 is itself formed by two other elements (d_{21},d_{22}). Document P contains two modifications to D, whose application gives as result a new version of D:

- Element p_1 (with its descendants p_{11} and p_{12}) replaces element d_2 in D. This is modification m_1.

- Element p_2 replaces element d_3 in D. This modification is called m_2.

The version generation algorithm starts with node D. A modification to this node would suppose a replacement of the whole document by another document or document fragment. As this node is not affected by any modification, the algorithm continues exploring the possibility that its descendants -e_1, e_2 and e_3- are themselves the object of some modification.Node e_1 remains untouched. e_2 is replaced by p_1 and its descendants while e_3 is replaced by p_2, thereby completing the recursion. □

3.7.4 Node versioning

To *resolve* a modification link is to apply the modification expressed by it. That is, to *replace* the target vertex content by the source vertex content. When the current node is affected by some modification, there can be several situations to consider:

The versioning of each node can be in one of the following three situations:

- *Simple case.*
  The node is affected by a unique modification link, l. Treatment of the node limits to resolve l, that is, to substitute the node by l's origin.

- *Transitive modifications.*
  There is a sequence of historical modifications: the origin node of some modification is itself the target of other modifications (it is modified somewhere else). This situation is shown in figure 3.15. The resolution implies an ordered traversal of the sequence of transitive links, where modifications are applied in a time ordered manner.

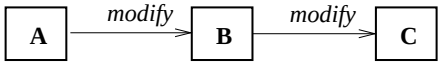


Figure 3.15: Transitive modifications.

Given two links e_1 and e_2 involved in a transitivity, two subcases can be distinguished, depending on the link date criteria:

  – $e_{1_T} \subseteq e_{2_S}$ *(e_1 modifies a portion of e_2's source) and e_1 is more recent than e_2.*
  e_2 should be applied, but e_1 should be applied first on the source vertex of e_2. The diagram in figure 3.16 shows the situation when $e_{1_T} \subset e_{2_S}$.

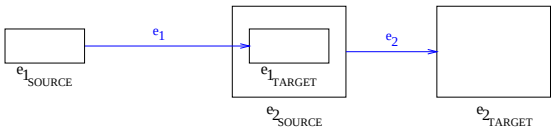


Figure 3.16: Transitive modifications. $e_{1_T} \subset e_{2_S}$.

Example 20. The example in figure 3.17 shows a fragment of the versioning graph used to generate a new version of document D. The substitution of the subtree with root in node d_3 implies replacing it with the subtree formed by node p_2; but as this should be substituted by node n_2, d_3 is, in the end, substituted by n_2. □

  – $e_{1_T} \subset e_{2_S}$ *(the same figure as the previous case) and e_2 is more recent than e_1.*
  This situation is impossible: a document cannot modify a later document.

- *Modifications overlapping.*
  In this case the conflict is due to the fact that a node set is the target of several modifications (n-arity in the node). There are several subcases to consider:

  – *All modifications apply exactly to the same node set.* That is, they address the same node (root node for the node set). This is the case seen in figure 3.18. The conflict is resolved by applying the *priority* criteria to the modifications. Only the most recent is applied.

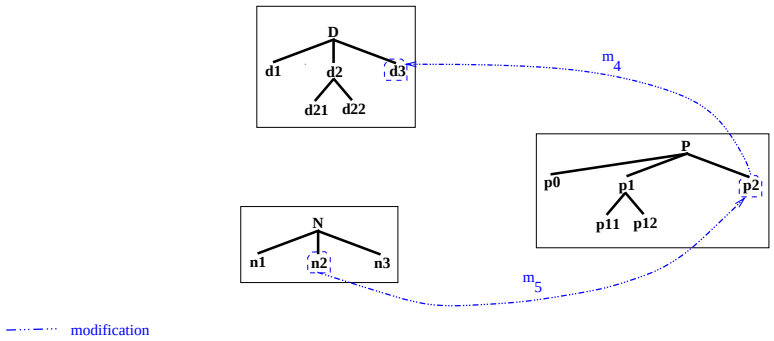


Figure 3.17: Transitive modification in version generation.

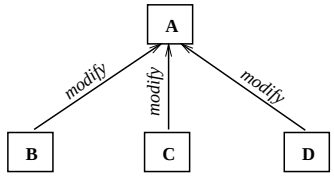


Figure 3.18: Exact overlapping; all targets match.

– *There is an inclusion of two or more link targets.* All links address the same document (see figure 3.19), but different overlapping fragments.
If two links e_1 and e_2, are considered, there are two possibilities:

1. $e_{1_T} \subseteq e_{2_T}$ *and* e_2 *is more recent than* e_1.
The target of e_1 is included in e_2's target. The link that affects the bigger fragment (e_2) applies because it is the more recent and the conflict is resolved.

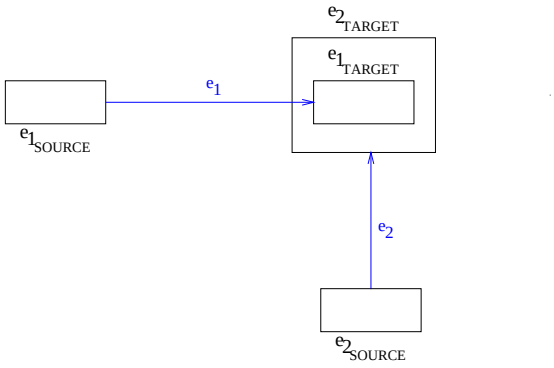


Figure 3.19: Modifications overlapping. $e_{1_T} \subset e_{2_T}$.

Example 21. An example of such a situation would be a theatre piece to which the author first changes scene X and later decides to change the whole act that contains the scene previously modified. A replacement of the act supposes a "refusal" of the first change. □

Example 22. In figure 3.20 there is another fragment of the versioning graph extracted from figure 3.9(c). The modification to d_{21} is ignored and only the one that applies to d_2 is used. □

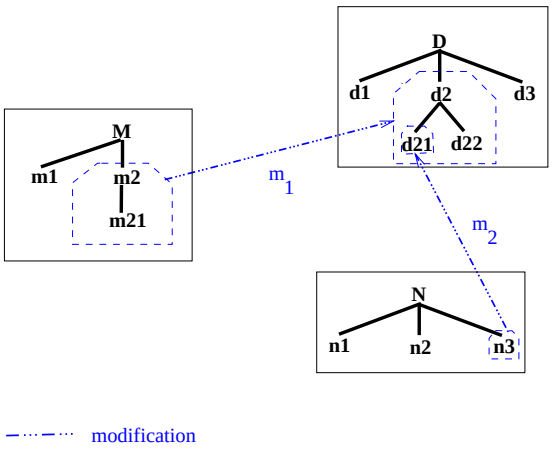


Figure 3.20: Modifications overlapping.

2. $e_{1_T} \subseteq e_{2_T}$ *and* e_1 *is more recent than* e_2.
   The difference with the previous case is the date.
   There is a possible incoherence, depending on how the modifications were created :
   * In the first case, the modification represented by e_1 was stated without knowledge of the existence of a previous modification (the one represented by e_2). If the application of e_2 changes the logical structure of the target document, e_1 cannot be applied. This is an *incoherent* situation.
   * The second situation is when the modification represented by e_1 was stated with knowledge of the existence of a previous modification (the one represented by e_2). In such a case, the modification stated in e_1 considers the new logical structure caused by the application of e_2 and both modifications are applicable.

   The difference between these two cases comes from the creation conditions that cannot be controlled or detected by an automatic engine. In consequence, the case where $e_{1_T} \subseteq e_{2_T}$ and e_1 is more recent than e_2 is considered as an *anomaly* and is not treated by the version generation algorithm.

The algorithm

Algorithm 2 versions a document copy `D`, applying to it all modification links in the range from `D`'s date to the **`date`** argument. The recursion for each node is shown in algorithm 3. The empty tree implies the end of recursion directly. In lines 2 through 5 all links with the target in the current node are selected and the cases of coincident overlapping are resolved: O_L is the node-set that should replace it. If there

is no inclusion overlapping, the current node can be substituted by O_L (the result of applying its own transitive modifications to O_L). Before substituting a node by the link's origin (O_L), this origin is updated -in case it has suffered any modification itself-. Then, O_L is ready to replace the current node (T_L). If there is inclusion overlapping in modifications between the current node and its descendants, the two possibilities explained on page 65 are dealt with: more recent modifications to its descendants cannot be dealt with and are output in a list of conflicts that cannot be resolved by the algorithm, while modifications later than the one that affects the current node are cancelled. The induction case (nodes output untouched) supposes recursion calls from lines 18 through 20.

Algorithm 2 Algorithm for modifications. Document treatment.

Inputs: D: document, date: criteria
Outputs: A new version of D. A list of conflictive node couples.
1: Let d be the root node of D
2: $Modify(d, date)$

Algorithm 3 Algorithm for modifications. Node treatment.

Function $Modify(n : node, date : criteria)$
1: **if** n is not null **then**
2: **if** n is the target vertex of some modification link **then** *{n is the root of some node set in a link's target}*
3: Let M_L be the list of modification arcs with n in the target's root
4: Let L be the link in M_L with more recent date attribute
5: Let O_L be the origin node of link L
6: Let M_D be the list of descendants of n that are the target in some modification link
7: **if** M_D is empty **then** *{There is no inclusion overlapping}*
8: $n \leftarrow Modify(O_L, date)$ *{Treat transitive modifications before applying O_L}*
9: **else** *{There is inclusion overlapping between n and its descendants}*
10: **if** there is some node c in M_D with a modification date more recent than L's date **then**
11: Mark n with an *Abnormal actualizations* conflict flag and do not version n
12: Add the pair (n, c) to the list of conflicts
13: **else** *{All modifications to children of n are previous to L and, thereby, cancelled by L}*
14: $n \leftarrow Modify(O_L, date)$
15: **end if**
16: **end if**
17: **else** *{n is kept untouched. Its descendants may, however, be affected by some modification}*
18: **for all** $c \leftarrow child(n)$ **do**
19: $Modify(c, date)$
20: **end for**
21: **end if**
22: **end if**

3.7.5 Input and output documents in version generation

Input documents to the transformation process are documents from system databases, of some of the document types allowed in the system. Links come from the links database. Thus, input data to the document generation process are:

1. A document to update (the *source* document)
2. A set of documents with the modifications that update the source document
3. A set of links that represent the modifications: they relate the source document with modifier documents

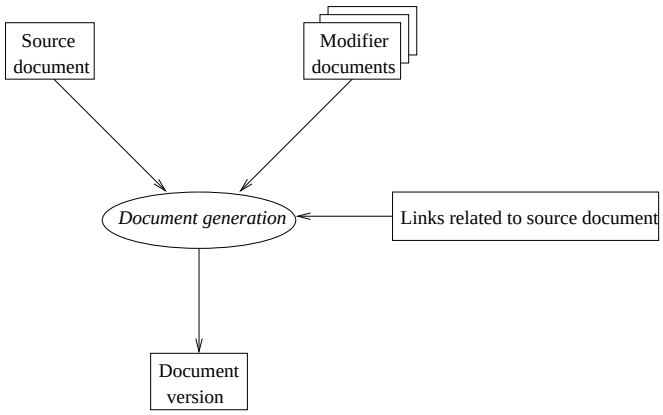


Figure 3.21: Data in the generation process

The source document and modifier documents can be of different types (conform to different DTDs). Indeed, the class of input documents does not modify the process for obtaining an updated version.

3.7.6 Modelling the graph with a links database

Link fields and semantics

A link can be modelled as a record that contains several fields. The link composition derives from the requirements to the links:

- Links must be able to address internal document fragments.
- The links database should accept queries about linking information.
- Links can be queried multidirectionally. A citation or modification can be exploited in both link directions. For example, it can be asked *What documents modify this document?* or *What documents are modified by this document?*.

- There is n-arity in the graph: a node can participate in more than one link. For example, the algorithm in section 3.7 needs to access the set of nodes that modify a given node.
- The links database must be accessible separately from the documents: to query this database, it must not be necessary to enter the documents. This condition is necessary to exploit multidirectionality and n-arity.

Fields corresponding to the link's origin are:

- **Document identifier**: logical document identifier (see chapters 2 and 4).
- **Physical location**. Address of the available document digital copy.
- **Internal locator** in the document, identifying the citing or modifying fragment.
- **Link type**: any (citation, modification).

  The type of the link is modelled on the source node: for links considered in this work (citations and modifications placed in some document), their type comes in fact from the content of the source link vertex. That is, it is natural to express the type beside the source node, as it is this node which influences the link type.
- **Document type**. It is the class the document belongs to. The document class can influence link types (an example can be seen in the prototype).
- **Document date**. In the case where several modification links with different origins share the target (document and internal fragment), the date is the criteria to decide which one applies.

Concerning the Link's target, the fields are:

- **Document identifier** (the same as for the origin)
- **Physical location** (the same as for the origin)
- **Internal locator** in the document (idem)

Representation with XLink

XML offers a solution that complies with all the requirements posed to the link records database: *out-of-line* link databases that can be accessed separately from linked documents, *extended* links to model multiplicity and multidirectionality, and XPointer to address internal document fragments.

3.8 Application to legal documents

Citations are very common in the legal text domain, where documents contain many references to previous rules or jurisprudence. Rules and decrees enumerate modifications to well-delimited parts of other documents, as well as citing the document that is

modified and the fragment affected by the modification. The results are new versions of cited documents.

For example, the Spanish rule *Ley 13/1986*, cites the fourth *artículo* of the *Ley 11/1977* :

```
A los efectos de su gestión económico-financiera los Organismos a que se
refiere el artículo 13 de la presente Ley se entendrán incluidos en el
apartado b) del párrafo primero del artículo 4 de la Ley 11/1977,
General Presupuestaria, de 4 de enero.
```

The same document (*Ley 13/1986*) includes a modification to the *Ley Orgánica 2/1980*:

```
[...] Quedan modificados los artículos 1°,  2°, 4° y
 8° de la Ley Orgánica 2/1980, de [...] que quedarán
redactados en la forma siguiente:

Artículo 1.

Con la denominación de Instituto de Astrofísica de Canarias se crea un
Consorcio Público de Gestión, cuya [...] y el Consejo Superior
de Investigaciones Científicas.

Artículo 4.

El Consejo Rector [...]
```

The new version of the *Ley Orgánica 2/1980* can be obtained by applying the modifications in *Ley 13/1986* to it, using algorithm 2 (the result of this application is the document shown in figure 3.11(c)).

3.9 Discussion

One of the most popular ways to model relationships is hypertext. Links express relationships and a navigational graph is obtained. Navigation actions traverse the graph from link sources to targets, and the number of links that share the same source node introduce complexity in this traversal. Linked resources are documents, and the achievement of more granularity requires evolution to systems and standards able to work with structured documents (for example, XML).

The typed link graph considered in this work is not a navigational one, and linked entities are document fragments. One difference with the traditional hypertext is the types of relationship considered and the use made of the graph. The citation subgraph could be the equivalent to the traditional hypertext graph: a citation is a link to a cited information piece that could allow navigation. But the modifications graph can be exploited in a more interesting way than navigation that would not finally manage to provide the expected help to the navigator user, who risks getting lost inside the navigation graph [92]. The modification graph can be used to provide users with desired documents, freeing them from the navigation and composition task necessary to obtain

a document version from a navigational hypergraph. Different traversals can be used to query the graph about relationships.

Relationships are not a problem exclusive to textual documents, but these do have an interesting peculiarity: modifications are mostly expressed inside documents that are themselves a semantic unit that should not be fragmented. This means that elements involved in a modification relationship are not first class entities (they are not documents most of the times, and they are not even files, but fragments); this is also a difference with software updates, where updates expand complete files [41]. The second interesting quality is that it is possible to extract the relationship from a document text, while paying attention to the target document structure. Elements implied in modifications do not need identifiers to address them; they can be addressed by their position in the document tree. Node-sets involved in a relationship can be addressed by the position of their root node in the document tree. That is, it is possible to work with any structured document, to manipulate any structure, without depending on fragment identifiers, which are more human dependant.

Some types of relationships dealt with in this thesis -citations and modifications- are embedded inside document content and affect document fragments. That puts the link graph obtained in this work in a granularity level higher than that considered in traditional hypertext or object versioning, where the links always related first class entities. To link pieces instead of complete entities allows a further step to be taken in the precision of the links, and to proceed to the generation of versions, which is impossible without this granularity. Modifications are "geographically" close to a citation that designates the target of the modification that follows. There are two links (a citation and a modification) that share the target, but have different origins; two heterogeneous links are close. The integrity of documents that contain modifications is untouched: they are never fragmented to obtain individual entities that could be directly inserted in a new document, neither their content, structure or attributes is touched. Relationships are independent information and are kept aside.

Besides relationships, versions have been considered in this chapter. There are three main aspects of interest related to document versions:

1. Detecting changes. There are two possible ways to do this: from citation detection, or comparing versions. In this thesis, the assumption is that not all versions are available, while documents that contain modifications are more certainly present. That leaves detecting changes from citations as the only possibility to detect modifications in this thesis.

2. Representing changes. There are three ways to represent changes:

   - To store all versions caused by a change.
   - To represent changes as annotations (attributes) to affected nodes.
   - To represent changes as links.

3. Querying changes. The way to operate here is dependent on the choice made for representing changes.

- If versions are first class objects, the only way to query changes is to search for differences between versions. It would be a file or tree (in case of availability of the document structure) comparison.
- If changes are represented as annotations, to query them is to query annotations. In the case of structured documents, that means to query node attributes.
- If changes are represented by independent links, to query changes is to query links, which is the same as querying any document.

The solution of maintaining all versions simultaneously has shown to have its main difficulty in links maintenance [38, 101]. The solution that represents changes as node attributes uses a document tree representation: it works with structured documents. It is a solution whose aim is to query the history of nodes in documents. The aim of querying the changes takes relevance over the desire to access document versions: to know the history of a node can be done by querying its attributes [36, 35].

Conversely, the approach chosen in this thesis is to maintain links separately from document content. This has several benefits: relationships can be exploited bidirectionally (the graph can be traversed in several directions), documents that participate in the relationship are not touched (the structural forest graph and document contents are not modified by representing relationships, as would an insertion of linking elements inside documents), the problem of maintaining revision links disappears, and the process of version generation is simplified (it can be based on a document tree, considering modifications as links that reach this tree).

The algorithm presented in this chapter generates a new version of a structured document, using the information about the historical modifications it has suffered. The versioning algorithm always works with the logical structure using a traversal graph algorithm. The versioning graph used to generate a document version uses links that reach the document that is the basis of the traversal. The complexity in this version generation process comes from the number of links that reach a target, which is different from navigation, where the complexity is due to the multiplicity in the source.

The problem of maintaining revision links detected in [38, 101] disappears when versions are automatically generated. The generation of versions proposed in this thesis goes a step further than a composition of documents made by following the rules expressed in some type of "structural" link between the frame document and the composing ones [109, 8]. Here, the composition rules are deduced from non-structural semantic links (modification links).

We are only aware of one proposal to automatically generate document versions; it was recently proposed in [13]. They indicate the possibility of keeping rules to automatically generate document versions. The version generation algorithm presented in this chapter does not use rules that indicate how to generate versions, but the information it uses is the relationship information stored in links. It is the link graph traversal that allows versions to be generated, with variable parameters (such as the version date).

The application of the versioning algorithm proposed in section 3.7 is limited when a document is the target of several incoherent historical modifications: modifications are made without considering modifications that were previously made. In this case,

there is no way to decide which one should apply and the algorithm leaves this situation as an unresolved conflict, that should be presented to the user for decision.

A complete automatic treatment of links could consist of an automatic detection of citations, which would allow the document identifiers and document fragment paths to be generated automatically from citations. There are experiences of generating document identifiers from citations [31]. The main problem with this detection is the variability of the citation language. Up to the moment, only in areas where the language used is rigid (as legal domain language is), has it been possible to obtain some kind of automatic recognition [121]; these results should be expanded to adapt them for work with structured documents (to generate document fragment paths).

4

Link-oriented architecture

Contents

This chapter focuses on the digital library architecture that allows linking functionalities to be obtained. This architecture (and protocol) integrates these functionalities with "classical" functionalities in digital libraries (querying, document retrieval and browsing), proving that link-oriented functionalities can be integrated in a digital library without affecting the operation of existing functionalities.

Digital libraries offer their users functionalities that allow them to exploit the information held in these systems. Tasks needed to acheive the correct completion of users's requests are the responsibility of the system. Digital library architectures should facilitate the distribution of tasks among cooperating services (or components in the implementation phase) in order to achieve the architectural properties specified by software engineering guidelines (open architecture, federation, integrability and interoperability) that in this particular type of system, which is the digital library, are particularly critical, due to the way digital libraries are, in most cases, built: integrating legacy, heterogeneous data and systems to get a single digital library that offers users an integrated view.

If minimal functionalities in a library (retrieving, querying, and browsing) are considered, a reference model can be obtained for digital libraries, which can be found in any of these systems. Three models are given special attention in this thesis: the reference model for basic services, a model for reference linking, and an architecture for document manipulation.

Integration of collections and cooperation of services requires the use of a protocol that controls the interaction among participating services. Several protocols have been defined expressly for use in digital libraries. The more classical are Information Retrieval protocols, based on a client/server model, such as Z39.50. However, these protocols are for situations of collection distribution and their main aim is to facilitate *queries* among remote collections. More recent protocols are designed with the aim of integrating the system operation, distributed among various services. Besides querying and data retrieval other functionalities are considered, which results in *service*-oriented protocols. Examples are *Dienst* and protocols defined in the *Stanford Digital Library Project*, both defined for use in digital library environments.

The link-oriented architecture proposed, which deals with linking integration, consists of the services architecture and their interaction (through their interfaces) protocol. This first abstract functional view of the library as a set of services is translated to a development view where the services are implemented by a set of components that interact by mutual invocation.

The prototype presented in chapter 5 is an implementation of linking-oriented services, based on the components architecture shown in subsection 4.2.6.

4.1 Protocols and architectures in digital libraries

Digital libraries may offer a variety of functionalities [88, 90, 89, 10, 11], that are commonly implemented by a set of separate (but cooperating) services. In any case, the functionalities offered influence the system architecture. The aim of generating new documents through document manipulation and relationships derived from citations

between documents, means special attention must be given to the following references: a reference model for basic services in digital libraries (presented in subsection 4.1.1), a reference architecture for citation linking (explained in subsection 4.1.2), and a proposed architecture for document manipulation (shown in subsection 4.1.3).

Interaction between services is controlled by high-level protocols specially designed for use with digital libraries. Two of the most complete examples of such protocols are described in this section. The first one is *Dienst* [43], the protocol defined in Cornell for the NCSTRL and the ERCIM Technical Reports digital libraries . The second set of protocols are part of the Stanford Digital Libraries project [104]. These protocols are described in terms of their services and interfaces in Dienst, and, in the Stanford project, their components and interfaces. But the existence of these service-oriented protocols does not mean that no other (lower level) protocols can be used in digital libraries. Components that implement these services interact using protocols such as HTTP [44], protocols that are part of CORBA [63, 122], and Information Retrieval protocols -such as Z39.50- [104, 9]. Z39.50 is an Information Retrieval protocol that considers services for retrieval and querying and that has inspired some design decisions (mostly, in semantic issues) in later digital library protocols.

4.1.1 Basic reference model for a digital library

Basic functionalities in digital libraries are searching documents, retrieving documents and browsing in library collections. Users can exploit them through user interface services that facilitate their interaction with the library system. These functionalities should always be present, whether or not the library is distributed, there is heterogeneity, and whatever additional characteristics the library has or services are offered. The set of services responsible for providing these functionalities is:

- *Repository services*, which provide access to repositories in the library (document retrieval).
- *User interface services*, that provide users with an interface to the rest of the services (and data) in the library.
- *Search services*, that enable documents in the library to be searched and to get the collection of document identifiers that match the query.
- *Naming services*, associating user meaningful names to digital objects, allow users to access intellectual works (instead of digital objects) [10]. This service introduces a level of abstraction for the physical implementation of (abstract) documents as digital files.

These basic functionalities result in the reference model of figure 4.1. In the most basic digital library the user interface service interacts with all the rest. It transmits retrieval requests to the repository services, it passes user queries to search services and it uses the naming service to obtain the mapping from intellectual work identifiers to digital object addresses.

This model is expanded in every different library according to its particular needs. Heterogeneity provokes the presence of *Translation services* [98, 100, 91]. Distribution

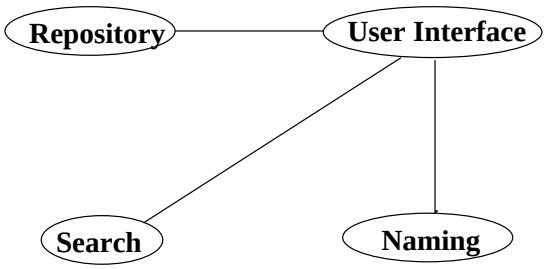


Figure 4.1: Basic services in digital libraries.

also influences the architecture of digital libraries in the sense that new services -that look after the distribution of queries and the redirection of data and messages- integrate the system [45, 122, 22, 61].

Some examples where these basic functionalities can be found are the *Networked Computer Science Technical Reports Library (NCSTRL)* [44], the *Ercim Technical Reference Digital Library* [22], or the *Networked Digital Library of Theses and Dissertations (NDLTD)* [95].

Implementation of this set of services in every particular library results in a set of components in each library. They include software components (or agents), as well as data repositories. Data that qualify other data and data about the system (metadata) are included in the design of the data architecture. *Collection services* are offered by *Collection Interface agents* [24], also, *user interface services* are, in most cases, implemented by a *User Interface agent* [24, 11]. In any case, variations are possible and the rest of this chapter introduction focuses on protocols that are used as the basis for these architectures, as well as on two models (subsections 4.1.2 and 4.1.3) that were found to be useful for guiding the design steps in the architecture modelization.

4.1.2 Citation-linking architecture

There is a reference architecture to which almost all systems that provide citation linking for journal articles conform [31]. This model deals mainly with the resolution of citations to equivalent document identifiers, and with the resolution of document identifiers to physical objects in the collections. The components architecture (in figure 4.2) is centred on three types of databases, used to resolve citations. The provider of the information (*Publisher*) supplies metadata about each work, that is stored in the *Location database*; the provider also updates the collections. The *Location database* holds the mapping of work identifiers to physical locators (URLs), and is used during the resolution of the identifier. The *Reference database* contains metadata which, at the very least, correspond to the information in a conventional citation, and that are used for resolution of identifiers. The *Content database* contains the documents, which will be effectively retrieved after the resolution of the identifier. The *Client* component is a generic one that represents any application able to provide documents to users or other software clients.

The *services* provided are the ones that allow documents to be *retrieved*. They

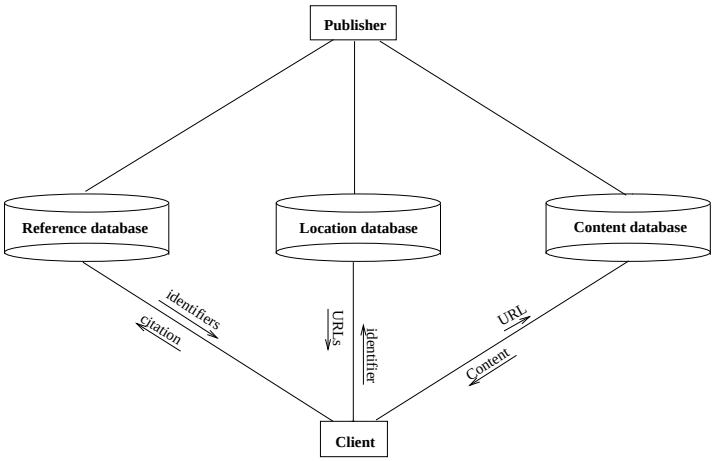


Figure 4.2: A components model for reference linking in journal articles.

include -with reference to the services in figure 4.1- the *Repository* services (for access to records in collections) and *Naming* services (to resolve identifiers). Naming services are unavoidable in reference linking, as is the way to allow the abstraction from physical implementation at the same time as mapping citations to digital object addresses.

Data flow between components is also shown in figure 4.2. The *citation* is sent by the client to the *Reference database*, which answers with a list of *work identifiers* that match the citation. The client sends the *identifiers* of interest to the *Location database*, which sends a list of URLs (physical addresses) that match the document identifier. Finally, access to document content can be achieved directly through the physical identifier (the URL).

Naming services, in most cases, follow conventions expressed in the *Digital Object Identifier (DOI)* [94] for their metadata (this standard is presented in chapter 2).

4.1.3 Document manipulation architecture

Arnold-Moore et al. [15] propose an architecture to deal with document manipulation. The model is defined to suit the functional requirements of a system that treats documents. These requirements are of four types:

- *Data definition*: semantic and structural constraints on data can be obtained from documents or -if available- from a document class definition.
- *Data retrieval*: access and querying of documents, which means access to entire documents, individual elements, element attributes and their values, document content and document metadata.
- *Data manipulation*: retrieving and arbitrary reuse of data elements, as well as generation of new data.
- *Document management*: this includes support for versioning and document composition. Document versioning facilitates access to all versions of a given do-

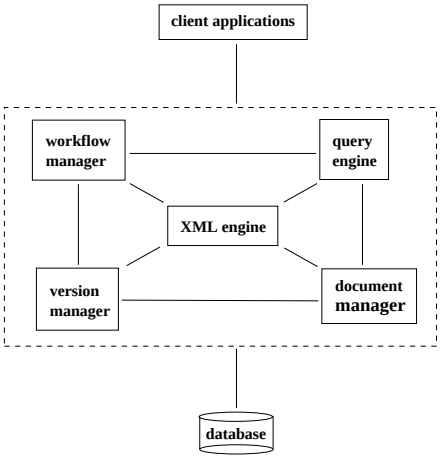


Figure 4.3: An architecture for a document management system; version manipulation has a dedicated component.

cument, distinguishing between the different versions. This requisite affects the identifiers, that can be different in different versions but similar enough to infer that two document objects are versions of the same document. Document composition consists of obtaining new documents from fragments retrieved from other documents. Composite documents can be containers for data that is shared by multiple documents, where changes in shared data have to be automatically reflected in the composite document.

These functional requirements result in the component architecture seen in figure 4.3. A *workflow manager* administers and directs the automated workflow. A *version manager* deals with versions and variants of documents. A *query engine* resolves content, structure, and metadata queries. A *document manager* provides security, access control, check-in and check-out, and composite document support. An XML engine provides document parsing and validation, tree manipulation, and document structure comparison; this component is used by all the others. All components form a block that client applications interact with.

If comparing with services in the basic reference model of subsection 4.1.1, *Search* service is implemented by the *Query engine*, while *Naming* and *Repository* services are implemented simultaneously by the *Version manager* and the *Document manager* components.

4.1.4 Query and retrieval protocols

Query oriented protocols are application level protocols that govern the interaction between a client and a server. They are created to facilitate information retrieval tasks. For this reason, they have been used in digital libraries to achieve interoperability when accessing heterogeneous information sources. Their main contribution is to provide semantic interoperability in attributes used for data description and query languages, which continues to be an open problem in digital libraries outside this protocol influence.

These protocols only govern interaction between actuating agents[1], not dealing with the architecture of digital libraries. The most representative of them is the ANSI/NISO *Z39.50* protocol [124]. The latest version dates from 1995. Its initial purpose was to facilitate access to bibliographic catalogues; it was later expanded to deal with other databases than just bibliographic records, such as document collections. The model architecture used by Z39.50 is the *client/server* model, where a client (*origin*) requests some services from a server (*target*). Z39.50 only rules the dialogue between them; it does not specify anything about the data structure, their organisation, the kind of metadata used, etc. Thus, one of the roles of the server and client applications is to translate from local data representation and semantic to Z39.50 criteria.

This protocol is used by the client to query and retrieve records from the server. The interaction is modelled as the exchange of a set of messages that invokes one of a set of facilities defined by the protocol. Some of the most representative are:

- *Initialization Facility*, to fix the dialogue parameters.
- *Termination Facility*, to end the dialogue session.
- *Search Facility*, to query the server databases.
- *Retrieval Facility*, to retrieve records from the server.
- *Explain Facility*, to query the metadata database in the server.

4.1.5 Multi-service oriented protocols

Later protocols for digital libraries than Information Retrieval protocols consider the need for advanced functionalities that complete basic digital library services.

Dienst

Dienst [43] is the architecture and protocol used by the *Networked Computer Science Technical Reports Library* [82], and the basis for the *Cornell Reference Architecture for Distributed Digital Libraries (CRADDL)* [77]. Dienst defines a set of services and the interaction between them. The set of services specified by Dienst can be seen in figure 4.4:

- *User interface services*, that provide a human-friendly gateway to the information obtained from other services. User interface services interact with other services and deal with the correct evolution of operations.
- *Repository services*, that store and provide access to documents, according to the Dienst document model.
- *Index services*, that provide search capabilities, accepting a query and returning a list of identifiers, corresponding to documents that match the query. They need access to repository services, to extract indexing information.

[1] The term *agent* is used here in its most general sense: any individual application able to execute some task.

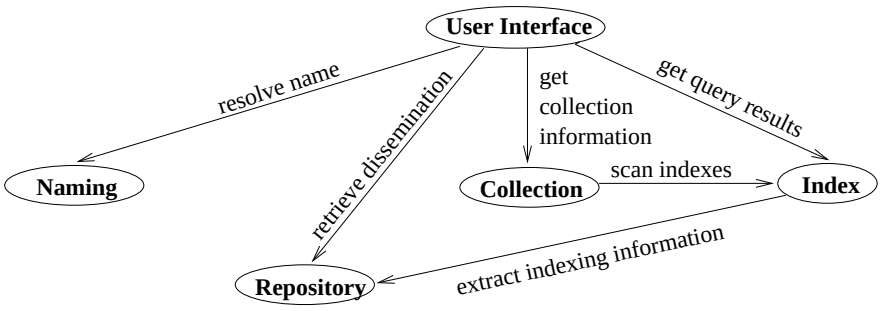


Figure 4.4: The NCSTRL services model.

- *Collection services*, that define and enable access to digital library collections.
- *Meta services*, that provide information about collections and services in the library.
- *Naming services*, that resolve URNs[2] to one or more physical collections.

Stanford InfoBus

The Stanford Digital Libraries project aims to develop a set of service protocols to allow heterogeneous information resources and services to cooperate. It does not focus on a particular digital library, but tries to deal with a general model of services. Protocols and functionalities are proposed to allow such heterogeneous services to interact. A collection of services that can interact in a digital library is defined and protocols are designed on the basis of these services. The set of services considered in the Stanford Digital Libraries project is mainly:

- Resource discovery services, such as *GLOSS* [64].
- User interface services. Two examples of this type of services are *DLITE* [105] and *SenseMaker* [17].
- Information processing services. For example, document summarisation services.
- Search services.
- Translation services.

The set of protocols that rule the interaction between these services is referred to collectively as the *Stanford InfoBus*. They are divided in five service layers [98]:

- Protocols for managing items and collection (*SDLIP*). The SDLIP protocol allows users to access information source collections, to search and retrieve records. This protocol facilitates access to the collections and searching them.

[2]URN (Uniform Resource Name) is a type of *persistent*, location-independent identifier that serve to locate an object within a namespace. It is different with the popular type URL (Uniform Resource Locator), that provides a *non-persistent* means to uniquely identify an object within a namespace.

- Protocols for search (*STARTS*). The aim, in this case, is to facilitate the choice of the best sources for querying, as well as merging the query results from these sources.
- Protocols for managing metadata (*SMA*).
- Other service oriented protocols: rule payment (UPAI) and management of rights and obligations (FIRM).

4.2 A proposal for linking-oriented services

The aim of this proposal is to provide a solution for link-oriented services, and to show how they integrate with the rest of the functionalities in the system. Link detection, queries about relations and document version generation, the three most important novelties in terms of services offered by digital libraries, require special attention to be given to linking aspects when designing services to provide these functionalities. These services are not classical in digital libraries, but can be integrated with other digital library services. The service model is used to introduce the architecture, as it is a model whose flexibility is proved (see, in part 4.1.5, how it facilitates integration of new functionalities with existing ones: basic services are present in all the propositions, while advanced services vary). The basic services model (subsection 4.1.1) is taken as the basis for the link-oriented proposal: link services are integrated with services in it. Citation linking services (subsection 4.1.2) have been included under the denomination of "Link detection". This naming is more precise and it is necessary to differentiate this functionality from other link-related services. The version problem is dealt with here in terms of version generation, as opposed to version management; the relationships with the architecture commented in subsection 4.1.3 will be discussed in section 4.3.

Services are described in scenarios in section 4.2.2, and the integration of the resulting services with other services is also addressed during the presentation. The result is a division of tasks among a set of services accessible through their interfaces that interact to achieve the desired functionalities.

The resulting architecture and corresponding protocol is described in terms of a set of views that show the system functionality, data flow and data architecture. A set of functional requirements guide the different steps in the design, and are the basis for the definition of a set of scenarios helpful for the design. Qualities of the system which also guide the design process are summarised at the end of the proposition (part 4.2.9).

4.2.1 Overview

The system described allows relationships between documents to be exploited in order to:

- Query relationships between documents.
- Generate document versions.

The system also considers relation detection as one of the desirable steps to automate the complete link treatment:

1. Detecting relations between documents.
2. Exploiting relations. Some examples are the searches for articles cited in another one, to search how many articles cite a given work, statistical treatements done on this information, etc.

Additional services are present in the system to show how services that exploit relationships integrate with other digital library services. The ones chosen are basic services in digital libraries: retrieving documents from library collections, and querying documents in collections. Finally, services that allow the system to function correctly are also included: administration services.

Users' view

Requirements for a system depend on the type of user considered. A digital library can have 'normal' *users* that access the system to get information and data, *administrators* that are concerned with the correct functioning of the system, and *authors* that insert new documents in the library collections.

User requirements

This type of user gets information from the library, but he/she never modifies either the library databases or the library functioning. What such a user can do in the library is:

- To retrieve documents from the system, specifying the identifier of the desired document.
- To obtain a list of documents that reference a given document (to query relationships).
- To retrieve "versions" of documents, specifying parameters that allow the desired version to be selected.
- To search by keyword in document collections. What the user will get in this operation is the list of document identifiers that match the query.

Other user related functionalities that could complete this list will be commented in section 4.3, within future possibilities to improve the system.

Administrator requirements

The administrator is the only user that can modify system functionalities or parameters that affect the whole library. That is, the administrator is able:

- To insert new services and components in the system.
- To increase the range of document types supported by the library.

Author requirements

Authors are users that cannot modify the library operation, but can modify its databases. They require functionalities

- To insert new documents in library collections.
- To extract relations between documents, and add this information to the library.
- To insert new collections.

Data

The digital library contains documents that are manipulated and presented to the user. Together with documents in system databases, data that express relations between them become crucial in the system. In this way, there are two types of data:

- Documents. They are the potential initial point of a sequence of modifications.
- Links. They express relations between documents (citations, modifications).

A more precise distinction can be made concerning documents. There are two types of document that the system manipulates:

- **Permanent** (or *stable*) documents. These are documents of which there is a digital copy stored in system repositories.
- **Virtual** documents. These documents are generated by the system on demand, and are not stored in any database[3].

  They are types of documents that are generated by the system, by using its services. This category includes:

  - Versions of stable documents
  - References to documents
  - Query answers

The list of permanent data in the system is completed with the following data:

Links

The links that express relationships between documents are stable data but they are generated within the system. This differentiates them from documents in the system that come from external sources.

Metadata

Metadata describe the system, and data in the system. There are mainly three types:

- Document metadata. Each document has an associated set of data that describes it.
- Collection metadata. Description of library collections: types of documents in the collection, etc.
- System metadata. These are data that describe the system composition as services and collections.

[3]This does not clash with the possibility of allowing users to create their own local databases to store the documents that interest them. System databases would not be interfered with in such a situation by user's local ones.

Functionalities

The enumeration of required system functionalities in this section is the cue for the definition of some scenarios that will show how the system architecture emerges:

- Document retrieval
- Keyword searching
- Querying about relations
- Link generation
- Document version generation
- Updating collections

4.2.2 A brief presentation of some scenarios

Keyword searching

In this scenario a user wants to query the library to know what documents contain a term `X`. When the query enters the system, it is redirected to all repositories that make up the library (if the user query language is not the same as that used by every collection in the system a previous step of query translation is mandatory before forwarding it[4]). The result from every collection is a list of document identifiers that will be returned to the user, as can be seen in figure 4.5.

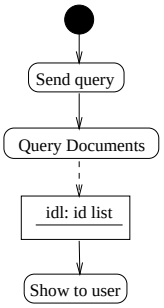


Figure 4.5: Keyword searching; a first draft of the scenario.

Querying about relations

In this scenario the user makes a query; from the user's perspective there is no difference between this situation and searching by keywords. Considering the internal library functioning, this is similar, but with the following difference: the result of the query this

[4]Heterogeneity in query languages is a possibility that is considered in the services model, but which does not receive special attention in this thesis. It is present in the model to show how this characteristic would not affect the rest of the design.

time is not a set of document identifiers, but a set of links. The document identifiers (or whatever other information the user wants) is extracted from the link result set before returning it to the user. Task evolution can be seen in figure 4.6.

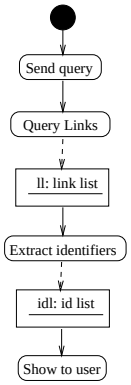


Figure 4.6: Querying about relations; a first draft of the scenario.

Inserting new documents

Authors can insert new documents in the library. The author indicates the collection where he/she wants to insert the document. If the document passes a validation process, it is translated to an interoperable format understood by the library community and the obtained copy joins the adequate repository (database).

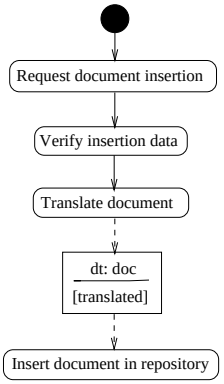


Figure 4.7: Insertion of new documents; a first draft of the scenario.

Link generation

This scenario depicts the situation where a document D has been introduced into the library collections, but its associated links have not yet been generated. A process of analysis of the document is required to detect links with their origin in the document. The process consists of the extraction of citations in the document to other documents

that give the clue for the citation relationships, that can also involve modification relationships. This process returns a set of links with origin in the input document, that can be inserted into the library link databases. The evolution can be seen in figure 4.8.

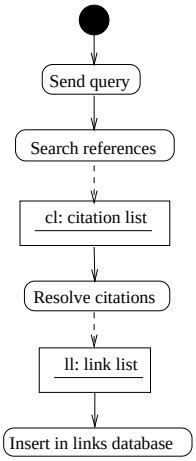


Figure 4.8: Link generation; a first draft of the scenario.

Document version generation

Given a document `D` that has been the object of some historical modifications, a user may want to retrieve the document, at the state it was in at a certain moment. For example, he/she may want the document as it was before any modification was made, or the document with all modifications applied to it. These preferences are expressed as parameters that the system receives with the request.

When the user accesses the system he/she asks to get a copy of `D`, and specifies the preferences concerning the application of modifications. This request is processed by the system that retrieves the document in its original form -with no modifications applied to it-. From user preferences the system generates a query that is used to search links that modify the document, matching the user's preferences. Once the collection of matching links is available, they are processed in order to resolve them and apply the modifications they express to the original copy of `D`. This modification processing results in a new version of `D`, that can be returned to the user. These steps are summarised in figure 4.9.

4.2.3 System services

Scenarios presented in part 4.2.2 prove a need in the system for services able to provide:

- *a user interface*
- *access to document collections*

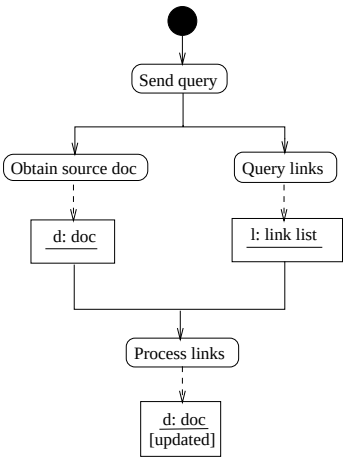


Figure 4.9: Document version generation; a first draft of the scenario.

- *query facilities*
- *document manipulation*
- *updating capabilities that make the inclusion of new documents and links possible*
- *translation (for queries and documents)*
- *a way to designate abstract document entities, in a manner independent from the location and the way the document copy is stored*
- *administration capabilities in the system*
- *facilities to incorporate new categories of documents, data, and services to the library.*

Services presentation

From previous needs, the services emerge naturally:

1. *Document management service*

   This service allows documents in the system to be manipulated. It allows new documents to be composed from document fragments, as well as documents to be modified or processed in any way. That is, every functionality that requires document manipulation has to pass through this service.

2. *Search services*

   Search services offer query capabilities in the system. They are able to process a query and return adequate results. There are two main subcategories inside search services: services to search documents, and services to search links.

- *Document search.* Its normal operation returns a list of document identifiers that match the input query.
- *Link search.* This returns a collection of links that match the query. It is also able to return fragments of links (for example, the target identifier), if it is specified in the query.

3. *Link generation services*

   Link generation services are able to analyse an input document and return a collection of links whose origin is in the input document.

4. *User interface services*

   These services allow the user to query the system, to ask the system to execute tasks and to see (and probably, analyse) results.

5. *Translation services*

   These services provide translation between external ontologies and system ontologies. They transform all queries to a canonical model, as well as documents in the system. These services are necessary to get a common model for data that allows the system services to operate on data. All documents and queries in the system are normalised using these services, in order to achieve data interoperability. There are two kinds of translation services:

   - *Query Translator.* This service translates a query from the user front-end language to the system query language. It is only used if this kind of heterogeneity is present.
   - *Document Translator.* This service translates input documents to an interoperable document schema and format. It is crucial for later document manipulation and composition to count on a common model and format for documents. Without such a common model it would be impossible to achieve any type of functionality that requires document manipulation and composition.

6. *Administration services*

   These services provide knowledge of system collections and methods to work with the system, as well as functionalities to manage it.

7. *Customization services*

   A library that evolves over time, with openness, needs customization services that allow new types of documents, as well as new services, to be incorporated into the system.

8. *Repository services*

   These services provide access to system databases. They allow records ("records" is used in a general

   manner to designate any type of data present in the system), in their most basic form, to be retrieved by using their identifiers. As mentioned in the proposal's

overview, there are two main types of data in the system and, consequently, there are matching services to work with the two types of repositories:

- *Document Repository.* This service provides access to document databases.
- *Link Repository.* Through this service, the link database can be accessed.

9. *Updating service*

   This service provides authors with the facilities to add new documents to the library, as well as to invoke the generation of new data by the system (generation of links).

10. *Naming service*

    This service allows documents to be designated by logical identifiers instead of physical identifiers. The naming service is used to obtain necessary matching between the logical identifier and the physical identifiers of documents' digital copies.

4.2.4 Services interaction

Services interaction is presented in terms of scenarios described in section 4.2.2. Two views are shown: *uses* view and *data flow* between services. A global view of services interaction can be seen in figure 4.15, while data flow between services is shown in figure 4.16.

Keyword searching

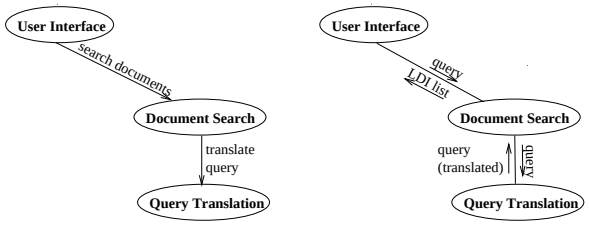
The scenario is revisited: a user wants to search documents that contain a term `X`. The user accesses a *user interface* service that allows queries to be entered in a user friendly front-end language. The user interface service redirects the query to *search services* in the system. It is up to a *query translation* service to translate this front-end query into its native equivalent for each selected collection (this step is only needed in case there is heterogeneity in query languages in the system). After the queries have been issued, the user receives back a set of results. The user interface then unifies these results and constructs an integrated view that is presented to the user. Services interaction for this scenario can be seen in figure 4.10.

Querying about relations

The user that wants to know about relationships makes a query that comes to the system through the user interface service, which passes it to the link search service. It is the responsibility of query translation services to translate the query into an equivalent query understood by the system. The result of the query is a collection of links, from which the information of interest is obtained.

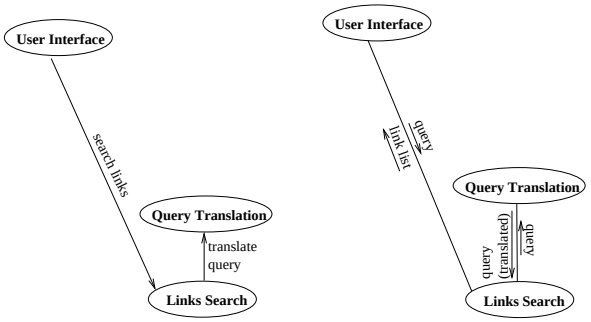
Inserting new documents

The insertion of new documents requires *updating services* that allow new documents to be inserted, validating all the insertion process. Every document that joins the system



(a) "Uses" view. (b) Data flow.

Figure 4.10: Services interaction (use and data flow) for keyword searching scenario.



(a) "Uses" view. (b) Data flow.

Figure 4.11: Services interaction (use and data flow) for querying links scenario.

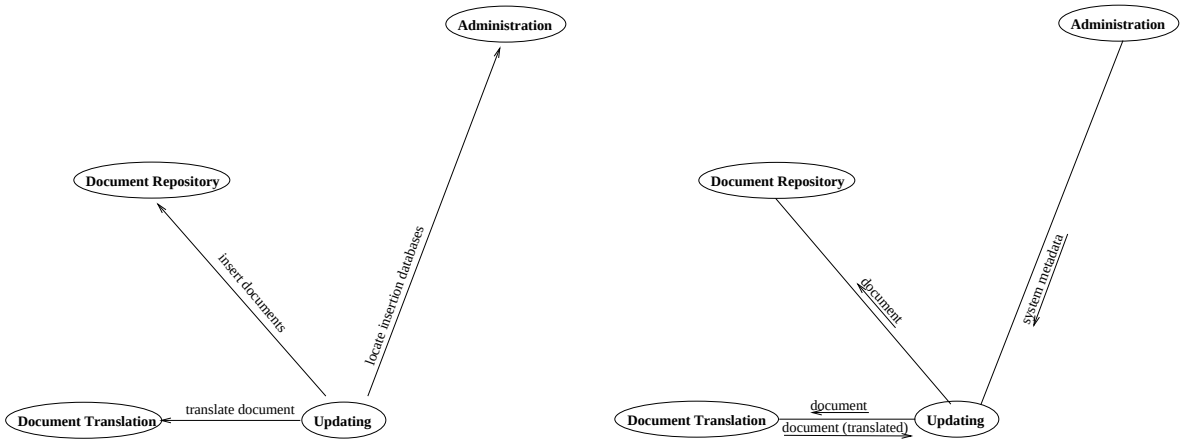
has to be translated to a normalised standard, which is the one services in the system are able to manipulate and query; this task is assumed by *translation services*. The end of a correct operation of this type is the insertion of the document in a database, using *repository services*.

Links generation

Generating links is a task for *updating services*, as this generation implies updating the library link databases. It is up to this service to invoke services to find out the links in the input document (Link detection services), to validate the correctness of generated links -making use of *administration services*- and, if it is all correct, to require the *repository service* to insert the links in the link database.

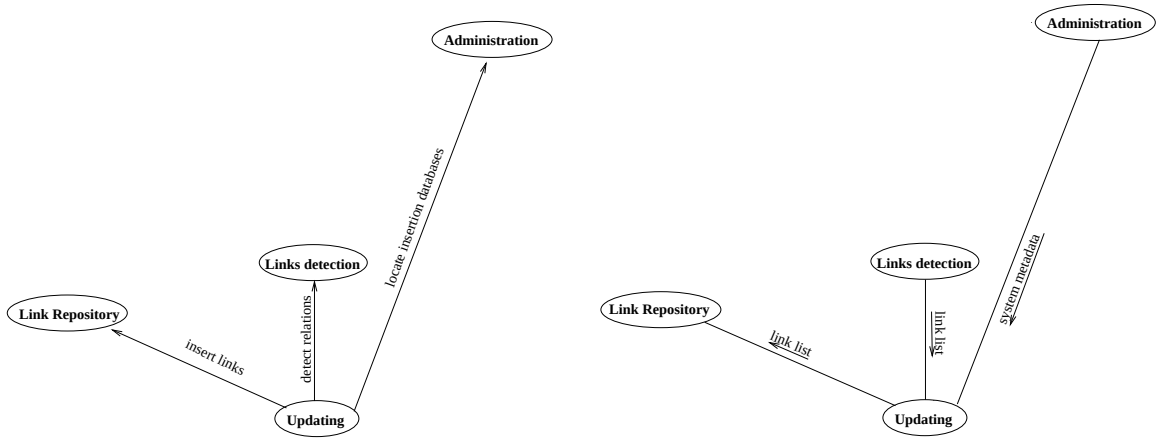
Document version generation

The user request for the updated version of document `D` is passed by user interface services to a document management service that translates the request into a set of queries and actions that result in the composition of the required document. The generation of a version requires: to access the original copy of the document which



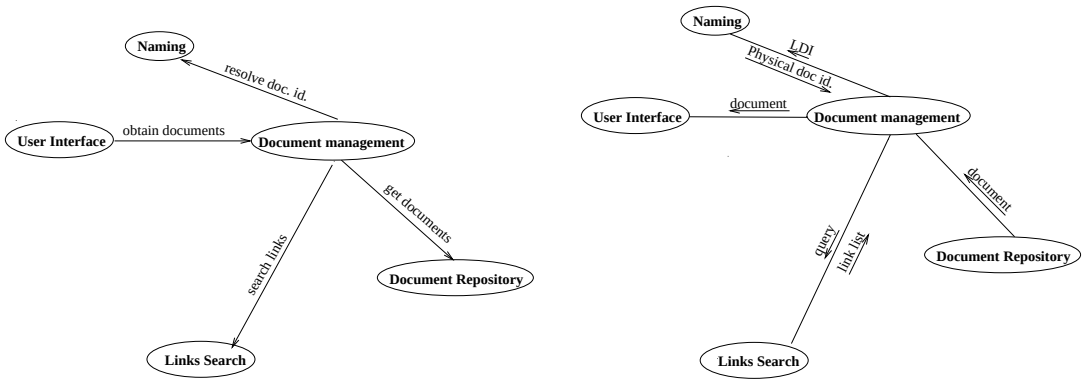
(a) "Uses" view. (b) Data flow.

Figure 4.12: Services interaction (use and data flow) for inserting documents scenario.



(a) "Uses" view. (b) Data flow.

Figure 4.13: Services interaction (use and data flow) for links generation scenario.



(a) "Uses" view. (b) Data flow.

Figure 4.14: Services interaction (use and data flow) for document version generation scenario.

does not have modifications (*repository services*), to query the links database (*search services*) to know which documents modify it, and to compose document fragments to get the desired version (this task is assumed by the *document management service*). Finally, the required version passes to the user interface service to be presented to the user. This evolution can be followed in figure 4.14.

A global view of the services interaction is shown in figure 4.15. This figure is a *uses* view. A service *S1* is said to *interact* with another service *S2* if it uses service *S2* at some moment during its functioning. Data flow between services follows what has been seen in the explanation of the scenarios. For example, the user interface service and the document access service exchange requests (from user interface to document access) and documents (from document access to the user interface). Figure 4.16 shows a global view of this flow.

4.2.5 Services interfaces

Service methods allow other services to invoke them. The following list includes methods that are relevant to cover needs derived from required functionalities.

User Interface service

- **GetDocument**

  Returns a document (content) for presentation to the user. It receives the document identifier as input.

- **Search**

  Search in library collections. It receives the user query as input. This method permits documents, links or whatever other type of data is available in the library for queries to be searched.

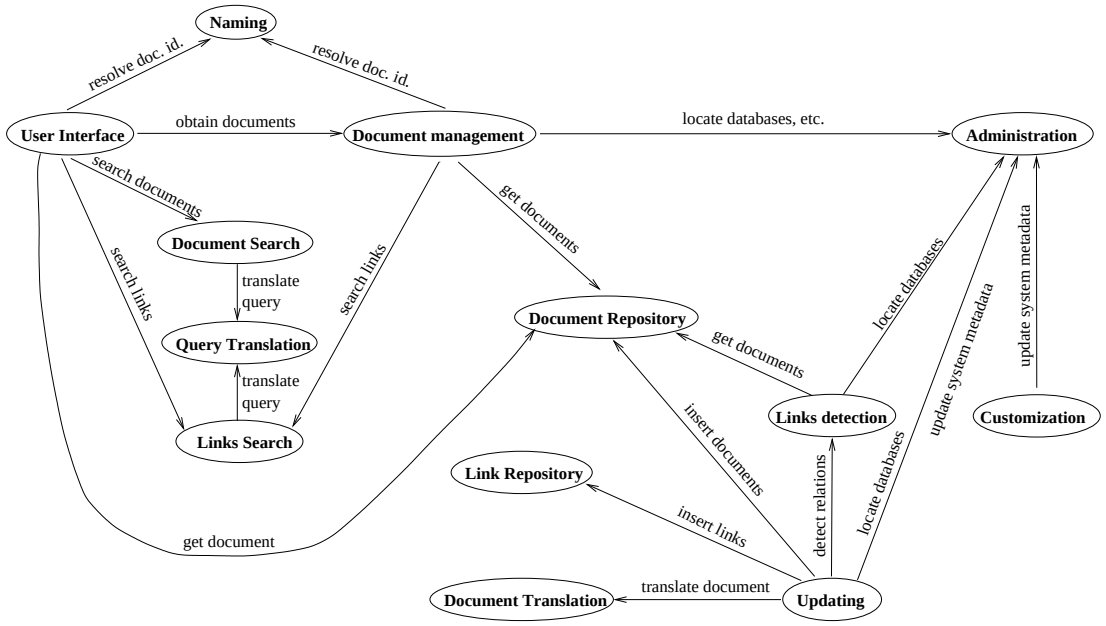


Figure 4.15: Services interaction. Global view.

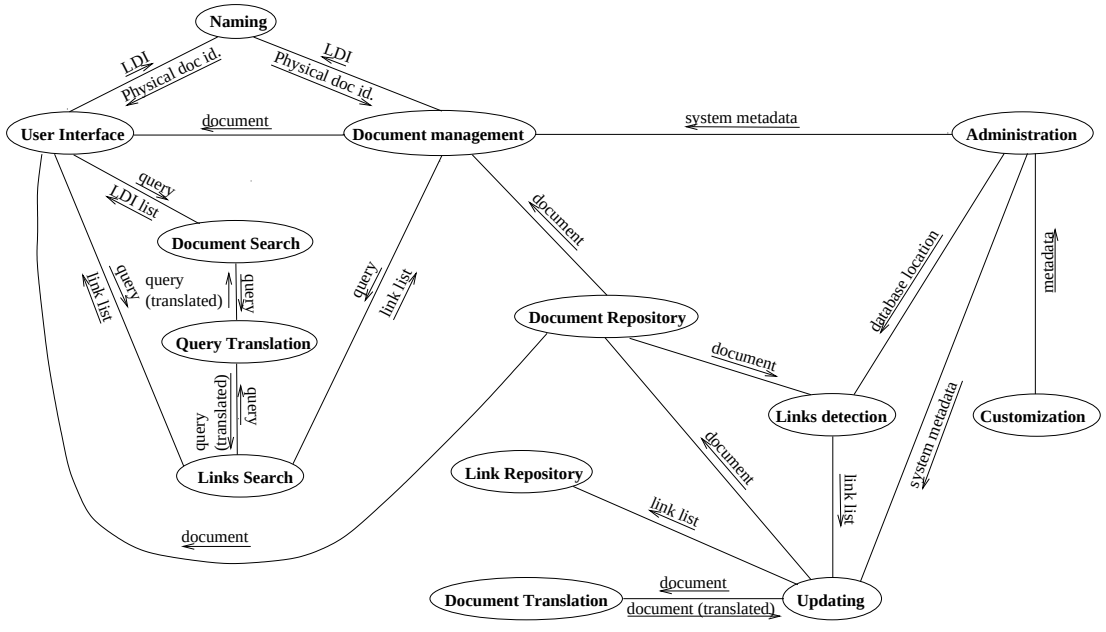


Figure 4.16: Data flow between services. Global view.

- **BrowseCollection**

  Allows the library collections to be browsed.

Document Management service

- **GetVersion**

  It returns a document version, in the conditions (moment, etc.) expressed by the user. It receives the document identifier and the criteria that distinguishes the required version from other versions of the same document as input. If necessary, this method generates the version by composition of documents.

- **GetRelatedInformation**

  This takes a document identifier and attributes that characterize the desired relationships as input. It generates a dossier with related information.

Repository service

- **Insert**

  This takes a document (or link collection) as input and inserts it in the repository.

- **GetDoc**

  This returns a document, whose identifier it receives within the parameters.

Naming service

- **CreateName**

  This takes an instance (location) of an object or service as input and returns a unique name. The newly created unique name and resolution are stored in the name service.

- **ResolveName**

  This takes a unique name as input and returns the set of instances (locations) for that name.

Search Documents service

- **SubmitQuery**

  This searches the document database, with the query received in the input argument. The result is a list of identifiers corresponding to the set of documents that match the query.

Links Search service

- **SubmitQuery**

  This returns links in the system that match the criteria in the input. The criteria are specified as a sequence of pairs (attribute, value).

Links Detection service

- **GenerateLinks**

  This receives a document and its identifier. It generates a collection of links obtained from the input document (links whose origin is in the document). The identifier is necessary to be included in the links.

System Administration service

- **ListServices**

  This lists services in the system, with a mode to access them.

- **ListServiceMethods**

  This lists methods available for a service and a mode to access them.

- **ListCollections**

  This lists collections in the system and a mode to access them.

- **ListDocumentTypes**

  This lists types of documents supported in the system.

- **DocTypeOntMapping**

  This returns a description of the mapping between the vocabulary used inside the text of a selected document type and the equivalent ontology in the system.

- **SearchToolsMapping**

  This returns a description of the mapping between the query language used in system services and the private query language for every search tool.

Customization service

- **InsertService**

  This is used to incorporate new services into the system.

- **InsertCollection**

  This is useful for incorporating new collections into the system.

- **InsertDocType**

  This allows new document categories to be inserted in the system.

Query Translation service

- **Translate**

  This translates a query from the system query language to the private search tool languages.

Document Translation service

- **Translate**

  This translates an input document in an external format to an internal system format. It uses the document class ontology mapping to carry out the translation.

4.2.6 System components

The development phase causes the services in the previous section to be implemented by a set of components (in figures 4.17 and 4.18). These components interact with each other in order to successfully achieve system functionalities. Components assume a series of well-defined tasks, in a way that promotes system scalability and modularity.

The presentation of components obeys the following criteria: first, there are components that are involved in link related functionalities -as they implement tasks that are the main interest in this thesis-; second, there are other components that complete the system in the sense that they are necessary for any digital library in order to be able to function, interact with the user, and maintain coherence in the system data and system functionality.

"Link" components

- **Document Server**

  This component is responsible for serving documents and everything that may be needed to postprocess them (associated links, associated metadata) before presentation to the user. It is a crucial element in the system: it knows what to do in order to obtain all types of documents[5] in the system. The user request passes through this element, as it is able to analyse it and decide the steps needed to successfully serve the request.

  This element implements or invokes the following services: *Document Management*, *Search*, and *Repository*.

- **Links search engine**

  This searches the links database. It represents (abstraction) the engine that queries the links database. It is a wrapper to this engine that allows peculiarities (such as, for example, the query syntax) to be abstracted from particular engines.

  This element implements the *Link Search* service and uses Translation services to translate queries.

- **Links generator**

  This generates links from document content. It takes the input document and analyses its content in order to detect citations, and it creates a link collection that can be inserted in the link library collections. It implements the *Link Detection* service.

[5] *Document* in a general sense: documents from databases, documents generated on demand, answers to user queries, ...

- **Document Translator**

  This translates documents from non-system formats to an equivalent normalised copy that the system is able to manipulate in subsequent operations. This component participates in the insertion process of any document in the library, before any other operation can be done on the document.

  It implements the *Translation* service.

Other components

There are more components in the system that interact with those presented above. These components (or similar ones) are present in most digital libraries, and their presence here shows how link related components can smoothly integrate a digital library.

- **Document query engine**

  This is a proxy for the document search engine, that effectively indexes and searches document collections. This component receives a query and returns a collection of document identifiers that match the query. It implements the *Document Search* service. If needed, it uses the *Translation* service to translate queries.

- **Query translators**

  They translate queries from the system language to local ones. This type of component appears in libraries with language heterogeneity. They implement *Translation* services.

- **User Interface**

  This component offers the interface which allows the user to interact with the system. It offers the user query forms and transforms results coming from other components to present them to the user in a friendly manner. This component interacts with the *Document Server*, to which it forwards user requests and from which it receives results.

  It implements *User Interface* service.

- **Database Update agent**

  This component includes all functionalities that suppose any kind of updating in the system. That is, it is the one that begins the document insertion process, as well as link generation.

  It implements the *Updating* service.

- **Customizer**

  This component permits the system to be customized by adding new services and document types.

  It implements *Customization* services.

The following components are databases, all of which are accessed through *Repository* service methods:

- **System metadata repository**

  Metadata about the system. It is also accessed through *Administration* services. It will be queried by other system components to obtain metadata about the system.

- **Document repositories**

  Document collections encapsulate document databases, metadata related to these databases and methods that allow access to both. Also, methods for updating the database.

- **Link repository**

  The link collections encapsulate the links database, its metadata, and the methods to access their information.

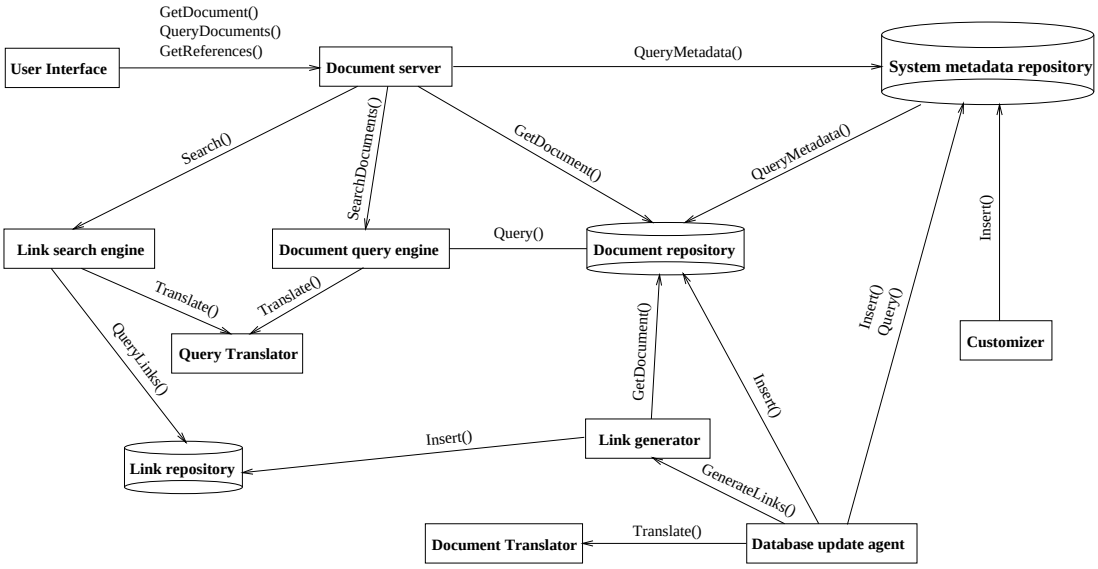


Figure 4.17: Components interaction. "Calls" view.

Equivalences between services and components

The services presented in section 4.2.3 are implemented as a collaboration between the components in this section. This part is an overview of this equivalence -even if not exhaustive-, where only mappings that are not evident are mentioned.

The *Document Management service* is obtained by the interaction between the *Document Server* component -that carries out document transformations-, the *Document Repository* -which provides the documents that will be the target of transformations-, the *Link Search Engine* -that selects links that will be used for document manipulation, and the *Link Repository* where the links are stored.

The *Repository service* is directly mapped to *Document Repository* components, *Link repository* and *System Metadata Repository* components.

The *Naming service* is implemented as a collaboration between *Document Repository*, *System Metadata Repository* and *Document Search engine*. The *Document Repository* is necessary as the resolution of document identifiers is in fact a query to document metadata, executed by the *Document Search engine*. The resolution of identifiers for collections and other elements, such as system components, is done by querying the *System Metadata Repository*.

Search services involve the *Document Search engine* and *Document Repository* on the one hand, and the *Link Search engine* and *Link Repository* on the other hand.

Link detection services imply the action of the *Link generator* component, that will write its outputs to the *Link Repository*.

Document Translation services involve the *Document Translator* that operates on documents to generate new document representatives, the *System Metadata Repository*, from where the ontology mapping for the class of the document that is the object of each translation can be obtained, and the *Document Repository* database where the generated representative document copy is stored.

4.2.7 Components interaction

Components interaction can be presented in terms of scenarios in part 4.2.2, as has been done with services. Only scenarios where links are specially relevant are commented on, in order to keep this chapter from becoming too extensive.

Querying about relations

In this scenario, a user asks the library for information about documents related to a certain document `D` which he/she specifies in the request. The result will be a list of document identifiers that are related to `D`, in the way the user specified in the request.

The user makes the request through the *User Interface* component. This invokes the *Document Server* document access methods, passing the document identifier and relation filtering criteria in the call parameters.

The *Document Server* invokes the *Link Search engine* by passing a query within the parameters where criteria about links are included. Return data from this query is a collection of links that match the query, which the *Document Server* will process to select the information that will be used for presentation to the user: document identifiers, link types. This collection of information is returned to the *User Interface* as a digital document that can be manipulated to adequate it for best user presentation.

To summarise, components involved in this scenario are: *User Interface*, *Document Server*, *Link Search Engine*, and *Link Repository*. Data exchanged between these components can also be seen in figure 4.19. Queries pass from the *Document Server* through the *Link Search engine*; the inverse traversal is done by data that match the query. And the *Document Server* returns documents to the *User Interface* component.

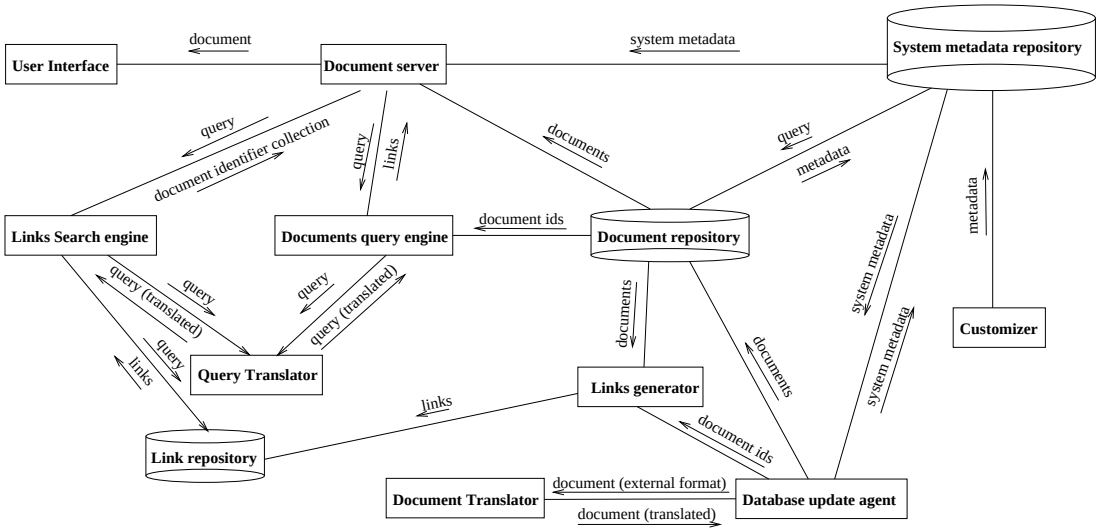
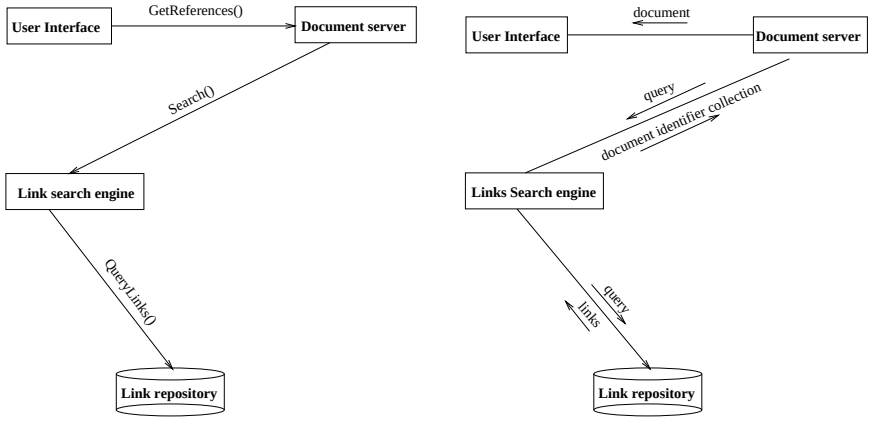


Figure 4.18: Data flow between components.



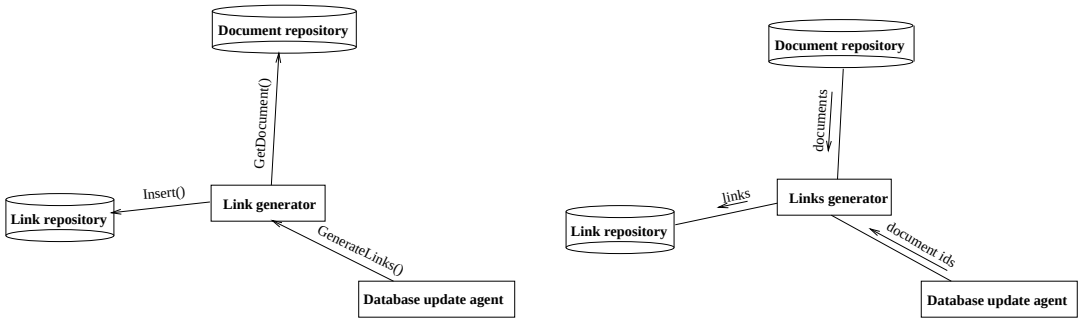
(a) "Calls" view. (b) Data flow.

Figure 4.19: Components interaction (use and data flow) for querying relations scenario.

Link generation

In this scenario a *Database Update agent* starts the interaction, requesting detection and generation of links with origin in a given document `D` from the *Link Generator*. This generation supposes the location of the available copy of `D` and access to it, in the *Document repository*. Analysis of the content of `D` is carried out by the *Link Generator*, which obtains a collection of links ready to be inserted in the link repository. The final insertion of links in the library system is done in the *Link Repository*, whose insertion methods are called by the *Link Generator*.

To summarise, components involved in this scenario are: *Database Update agent*, *Link Generator*, *Document Repository*, and *Link Repository*. Data exchanged by these components can also be seen in figure 4.20. The document to be analysed is retrieved by the *Link Generator* from the *Document Repository*. Generated links are passed from the *Link Generator* to the *Link repository* when calling the latter's insertion method.



(a) "Calls" view. (b) Data flow.

Figure 4.20: Components interaction (use and data flow) for link generation scenario.

Document version generation

In this scenario a user requests a document `D`, at the state it was in at a certain moment, which the user also specifies in his/her request. The user makes this request through the *User Interface* component, which invokes the *Document Server* document access methods, passing the document identifier and version criteria in the parameters. The *Document Server* breaks down the query into a set of subqueries and actions, some of which are achieved by calling some other system component methods. To get the requested document version involves the following steps:

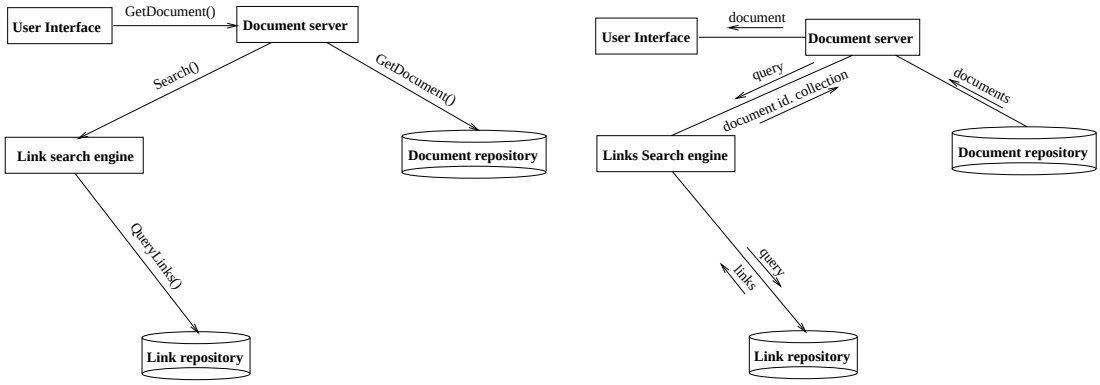
1. Location of the document copy that will be used as the base document to which modifications that allow the requested version to be obtained will be applied, and retrieving it from the concerned repository.

2. Querying the link databases about links related to the input document (in fact, the query is a search for links that express modifications to the base document).

This querying is done by calls to the *Link Search engine*, that will return the collection of links that modify the base document.

3. The application of modifications expressed in links to the base document. This is a process of document composition expressed as a transformation of the base document, carried out by the *Document Server*.

4. Returning the resulting document to the *User Interface*, which will present it to the user.



(a) "Calls" view. (b) Data flow.

Figure 4.21: Components interaction (use and data flow) for document version generation scenario.

To sum up, the components involved in this scenario are: *User Interface*, *Document Server*, *Link Search Engine*, *Document Repository* and *Link Repository*. Data exchanged between these components can also be seen in figure 4.21. Queries pass from the *Document Server* through the *Link Search engine*; the inverse traversal is done by the information extracted from links that match the query. The *Document Server* gets documents from *Document repository* components, and returns digital documents resulting from the operation to the *User Interface* component.

4.2.8 Data architecture

The set of data used in the library is made up of two main groups: data that make up the basic library documents -that the user will access at some time- and metadata -used during system operation, but never seen by the user- about these data: links that express relations between documents, and data that describe documents. There are also metadata about the system itself that are used for correct functioning.

System metadata

System metadata are deduced from services needs.

Document Management services manipulate document and generate new ones from stable documents (and links) in library collections. To get access to them they need:

- information about the location of document and link repositories
- information about the link structure (fields), to formulate queries on links
- to know the equivalence between document identifiers and the location of their digital copies. This equivalence is provided by the *Naming service*.

The *User Interface* service provides users with documents, formatted in a friendly way, as well as with interfaces to use system services. It "translates" user requests to the invocation of convenient methods to other system services. Thus, it needs:

- to know what services are in the system and the methods that it can invoke. It needs information about Search and Document Manipulation services.

Search services allow queries about documents or links to be made. What these services need is:

- in case of language heterogeneity, the location of translators that will translate the query and method that allows it to be done.

The *Link Detection* service analyses documents in order to create links, expressed in a canonical model, that other system services are able to understand. So, it needs:

- to know where the link repositories are
- information that helps it to find the equivalence between citations detected in document content and categories (types) of documents in the system
- the structure (information fields) of links, to be able to generate links that apply to this structure
- information that helps the mapping to be done between manners to characterize a document fragment in natural language used in document content and their equivalent description in the internal system.

The *Document Translation* service obtains a document representative in a format that system services are able to manipulate, from copies in external formats. To carry out this translation, it needs:

- to know the mapping between ways to characterize document fragments in input document formats and characterization for content fragments in the system model.

Query Translation services work with query languages and attribute models. A detailed description of what such a service would need can be found in [16].

Administration and *Customization* services keep metadata about the system and allow new categories of data and services to be introduced in the library. For this, they need:

- to know about system services and components
- to access metadata about each repository

In conclusion, metadata in the system are:

- A list of system services, their location and their methods.

Service name	*Service category*	*Location*	*Service methods*			
<name value>	<cat. value>	<loc. value>	m_1	m_2	...	m_n

- A list of document classes in the system.
- For each document class: the equivalence between input domain ontology expressions used to cite it and the system ontology way to address it. This information is used by the *Link Detection* service to obtain *Logical Document Identifiers* from citations.

Document class	*Term or expression used in citations*			
<class value>	e_1	e_2	...	e_n

- For each class of document: mapping between input domain way to characterize its fragments and the equivalent system manner to do so. This information is used by the *Document Translation* service to obtain a document representative in a system format that guarantees interoperability and the possibility of manipulating and accessing document fragments in a correct manner.

Element name	*List of equivalent expressions*			
<name value>	e_1	e_2	...	e_n

- For each repository: metadata about documents in the repository (classes in the repository).
- The structure of links.

Document architecture

A document is an aggregation of three information elements: document content, metadata that describe it, and document links (shown in figure 4.22).
Document content.
Document content is stored in a digital copy of the document, whose logical structure reflects the abstract document entity semantic structure (which has been the object of a detailed explanation in chapter 2).

Document metadata.
Every document in the system is designated by a unique identifier that distinguishes it from other documents in the library universe. The advantages of such an identifier are several, but for these linking oriented services, the main advantages that have been critical in the decision to use such an identifier instead of a physical locator are as follows:

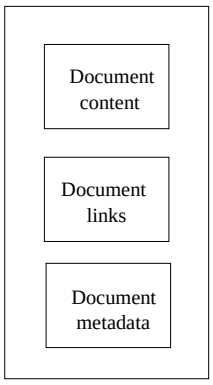


Figure 4.22: Documents are the aggregation of three information elements: content, metadata and links.

- It is possible to deduce a Logical Document Identifier from citations in document content, but it will never be possible to deduce physical addresses from citations. This property is the one that citation detection is based on.
- Links detected from citations will therefore be between abstract document entities, and never between physical copies of the document.
- A logical identifier abstracts documents and their versions from their physical copy location. It describes the abstract entity that a document is, which is helpful when expressing requests such as "get version of document `D`, in its state at moment `t`", where document `D` is designated by its logical identifier.

The document logical identifier follows the criteria used in the DOI standard [94], taking into account that the local part of the identifier is created based on the document class, the document identifier inside it, and -if necessary- the version discriminator.

Example 23. Take the following identifier:

`urn:thisthesisprototypedomain:collection1/D1-13-1980`

This identifier designates a document inside the namespace for this thesis prototype, belonging to `collection1` inside it, and designated inside this collection by `D1-13-1980`. □

Every document identifier *resolves* to a document representative. For every such abstract document there is a set of metadata that describe it, used for the resolution, and shown in table 4.1.

Links.
Links are explained in detail in chapter 3; that is why there is no detailed description here.

4.2.9 System qualities

The architecture presented up to now achieves some properties, some of which can be found in general software architecture guidelines [18], while others are directly inherited

Field	*Field description*
Document Logical Identifier	
Document Physical address	Address of the digital copy stored in the library databases.
Type of document	Class the document belongs to.
Document date	Date of the version the stored document copy corresponds to.
Title	A document description, readable by a human being.

Table 4.1: Logical Document Identifiers.

from requirements specified by Arms [10] for digital libraries, or some digital library architecture designs [45, 25].

The architecture for linking services integration is therefore thought to be *extensible* and *scalable*, which means it allows for new services and elements to be easily added to the system. A service-oriented architecture design -as this one is- results, in all ways, in an *open* architecture (library functionality is available in the form of distinct functional units or services, each of which publishes its operational semantics through its access methods, which makes it feasible to easily add or delete new elements without affecting existing semantics). Additional properties of service-oriented design are that the resulting system is able to integrate new functionality through the implementation of additional services which interact with existing services, that is, *federation* is achieved in terms of services autonomy. One more guideline in this architecture design is strictly restricted to the digital library domain. An important issue in digital libraries is that "*users of the library want intellectual works*" (Arms 1995), which means that the library is responsible for managing the mapping between such abstract entities and the digital document it holds. This property is the one that resulted in the need for document *metadata* and document *naming* services.

Some more quality attributes of the system in this chapter follow:

- *Modifiability.* Changes would be located in services (and components in the implementation phase), such that they would never result in important modifications to the library.

- *Interoperability* in data is achieved by the use of a standard format for all data exchanged by library services, as well as for all data and documents that come into the library databases.

- *Integrability* of its components, that integrate through their interfaces. High interoperability would be achieved by implementing the system with middle-ware designed for this goal.

- *Conceptual integrity.* All services that provide similar functionalities do so in a similar way. For example, all search services have a similar operation: they get the request, they translate it -if necessary-, they perform it on indexes, and they return results -which they have previously filtered and normalised-.

4.3 Discussion

The architecture and interaction protocol is defined in terms of "services", following the most extended model to design digital library architectures [11, 45, 104] and to characterize their functionalities and the distribution of tasks. It is one of the goals in this thesis that the proposal made for linking-oriented services accomplish smooth integration with other digital library services; so, following the services model shows the feasibility of this integration more easily.

The services that make up the main contribution of this chapter are related to links and document manipulation. Other services are included as they are used; more information about them can be found in the references [10, 77, 98]. Basic services are considered in all models and architectures presented, as well as in all protocols. Additional services and protocols are dependent on the library's aim; for example, payment services are meaningless in a digital library that offers free access to its resources. Metadata importance is not ignored in any of the commented references: the Stanford Digital Library project dedicates special protocols to managing metadata, Dienst includes services to access them, and Z39.50 has an Explain facility which is only used to query metadata. To provide an open architecture (that allows integration of additional services without affecting existing services operation) is common to all, except to the document manipulation architecture presented in part 4.1.3; this is conceived as a software unit not to be broken down nor to be allowed to expand to additional functionalities, in order to prevent the cost of support and maintenance due to the use of independent products that interact through interfaces. Therefore, this is the reason for the inexistence of an open protocol associated to it. This establishes a difference with this chapter's proposal, which has integrability and openness as an initial requisite to make the integration of link-oriented services in legacy libraries possible, as well as the future integration of other link-oriented services that could complement these ones, which places it closer in its properties' requisites to the rest of the models commented in section 4.1.

Moreover, as for the possibility of finding a general protocol for document manipulation, this is a very variable field, where "manipulate" can mean different things in different libraries: to split documents, to compose new documents, to obtain new copies in different formats, to create indexes, to create new documents, etc. With this wide range of possibilities it seems very difficult at the moment to reach a normalisation that makes the definition of such a protocol possible.

As for the software architecture, we have chosen, from the available views to describe it [18], those we think reflect more accurately the working method followed. The definition of a set of services implies the characterization of a set of interfaces that make these services available for use by the rest of the system; this leads to the choice of the "uses" view. On the other hand, considering that digital libraries are indeed sets of services that provide access to enormous collections of information, data that contains this information (documents and results from system operations) are decisive when designing a system of this nature; thus the choice of the "data flow" view.

The first view of the library as a set of services changes to become a set of components, the collaboration of which implement these services in the implementation phase. Interaction between these components will be looked at again in the chapter about the

prototype (chapter 5). In any case, the evolution from services to any other software implementation is not a major problem [32].

Research in reference linking in the journal articles domain has been active over recent years. This has resulted in the emergence of proposals for "link services", dedicated to detecting links from article content [67]. These services are included in this chapter linking proposal (they are necessary to achieve a complete or almost complete automation of work with links), under the denomination of "Link detection". This naming is more precise and it is necessary to differentiate this functionality from other link-related services. On the other hand, it is possible to find works that present "link services" with different meanings: services dedicated to keeping links updated consistently with changes in the location of physical resources [97], or services that facilitate the manual creation of links to users [32]. There is also the problem of the persistence of identifiers in changing libraries. The proposed solution is some type of persistent identifier, whose resolution provides the physical address of the desired resource [31, 94]. This is the solution described as *Naming service*, which deals with the document identifiers presented in the data architecture subsection (4.2.8).

Document versions are one of the most interesting problems in digital libraries, about which there are not many references. From all the bibliography on the subject, the only reference that proposes an architecture for a system that deals with document versions is the proposition by Arnold Moore et al. [13], presented in subsection 4.1.3. They argue that an advantage of their system is the efficiency achieved due to the fact that all document manipulation variations are included in a single block of software. The integration of services contributed in this thesis with other services is one of the desired qualities that guides the design of this proposal; this objective places efficiency on a second level of importance with respect to the provision of a set of interfaces for manipulation services that really facilitate their integration with other digital library services.

Every abstract document entity maps into an aggregation of three information objects: metadata that describe the document, the document content, and links related to the document. The reference linking model and NCSTRL's data model associate an identifier (*handler* or DOI) to each document entity, whose resolution returns the addresses of all physical copies of the document available. Metadata that describe the document and help in naming the resolution are part of the document architecture. The document content may come from any of the document digital copies that correspond to different formats. Links are not included as part of document information, as clear separation of links and document content is a more recent reality than would seem likely: There seem to be very few implementations where links are considered aside from the documents they complement. This is due to the little attention dedicated to the exploitation of semantic links, which is related to the other reason that caused links and content to be mixed: most efforts have been dedicated to the creation and maintenance of navigational hypertext. Given that this hypertext is implemented as HTML documents, it is evident that it is an imposition to include links -manually, in most cases- in document content, losing the semantics of the relationship they represent in the insertion process.

An alternative way to model relationships would be to do it in document metadata.

This is, for example, the possibility offered by the element `Relation` of the *Dublin Core* standard [23]. This possibility is not used in this thesis to facilitate the creation of temporal documents: digital documents that flow in the system may vary (same content and different links, depending on the use given to the document). For example, different versions of the same abstract document entity at different moments, share the content (original content to which modifications apply), but differ in links (commonly, modifications applied to a document at a moment t_p later than t_a include modifications valid in t_a plus those done in the time interval $[t_a, t_p]$). Additional reasons that justify this decision on links are given in chapter 3.

It has been decided not to store virtual documents for several reasons: first, it is not the aim of this thesis to show incremental updates of digital libraries; second, even if such a possibility were to be considered, there is evidence (see references [38, 15]) to show that such a policy can bring with it several problems, in terms of volume of documents in the collection, validation of documents that should make up the collections, and manipulation of versions. In particular, for the manipulation of versions, a completely different policy has been chosen: to store the links that facilitate its generation, which can additionally be useful for other aims (such as querying relationships), thus avoiding the mentioned problems.

As has been said, services considered are link-related and basic services used by them. More functionalities could complete the library and thus may expand services: to allow users to add new links, to allow users to detect relationships (for example, by comparing documents), etc. The interaction with users is a means of improving digital libraries, that that can take advantage of the user's knowledge to supersede system limitations.

The proposed architecture is extensible and independent from the actual tools used in the prototype. For example, the Link Search engine caches the peculiarities of the engine that will be effectively used in prototyping. Consequently, it also opens the door to dealing with software heterogeneity if this should become a reality in the library (it acts as a wrapper). Moreover, the proposal could be extended to deal with a case of distribution: component interaction is defined in terms of method invocations, which do not change their interfaces in any way by the fact of being distributed or not; coming back to the links example, the distribution of link databases would be cached by the Link Search engine. This component could deal with the distribution of queries and the integration of results, with transparency to other components in the system.

5

The prototype

Contents

The prototype for this thesis works on a database of legislative documents. It implements some of the components proposed in the chapter about the architecture (chapter 4). Work has focused on components and methods related with algorithms shown in chapters 2 and 3. Emphasis is given to aspects related to obtaining document copies on which structure is correctly reflected. Other aspects that receive special attention are the modelization of relations as links, the use of XML and associated standards for this aim, and finally, the experimentation of version generation on legal documents.

The domain chosen for the prototype is legislative digital libraries. Legislative documents are highly structured, intensively related and rules suffer amendments that result in new versions of the amended rules. Moreover, access to relationships (for example, to obtain all jurisprudence related with a given rule) is important for specialists who use these documents, as well as access to all versions of a document (for example, to understand a tribunal sentence it is necessary to get access to the text of involved rules, as they were at the moment the sentence was made). Finally, in legislative documents, modifications are embedded inside other documents, so that the document to be modified is cited and how it should be modified is expressed later. A document can modify several other documents, and modifications to a given document can come from various sources.

5.1 Component interfaces

This section presents the interfaces for components[1] presented in chapter 4. Input parameters and output data are present for component methods. There is also a revision of some scenarios presented in chapter 4, implemented with the prototype, where some components used for the implementation can be seen, which, while not being dealt with in this thesis, are needed because some components in the prototype are implemented on them; for example, services that analyse documents, search or manipulate data use an XML parser. Task evolution, with data flow between components is the presentation chosen for this revision of the scenarios. Data are the origin of all this work and also the goal of all system functioning; their availability, interoperability and easy manipulation are necessary conditions.

Components are implemented as objects. All documents and data exchanged in the system are XML. Final results are presented to the user via a Web interface; the transformation from XML to HTML -needed because not all browsers are able to work with XML data- is the final step in the scenarios.

In the prototype, the type `XMLdoc` designates a document that suits the definition given in chapter 4, with three internal elements: the metadata, the content and the links that affect the document. It is a string that contains three substrings, all XML data. Types `id-list` and `link-collection` are enumerations of items. The list of identifiers is a string list, where each item is a document identifier. The list of links contains xlinks, each of them agree with the characteristics shown in chapter 3.

The LDI is the document identifier used to obtain the URL that designates the

[1] (UML notation)

physical copy of the document. The library is located in one server, which means that here the library universe, where the LDI is unique, coincides with the server domain.

The components diagram (in figure 5.1) is shown again, and the interfaces used in the prototype are as follows:

Document Server

- **GetDocument(id:LDI):XMLdoc**

  This receives a document identifier (LDI). It returns the original version -the one available in the document database- of the document, where the links are those with their origin in this document.

- **GetDocument(id:LDI, date:String):XMLdoc**

  This receives a document identifier (LDI), and the date of the version of the document requested. It returns the document, where the content is the document version at that `date`, and the links are those with their origin in this document.

- **GetReferences(id:LDI, type:String):XMLdoc**

  This receives a document identifier and the type of references desired. It returns a list of links related to document `id`, all of selected `type`. The type is one of either citation, substitution, insertion or deletion.

- **QueryDocuments(query:String):id-list**

  This receives a `query` as input. It returns the collection of document identifiers that answer the query in the input argument.

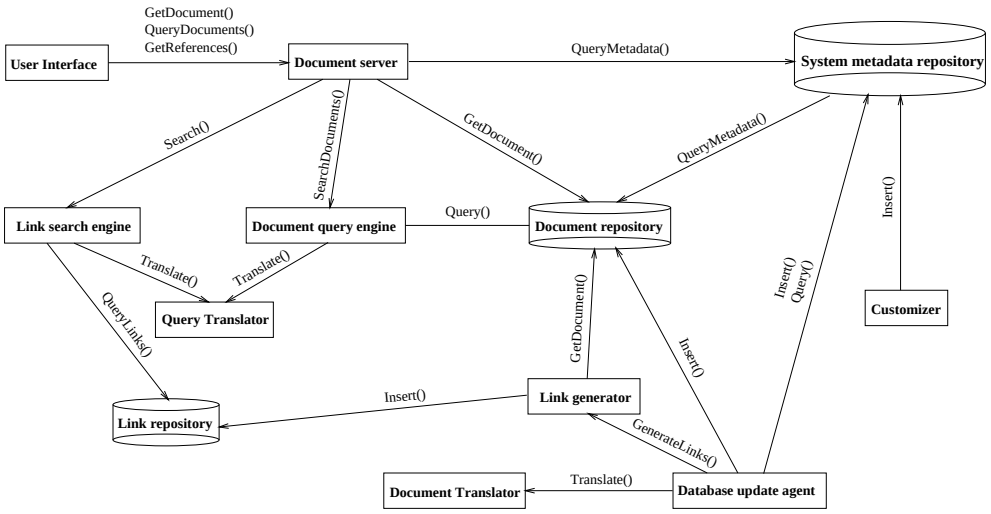


Figure 5.1: Components interaction.

Document Search Engine

This component caches the query engine (an XML indexer [54]) as well as the indexes used by it.

- **Search(query:String):id-list**

  This receives a `query` as input argument. It searches in the document database for documents that match the query. It returns a list of identifiers corresponding to the collection of documents that match the query.

Link Search Engine

- **Search((attribute:String,value:String)+):xlink-collection**

  This receives a query consisting of a list of link attributes and the matching value for each attribute. It returns a collection of links in the system that match all the criteria in the input. The criteria are specified as a sequence of pairs (*attribute, value*) -the sequence presupposes an AND operator in the criteria-.

Link Generator

- **GenerateLinks(id:LDI):xlink-collection**

  This receives a document identifier in the input argument. It generates a collection of Xlinks obtained by analysing the input document. All links in the resulting collection are links with their origin in the document.

System Metadata Collection

- **ListComponents():String**

  This needs no argument. It returns a list of components.

- **ListComponentMethods(component:Name):String**

  This returns the list of methods available for the specified component.

- **ListCollections():String**

  This returns the list of document collections available in the library.

- **ListDocumentTypes():String**

  This returns the list of document classes available in library document collections.

- **DocTypeOntMapping(doc-type:String):Mapdoc**

  This receives a *document class* (`doc-type`) name. It returns an object with two elements: the description of the mapping between ontologies used in document content to characterize document elements and system tags used for those elements, and the hierarchy between elements in that class. This information is used by the component that carries out the document translation.

Document Repository

- **GetDocument(id:LDI):Stream**

  This receives the identifier of a document. It returns the document digital copy, as it is in the repository database.

- **GetDocMeta(id:LDI):String**

  This receives the identifier of a document. It returns metadata about that document.

- **GetDocLinks(id:LDI):String**

  This receives the identifier of a document. It returns all links with their origin in the document. To resolve the identifier it uses the Link Search component.

Link Repository

- **SearchLinks(Query:String):link-list**

  This receives a query and returns the list of links that match the query.

Document Translator

- **Translate(in InputDocument:String, out OutputDocument:String, in OntologyMapping:MapDoc)**

  This receives an input document and the ontology mapping needed to translate the input document to an equivalent document with the same content but different logical structure.

5.2 A revision of the scenarios

This revision of scenarios presented in the chapter about the architecture (chapter 4) considers components in the system that are not designed by this prototype, but which are external applications needed for functioning. Some of these applications (or components) were cached in chapter 4 under services, as is the case of *Search* services, which hide external search engines used to index library collections of documents. This revision is centred on the interaction between components, and activity diagrams, that show task evolution in the system, with data flow between components. Another difference with the presentation of scenarios made in chapter 4 is that here the type (or format) of data or documents that system components exchange during operation receives more attention. For example, it can be seen that the format used to achieve data interoperability in system services (components in the prototype) operation is the XML standard and its associated standards.

The scenarios considered are those implemented in the prototype: retrieval of a document (as it is in the document database), retrieval of a version that needs to be generated, querying relationships, and translating a document.

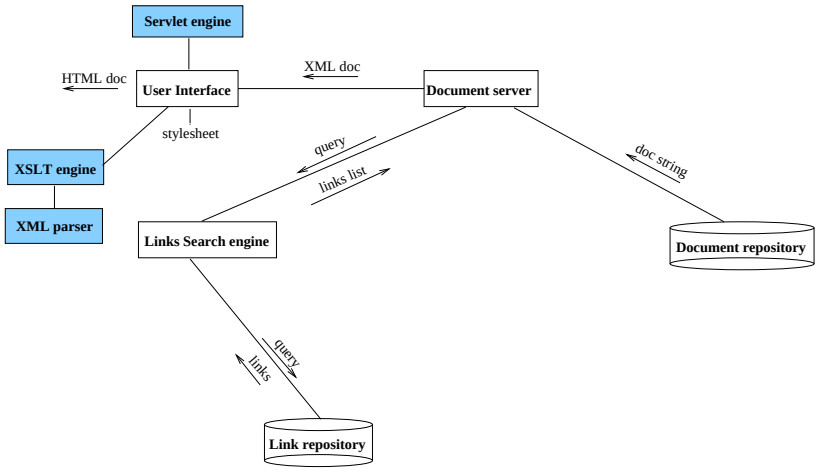
5.2.1 Document Retrieval

In this scenario (figure 5.2), the retrieval of a document (and associated citation links with their origin in the document) is shown. This scenario presents an operation similar to any basic retrieval in a system where documents are presented to the user in a Web browser. There is a "preservation" of citation links with their origin in the document, as they can be used for navigation if this is the wish. However, any type of link (with the document in the origin or in the target) could be retrieved with these methods, by only allowing a selection criteria on links to be specified.

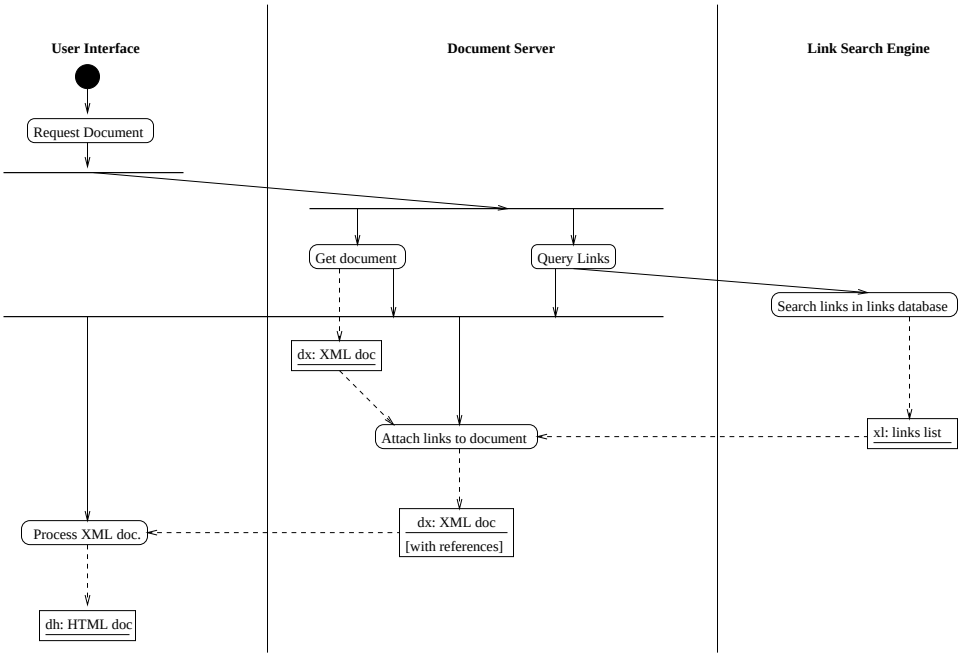
The user request is passed to the Document Server, which reacts by retrieving the document version available in the document database, as well as requesting all citation links with their origin in the document from the Link Search engine. Once the Document Server has both items, it can send an XMLdoc (content + links) to the User Interface, which applies a stylesheet to it to obtain the HTML copy the user can see in the browser.

5.2.2 Get Document Version

This scenario (figure 5.3) shows how to retrieve versions of documents. The user request (that includes the document identifier and date of the desired versions) is passed to the Document Server, which reacts by retrieving the document version available in the document database; it also requests the Link Search engine to get all modification links whose target is in the document that conforms to the date criteria. Once the Document Server has both items, it generates the updated version applying the links to the document version obtained (this method follows the algorithm in section 3.7 of chapter 3). The composition of the version may request the recovery of several other document fragments, steps embedded in the diagram inside the '`Process links on document`' task. The final step corresponds to the User Interface, which receives the updated XML document and generates the HTML copy the user can see in the browser.

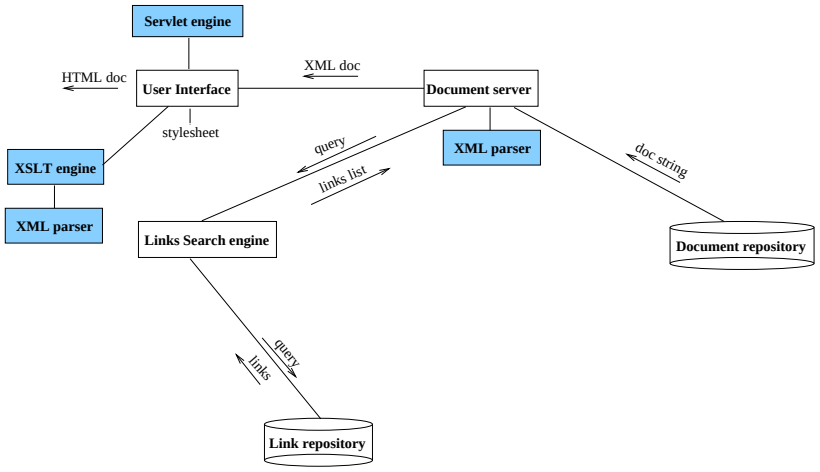


(a) Components and data flow between components.

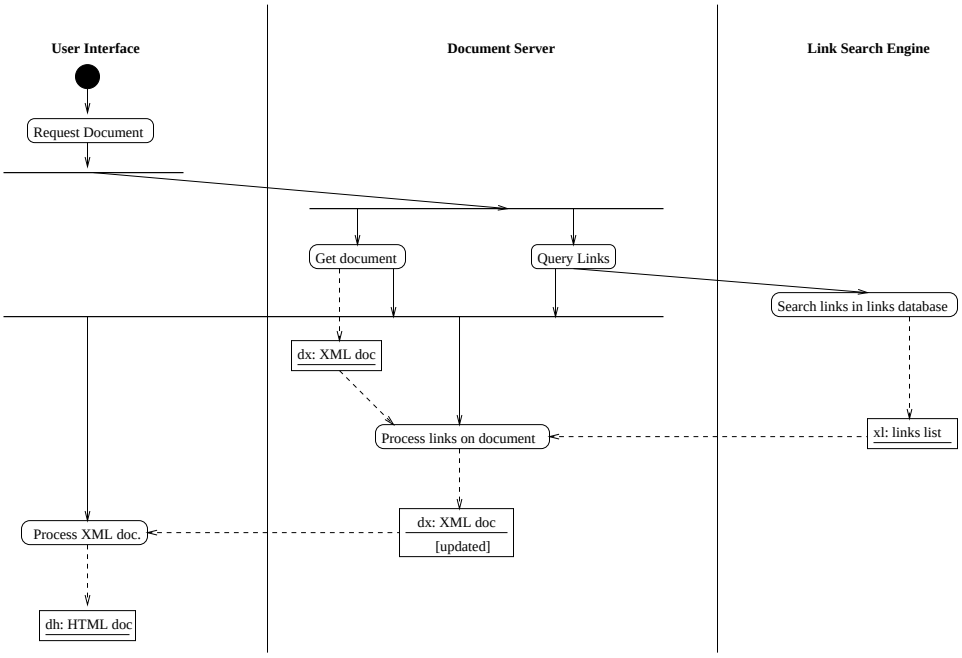


(b) Task evolution.

Figure 5.2: Scenario for document retrieval.



(a) Components and data flow between components.



(b) Task evolution.

Figure 5.3: Scenario for generation of a document version.

5.2.3 Querying relationships

Sometimes the desired information is *documents that reference another document* (for example, jurisprudence related to a given rule). This functionality is described in the diagrams in figure 5.4. As when generating versions of documents, a search on links is needed to obtain the list of documents that are related to a given document. But the criteria to filter links will be different: this time citation links are the ones of interest. Another difference is that the document is not needed to answer the query, so it is not retrieved.

5.2.4 Document search

This method allows documents to be searched (classical queries). An indexer or search engine queries the documents collection and returns the result to the agent who has invoked it -in this case, the Document Server-. The indexer is wrapped in the diagram (figure 5.5) by a generic **Document Query Engine**, which is an abstraction of the search engine peculiarities.

5.3 The document databases

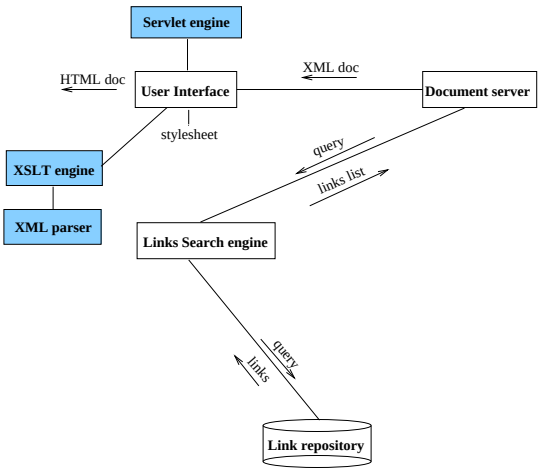
Documents used in the prototype are legislative documents. There are four classes of documents considered: *rules*, *jurisprudence*, *dossiers* and *comments*. Rules and jurisprudence are the most interesting ones from a semantic point of view; these classes have a very well defined semantic structure, which is, moreover, present in citations and is -consequently- crucial for link detection and later version generation using link information. They are, in consequence, the ideal candidates to experiment with the logical structure capture (document translation). Dossiers and comments do not follow a fixed semantic structure (their structure may vary from one author to another); for these documents, where divisions and hierarchical inclusions do not follow rigid rules a more flexible DTD can be used: for example, the TEI standard [48] provides all the necessary utilities to model such documents.

A brief description of the logical structures of these classes is in subsection 5.3.1. The results obtained when translating a database of rules are commented in subsection 5.3.2 .

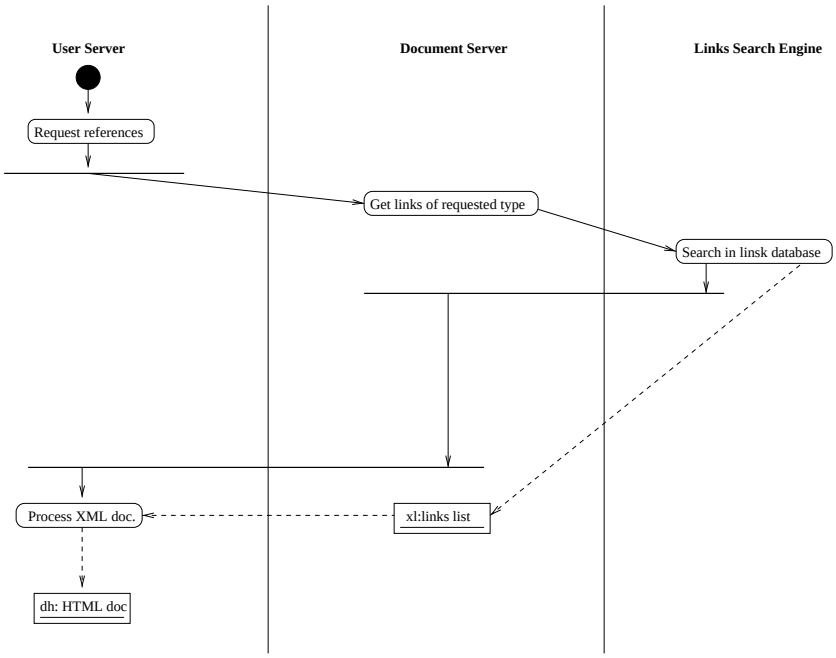
5.3.1 Classes of documents in the legal information library

The legislative digital library hosts four classes of documents:

1. **Rules**: the Spanish Constitution, laws, regulations and decrees.
2. **Jurisprudence**: sentences related to rules, because they are stated with a basis in some of them.
3. **Comments** about rules or jurisprudence.

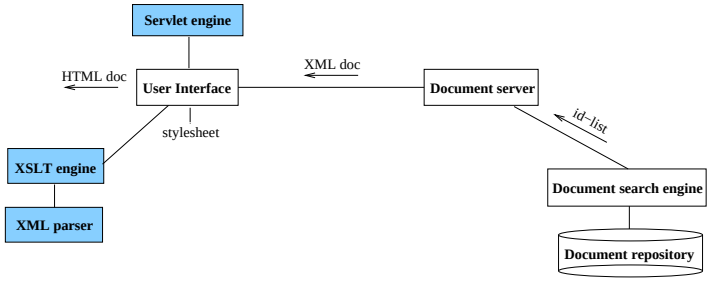


(a) Components and data flow between components.

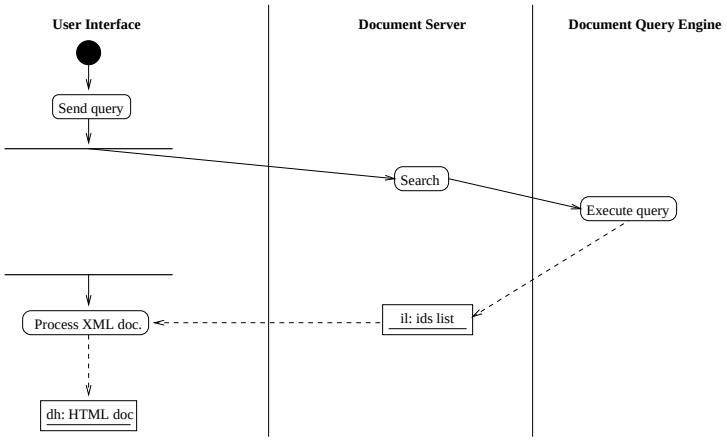


(b) Task evolution.

Figure 5.4: Scenario for querying of relationships.



(a) Components and data flow between components.



(b) Task evolution.

Figure 5.5: Scenario for searching in documents.

4. **Dossiers** that compile references to rules and jurisprudence, classified on some author's criteria.

One more class of document (*headers*) is also in the system, even if it is not intended to be perceived by final users. These documents contain metadata about *base* documents (the four types just described).

Rules are official documents, that always present elements in the set that is explained below, following the inclusion rules also described below. Thus, a rule may have elements in the following set:

- *denominacion.* It is the title of the document.
- *exposicion de motivos.* It is made up of a sequence of paragraphs between the rule title and the first element from the subset below.
- *libro.* It can contain any sequence of elements of type *titulo*, *capitulo*, *seccion* or *articulo*, in any order. There can be zero or more *libro* elements in a rule.
- *titulo.* It can contain any sequence of elements of type *capitulo*, *seccion* or *articulo*, in any order. There can be zero or more *titulo* elements in a rule.
- *capitulo.* It can contain any sequence of elements of type *seccion* or *articulo*, in any order. There can be zero or more *capitulo* elements in a rule.
- *seccion.* It can contain any sequence of elements of type *articulo*, in any order. There can be zero or more *seccion* elements in a rule.
- *articulo.* Any rule must have at least one *artículo* to be considered a rule. There can be one or more *articulo* elements in a rule.

One more element is not explicited by jurists when they describe the semantics of these documents, but is implicit in any document: the *paragraph.*

The element *exposicion de motivos* does not have any semantic keywords that allows its limits to be automatically recognised. It disappears as an element in the class grammar used in the prototype, to be replaced by the possible appearance of one or more paragraphs. Rules grammar (figure 5.6) expresses the inclusion hierarchy, that is also shown in the tree in figure 5.7: all children or descendants of a node are elements allowed inside that type of element; any ascendant of a type of element may be included inside it.

Jurisprudence are also official documents, with a fixed structure, but simpler than that of rules. Only a few elements make up that kind of documents, with no relevant aspects to comment on the inclusion hierarchy, which is shown in figures 5.8 and 5.9.

Metadata (**headers**) associated to legal documents is composed of fields listed below and the grammar that describes this class is shown in figure 5.10. The document identifier, document type and document year are metainformation needed in links, to be used later by the document version generation algorithm. Bulletins (`boletines`) is information considered interesting by jurists. The resulting list of fields is as follows:

- *document identifier* (`ldi`).

$$
\begin{aligned}
< norma > \quad &::= \quad < p >^{*}(< libro > | < titulo > | \\
&\qquad < capitulo > | < seccion > | < articulo >)^{+} \\
&\qquad < disposicion >^{*} \\
< libro > \quad &::= \quad < title >?(< titulo > | < capitulo > | < seccion > | \\
&\qquad < articulo >)^{+} \\
< titulo > \quad &::= \quad < title >?(< capitulo > | < seccion > | < articulo >)^{+} \\
< capitulo > \quad &::= \quad < title >?(< seccion > | < articulo >)^{+} \\
< seccion > \quad &::= \quad < title >? < articulo >^{+} \\
< articulo > \quad &::= \quad < title >? < p >^{+} \\
< disposicion > \quad &::= \quad < title >? < p >^{+}
\end{aligned}
$$

Figure 5.6: Grammar for Spanish rules.

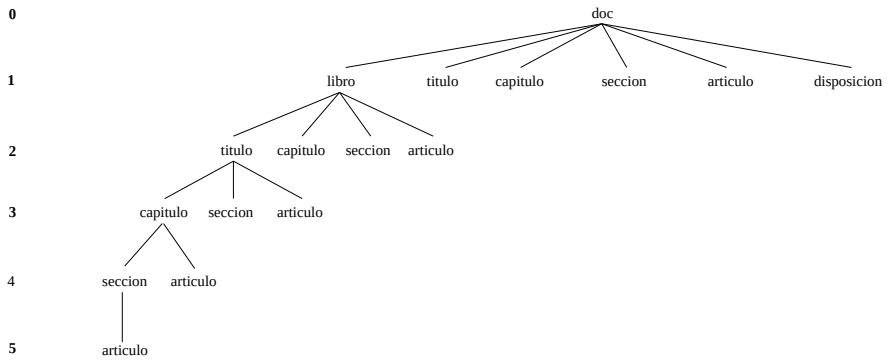


Figure 5.7: Inclusion hierarchy between elements in Spanish rules. Partial representation (the tree is completely expanded in its leftmost branch.)

- document *class* (`type`).
- *title* of the document (`titulo`).
- *year* of publication (`año`).
- *bulletins* where it appeared (`boletines`). This field is interesting for rules.
- *date* of the document (`fecha`).

5.3.2 Translating documents

The translation algorithm in chapter 2 has been tested on a set of Spanish rules, coming from several public servers. These input documents are marked up with different tags, depending on their origin. For example, while some documents presented a sequence of tags `<center><h3>` before the *articulo* semantic elements, others delimited such a type of elements with a `<p><b>` sequence of tags. A first step consisted of the elimination of all elements that were not information present in the abstract document

$$
\begin{aligned}
< jurisprudencia > ::= & < inicio >< fund-hecho >? < fund-derecho >< fallo > \\
< inicio > ::= & < p >^{+} \\
< fund-hecho > ::= & < p >^{+} \\
< fund-derecho > ::= & < p >^{+} \\
< fallo > ::= & < p >^{+}
\end{aligned}
$$

Figure 5.8: Grammar for jurisprudence.

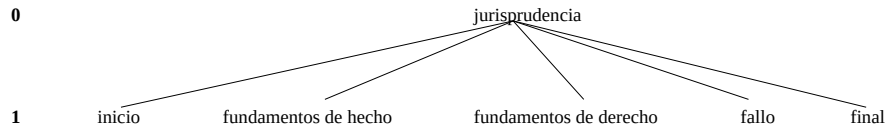


Figure 5.9: Inclusion hierarchy between elements in Spanish jurisprudence.

(links, indices, etc.). They were translated to obtain equivalent documents with the same content but tagged in accordance with the grammar in figure 5.6. This way, the input set of documents to the translation algorithm complied with the input requirements explained in section 2.4 of chapter 2 (an example of the application on a document from the prototype was also included in this chapter). The translation was tested on a set of 1665 documents. From these documents, the algorithm obtained successful results with 1583 of them; that is, it was able to obtain a semantically tagged document in 95% of cases. Documents that are not well treated by the translation algorithm have one of the following characteristics:

- They have tables that are not XML well-formed.
- They have input characters or entities not recognised (`&icute;`, `&ordm;`, etc.).
- They do not have a root element.

The program does not consider tables and figures, which may need special consideration. It only deals with textual documents.

$$
\begin{aligned}
< cabecera > ::= & < ldi >< type >< titulo-doc >< año >< boletines >? \\
< boletines > ::= & < boletin >^{+} \\
< boletin > ::= & < numero >< fecha >< pgnas > \\
< fecha > ::= & < dia >< mes >< año >
\end{aligned}
$$

Figure 5.10: Grammar for metadata.

5.4 Relationships and links in the prototype

Legal documents are densely related. Jurisprudence has citations to rules, comments have citations to any other type of document, and rules cite other rules. An even more interesting aspect of these documents is that some rules modify other rules, and that this modification consists of two adjacent information items:

1. A citation to a document to be modified, where the affected fragment is clearly identified, and which explicitly says that the cited fragment is modified, eliminated or whether it must undergo an insertion.

2. The fragment that has to be inserted or that will substitute the target of the modification comes just after the citation.

These characteristics make these documents the ideal databases to experiment with citation relationships, modifications and version generation. An example is presented in subsection 5.4.3.

5.4.1 Mapping of link fields to XLink attributes

The mapping of links fields in table 5.1 to XLink fields is direct. The document identifier and fragment locator are expressed in the `xlink:href` attribute (the internal locator is the XPointer). Also, the type of relationship is expressed with the role (`xlink:role`) every resource plays in the relationship. These two attributes come from the XLink specification -which explains the use of the namespace `xlink` in them-. Attributes without prefix namespace are locally defined (local to the library namespace): document class (`doctype`) and document date (`date`).

Link field	*XLink attribute*
Document ID Internal locator	xlink:href (*)
Document type	doctype
Date	date
Link type	xlink:role (*)

Table 5.1: Mapping of link fields to XLink attributes. Attributes with an asterisk are predefined in the XLink namespace. Attributes with no asterisk have locally defined semantics.

The links DTD reflects the composition of the links explained: every link has an *origin* vertex and a *target* vertex. Each link is represented by an element `enlace`, that has three elements: the origin (`origen`), the target (`destino`), and the arc (`arco`) that connects them.

The element `origen` is the link's origin: the resource that contains the citation or modification. It may be an element or a text fragment inside an element. The element `destino` is the resource referenced or affected by a modification or citation: the link's target. It may be an element or a text fragment inside an element. The element `arco` establishes edges between vertices. Arcs can be of different types, corresponding to the

different kinds of relationships between the vertices of the graph. There are as many different types of arcs as there are different types of relationships.

The attributes for each of these elements are explained in tables 5.2, 5.3 and 5.4. The link type is derived from the role the origin of the link plays in the relationship; therefore, the type of the link is expressed in the origin element's role attribute.

Attribute	*Value*	*Description*
xlink:href	?	Document ID + fragment locator
xlink:type	'locator'	Resource of an extended out-of-line link
xlink:role	('citation' | 'substitution' | 'insert' | 'delete')	Type of link
string	?	String inside the origin element. It is the real origin in citations.
date	?	Document date
doctype	?	Document class

Table 5.2: *Origin* attributes. A '?' character means that any string value is accepted.

Attribute	*Value*	*Description*
xlink:href	?	Document ID + fragment locator
xlink:type	'locator'	Extended out-of-line link
xlink:role	'target'	It is the target of the link
string	?	String inside the target. It is the real target.
date	?	Document date
doctype	?	Document type

Table 5.3: *Target* attributes. A '?' character means that any string value is accepted.

Attribute	*Value*	*Description*
xlink:from	?	Origin of the arc
xlink:to	?	Target of the arc
xlink:type	'arc'	It is an arc element

Table 5.4: *Arc* attributes. A '?' character means that any string value is accepted.

5.4.2 The influence of document type on document relationships

The nature of documents can determine the type of relations permitted between two documents, by forbidding a certain type of relation between two classes of documents (for example, documents of class A cannot modify documents of class B).

Types of documents in the legal information system are

1. Rules
2. Jurisprudence
3. Bibliography
4. Comments, notes, articles and others

The relationship restrictions coming from document classes are as follows:

- Rules can cite and modify other rules.
- Comments, jurisprudence and bibliography can cite, but they cannot modify rules.
- Jurisprudence, bibliography, and other documents can only have citations to documents of some of these classes.

That is, in the versioning of a rule, only other rules can participate as modifiers.

5.4.3 An example

The example considered was presented in section 3.7 of chapter 3. Input documents to the updating process are the rules shown in figure 5.11:

- A source document, `lo2-1980.xml`, whose first element *articulo* has to be replaced. Its source text is in figure 5.11(a).
- A modifier document, `l13-1986.xml`, which contains the element that will replace the first element *articulo* in `lo2-2980.xml`. The replacing element is the first *articulo* inside the first element *disposicion*. The source text for this document is in figure 5.11(b).

The output document of the modification process -which can be seen in figure 5.11(c)- is a new version of the source document, where the first node-set that constitutes the element *articulo* has been replaced by the first element *articulo* inside the first element *disposicion* in document `l13-1986.xml`; the rest is unchanged. Figure 5.12 shows the effect of the process in the document tree. The link that models the modification can be seen in figure 5.13.

The link for the example -in figure 5.13- is a relation from an ORIGIN (vertex that replaces) to a TARGET (vertex to be replaced). The XML representation for this link can be seen in figure 5.14. As already explained, the element ORIGEN corresponds to the origin link vertex, while the element DESTINO represents the target link vertex. The element ARCO represents the link between ORIGEN and DESTINO, as required by XLink[2].

A database of xlinks has been used to store the link graph information. The availability of this information as XML data has been useful for querying the links during

[2]Mandatory XLink attributes (for example, `xlink:type`) do not appear in the figure as they are defined with default values in the links DTD.

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<doc>
<articulo id="a1"><title>Artículo Primero. </title>
<p>El referendum en sus distintas modalidades, se celebrará de
acuerdo con las condiciones y procedimientos regulados en la
presente Ley Orgánica.</p>
</articulo>
<articulo id="a2"><title>Artículo Segundo. </title>
<p>Uno. La autorización para la convocatoria de consultas populares
por vía de referendum en cualquiera de sus modalidades, es competencia
exclusiva del Estado.</p>
</articulo>
</doc>
```

(a) Source document for the example: *lo2-1980.xml*

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<doc>
<p>Ley 13/1986, de 14 de Abril de 1986, de Fomento y Coordinación
General de la Investigación Científica y Técnica</p>
<p>Don Juan Carlos I,Rey de España.</p>
<disposicion id="da"><title>DISPOSICIONES ADICIONALES. </title>
<p><a>Undécima.</a>
 1. Quedan modificados los artículos 1.°, 4.° y 8.° de la Ley
Orgánica 2/1980, de 30 de abril , que quedarán redactados en la
forma siguiente :</p>
<articulo id="da111"><title>Artículo 1.</title>
<p>Con la denominación de Instituto de Astrofísica de Canarias se crea
un Consorcio Público de Gestión , cuya finalidad es la investigación
astrofísica .</p>
<p>El Instituto de Astrofísica de Canarias estará integrado por la
Administración del Estado, la Comunidad Autónoma de Canarias la
Universidad de La Laguna y el Consejo Superior de Investigaciones
Científicas .</p>
</articulo>
<articulo id="da112"><title>Artículo 4.</title>
<p>El Consejo Rector estará integrado por el Ministro de Educación y
Ciencia , que actuará como Presidente; un Vocal en representación de la
Administración del Estado, que será nombrado a propuesta del
Ministerio de la Presidencia , y tres Vocales más en representación de
cada una de las restantes Administraciones públicas y Organismos que
se relacionan en el artículo 1.° Formará parte del Consejo Rector,
asimismo, el Director del Instituto , que será miembro nato.</p>
</articulo>
</disposicion>
</doc>
```

(b) Modifier document for the example: *l13-1986.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<doc>
<articulo id="da111"><title>Artículo 1.</title>
<p>Con la denominación de Instituto de Astrofísica de Canarias se crea
un Consorcio Público de Gestión, cuya finalidad es la investigación
astrofísica .</p>
<p>El Instituto de Astrofísica de Canarias estará integrado por la
Administración del Estado, la Comunidad Autónoma de Canarias la
Universidad de La Laguna y el Consejo Superior de Investigaciones
Científicas .</p>
</articulo>
<articulo id="a2"><title>Artículo Segundo. </title>
<p>Uno. La autorización para la convocatoria de consultas populares
por vía de referendum en cualquiera de sus modalidades, es competencia
exclusiva del Estado.</p>
</articulo>
</doc>
```

(c) Modified document for the example: new version of *lo2-1980.xml*

Figure 5.11: Version generation. Input and output documents.

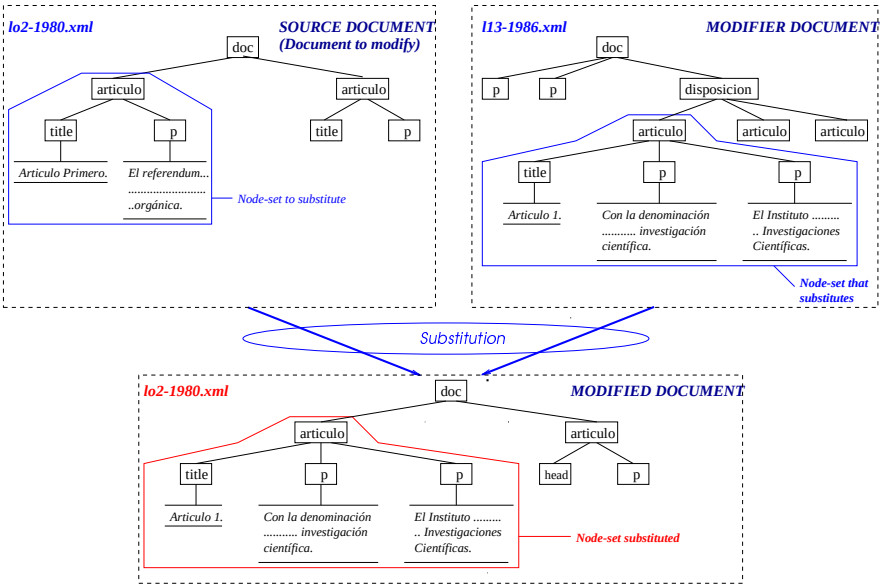


Figure 5.12: Element substitution, based on links. The first element *articulo* in the source (*lo2-1980.xml*) document is substituted by the first element *articulo* inside first element *disposicion* of the modifier document. The result is a new version of document *lo2-1980.xml*.

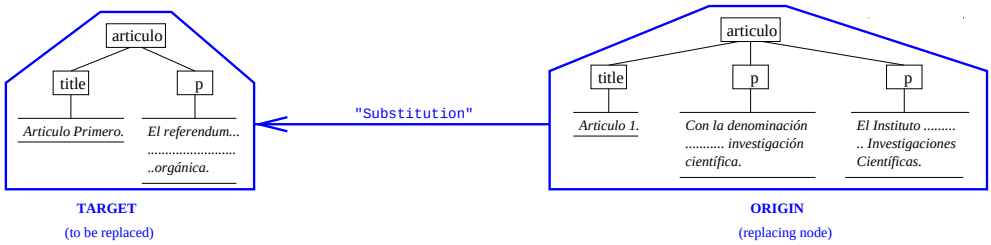


Figure 5.13: "Substitution" link. The ORIGIN will replace the TARGET when generating a new version of the source document. The ORIGIN is a subtree of the source document made up of the element `articulo` and all its descendants. The TARGET is the subtree in the modifier document tree whose root is the first element `articulo` inside the first `disposicion`, as can be seen in figure 5.14.

```
<ENLACE>
<ORIGEN  xlink:href= "l13-1986.xml#xpointer(child::disposicion[1]/articulo[1])"
         xlink:role="substitution"
         date= "1981"
         doctype= "norma"   />
<DESTINO xlink:href= "lo2-1980.xml#xpointer(child::articulo[1])"
         xlink:role="target"
         date= "1986"
         doctype= "norma"   />
 <ARCO    xlink:from="substitution" xlink:to="target"
         xlink:show="undefined"    xlink:actuate="undefined"/>
</ENLACE>
```

Figure 5.14: Text for the example link.

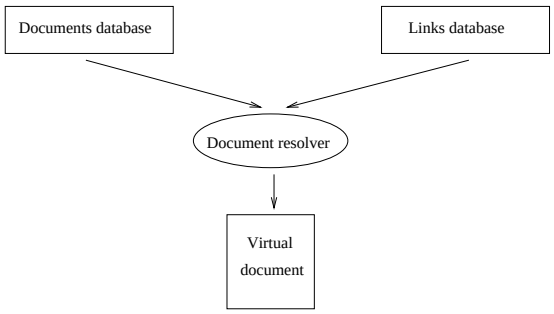


Figure 5.15: Virtual document generation.

the generation process as any other XML data can be queried, thus querying the link graph.

The type of links has been modeled in the links origin, given that the type of a link (citation, modification) in legal rules comes, in fact, from the type of citation detected in the origin document. Also, the citation link and geographically adjacent modification have the same target, but not the same origin. The origin of the citation link is the string that "cites" the target, while the origin of the modification link is the node-set that has to replace the target.

5.5 Version generation

Version generation has been tested on rules, as they are the class where modifications are relevant.

5.5.1 Data types in the generation process

Two types of databases are involved in virtual document generation. Figure 5.15 is an extraction from the components architecture in chapter 5, focused on databases used for virtual document generation -the *Document resolver* represents the collaboration of components that generates document versions (*virtual documents*)-:

- *The documents database*, that contains documents in the collection. The source document (the one that is modified) and documents that contain modifications to it, are inside this database.
- *The links database.* This database contains links that express modifications between documents. Each link maps a portion of a source document to some portion of some other document holding the target text of the modifications.

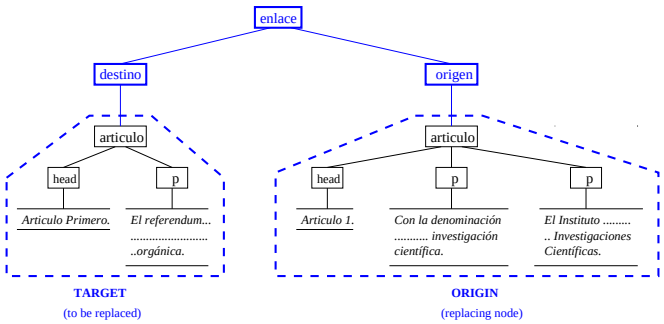


Figure 5.16: Elements in link variables. The variable has only one `enlace` element. This element has itself two elements: `origen` - that contains the subtree which is the origin of the substitution link-, and `destino`, which has the link target node.

5.5.2 Complete substitution algorithm

To treat a document in order to obtain new versions from substitutions, it is necessary to have information about the modifications that affect the document. Information about whether a node is affected by a modification is kept on links. This means that to recognise nodes affected by substitutions, a phase of search on links has to precede the source document treatment. That is, the complete process that leads to a modified document has two steps:

1. Links treatment
2. Links application to source document

Step 1: Links treatment

This phase consists of the creation of a links variable that will be used during source document nodes treatment. The links variable is created from links databases. A search returns a collection of links that contains links related to the source document.

The algorithm 4 filters the links and retrieves links vertices to insert them in the links variable. As a result, the variable SUSTITUCIONES contains a list of `enlace` elements. There is an example in figure 5.16.

Algorithm 4 Links variable creation

Inputs: SUSTITUCIONES: empty links-list; l:links-collection; D:document
  for all links of type "substitution" with origin in D **do**
    recover origin node and create an element *origen* with it;
    recover target node and create an element *destino* with it;
    place both elements in an element *enlace*;
    add this *enlace* to SUSTITUCIONES;
  end for

Step 2: Links application to document

The algorithm 2 in chapter 3 applies the substitution links to documents. The links considered are in the variable created in step 1.

5.6 Discussion

The system

This prototype only implements services needed to achieve the main goal: to provide link-oriented services. Among classical services, retrieval of documents and link searching are implemented. Navigation and document searching would improve the prototype; a navigational hypertext can be constructed from the citation links graph.

The prototype includes the following components: Document Server, User Interface, Link Search Engine and Document Translator. They have been implemented as objects and the user interface is available via a Web interface, controlled by a servlet. The Document Server implements methods for version generation and document retrieval. Qualities of the prototype are:

- *Portability.* Object oriented technology to implement components guarantees system design independence from underlying platforms. Concerning the implementation, as most components are Java, the portability is inherited from Java portability.

- *Reusability.* Some components in this system can be taken from existing implementations (that is, search tools and collection access services). Also, components designed for this system could be reused by other systems; not in a unit granularity (for example, links cannot be used or searched if there is no links collection), but without the need to incorporate all the system in order to use some functionalities.

The prototype library is centralised in one server, which simplifies the manipulation of document identifiers and their resolution. A case of multiplicity of collections or distribution would be treated in a similar manner (the same components, multiplied), on the same basis as for the identifiers, but the name services would become more complicated.

Document translation

Document translation has been tested on a set of legislative documents obtained from several public servers. Source documents were HTML pages tagged with criteria varying on server provenance[3]. Before using them, a step of "cleaning" was needed to adapt them to the algorithm translation input requirements. The requirement to be well-formed XML documents was added to the algorithm as the application that implements it works on an XML parser. The input document copy is analysed sequentially by the parser, which passes items of type "element", "text", etc. to the application, according to the specifications of the SAX 1.0 API[4]. Ontologies mapping and hierarchical inclusion

[3]Procedence were servers of various Spanish ministers
s (http://www.igsap.map.es/cia/dispo/Ibe.htm, http://www.map.es/gobierno/legisla) and the public server "La Ley" (http://www.laley.net).

[4]Available at http://www.meggison.com/SAX.

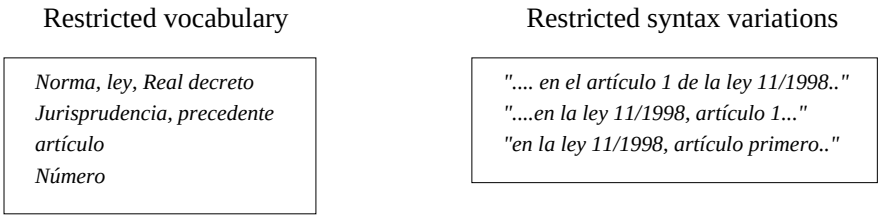


Figure 5.17: Variations in citations found in Spanish rules. Syntax and vocabulary are rigid. For example, there are not many variations between the three manners to cite an *article* shown in the list to the right.

rules between elements for each document class are input parameters to the translation method.

Link querying and document version generation

Link querying and version generation are also implemented on an XML parser. Availability of documents and links in XML, directly provides a language to address document fragments (XPointer), and another language to manipulate them (XSLT). The tree vision of XML documents matches the tree model and recursive treatment of documents carried out for versioning. Version generation has been tested on a set of legislative documents of *rule* class, which are the legislative documents where it is easier to find historical modifications. It works with documents with exact modification overlapping on a node-set. Partial overlappings have also been tested with success. On the other hand, transitive modifications require greater flexibility of tools that manipulate XML than those available at the moment. However, the fast evolution of these tools, with the progressive stability of XML and associated standards leads to the idea that the desired flexibility is not far off[5].

Citation detection

Modifications in legislative texts are geographically close to citations and can be found if detecting citations. Linguistic and syntactic rigidity (figure 5.17 shows some examples of citation variations taken from Spanish rules) in these texts eases the automatic detection of citation [121] by comparison with other contexts more "flexible" in their language structures. That allowed a citation detection to be implemented by recognising keywords inside text. To detect citation automatically helps automatic link creation. However, we note that the citation detection problem is not them main goal of this thesis, and that in the prototype it suffices to show the feasibility to integrate such components in the system and that -at least in the prototype domain- it is not impossible for that detection to be automated.

[5]The link database conforms with the XLink Working-Draft of February, 2000, and the Working-Draft of December 1999 of XPointer. The candidate recommendations that supersede these do not suppose great changes in the implementation (mainly, a one by one mapping).

6

Conclusions

Contents

The area of application of this thesis is digital libraries; structured documents, relationships between documents and the generation of new documents are the aspects of main interest. The aim is to be able to exploit relationships between document fragments beyond mere navigational hypertext, and during the work some other goals have been presented as a necessity to accomplish the main aim. The first requirement is to dissociate the abstract document entity from its digital copies; this places us in the right position to distinguish the abstract document structure and identify our first aim: to obtain a digital copy of the document whose logical structure is an image of the abstract document structure. This goal is also influenced by the fact that the relationships that are mainly of interest are citations and modifications between documents -with a special interest in modifications that come close to a citation to the modified document fragment-. Citations are mostly made in terms of the abstract document structure, addressing the cited document fragment by its relative position in the document logical structure tree. That is, there is a second need: to obtain a digital copy of the documents whose logical structure reflects the abstract document structure.

Once this semantically structured digital copy is available, we can proceed with our next goal: to query relationships, and to automatically generate historical document versions. Querying relationships is possible if they are expressed in terms of links, which -if available in a link database- can be queried like any other document. This database is in fact the implementation of a link graph with three main types of heterogeneous links: structural links (in the document logical tree structure), citation links and modification links. The generation of new documents, and precisely document versions, has been done with a traversal in this graph, where nodes are document fragments addressed by their position in the document tree.

6.1 Contributions

Links have been used in this thesis as first class information objects. They have been taken out of document content and given the relevance they deserve as the owners of information that allows information to be enriched beyond information found in document content. We have enriched the services in digital libraries with services to translate digital document copies to semantically structured copies, and link-oriented services (services where links are crucial): citation detection and document version generation. We have gone beyond document composition from links that directly express composition rules, to an exploitation of the link graph to "deduce" composition rules that are implicit in the link graph, but which nevertheless need to be recognised.

Special attention is given to document versioning. There are three main aspects of interest related to document versioning: detecting, representing and querying changes. The choice for change detection is to detect changes inside document content instead of comparing documents. The way to represent changes is as links between the modified document and the modifier document. Querying changes is therefore querying links. The existence of fragment identifiers is not assumed. The solution is applicable to structured documents, which also allows document fragments to be addressed unidirectionally, and is not dependent on arbitrary ways to assign identifiers.

It has been taken in account that modifications come mostly as part of some document content, which means they are in the document databases, and that to isolate them as first class documents (or individual information pieces) would require a replication of information and the integrity of the document database to be broken, as these information pieces cannot be considered as document entities by themselves. The chosen solution is therefore general: it can also be applied to simpler situations when modifiers are first class objects.

Every document entity is the compound of three information pieces: metadata that describe the document entity, the document (content) as it was created by its author(s), and links related to the document. The advantages of considering these three information objects as individual pieces of information are several. First, the content can be manipulated and queried without affecting metadata and links. Next, metadata and links can be queried separately from the document content. Moreover, links can be shared by all documents involved in the expressed relationships: queries and graph traversals can be made in any direction (from origins to targets and vice-versa).

The algorithm proposed in chapter 2 captures the semantic logical structure of a document instance, the target class grammar (DTD) being known.

The great advantage of the algorithm is that the generated copy is semantically tagged and its logical structure is an exact match of the abstract document entity structure. Not only preserving the semantics is important, even if it is by itself a great advantage: having the structure in a digital document copy is also decisive to be able to establish the connection between citations and the structure of the document copy stored in the database. The algorithm makes a sequential traversal of the input document copy, recognising semantic elements in the same way as a human does. Moreover, it is the safest way to analyse a document sequentially whose size is unknown,thus avoiding having to worry about this attribute. It is not possible to obtain a general algorithm that works on any document. Lack of knowledge about tagging in input documents and their inclusion hierarchy means some decisions have to be made that restrict the algorithm generality; the decisions taken have been influenced by the origin of documents used in the prototype. The fact that input documents to the prototype implementation were HTML pages coming from different servers influenced the decision to forbid the semantic element from nesting inside the HTML element content: the input documents never presented such cases, and by contrast, had variable tag nesting.

The mixed view of a document as both a tree and a set of node sets is emphasised. Each set of nodes is addressed by its root and make up vertices in the relationship graph. These characteristics of structured documents were used to allow all versions of a document to be stored. The link graph adds to structural relationships between document fragments' semantic relationships (citations, modifications). It can be queried, navigated and subgraphs can be extracted to generate new versions of documents. This allows to go beyond the explicit expression of rules that guide document composition to a process where they are calculated during the composition process, which is a graph traversal whose selection is a recursive traversal of the original document version available in the library database.

6.2 Related work

An area where citation relationships are relevant is the reference linking domain. They coincide in that the relationships considered come from citations inside documents to documents that can be merely cited, or cited and modified. Citation detecting is a shared problem (in which some future work in this context is to be done), mainly because work in the reference linking domain is especially concerned with the extraction of a document identifier -independent from the document's physical implementation, but related to the abstract document entity- from the citation detected in document text. Citations related to modifications are only in that category (and citations in the legal domain, which is the area chosen for this thesis prototype always comply with this property). This thesis also agrees with reference linking works in the importance of citation links for user navigation. Aside from this advantage, other ways of taking advantage of the valuable information embedded in links with different goals in mind are also of interest. The thesis is concerned with access to the internal logical structure, which is not a general requirement in the reference linking domain. Links between document fragments are heterogeneous: they contain information about citations, but also about modifications that are expressed inside document content. The modification link is not between the citation and the cited document fragment, but between the modifier document fragment and the modified one (which coincides in this case with the cited one).

6.3 The prototype and the technology

A prototype has been implemented to test the proposals made. Legislative information is semantically structured, densely related, with frequent citations and modifications inside documents to other documents: for example, rules are frequently modified, with the result of new versions of modified rules.

The translation of documents (the generation of semantically structured document copies presented in chapter 2) has been tested on a set of legislative documents obtained from several public servers; tagging is heterogeneous, varying from one server to another. Output documents from the document translation process are the base documents whose fragments make up the vertices of the link graph, from which rule versions have been generated.

Document version generation and link querying use an XML parser. The availability of documents and links in XML format provides a language to address document fragments in XML documents (XPath) and a language for document manipulation (XSLT). Moreover, the XML model of documents as tree structures, adapts perfectly to the hierarchical view of documents necessary for the recursive treatment on version generation.

The implementation of the link database follows the working draft of February 2000 [119] for XLink, and the working draft of December 1999 [116] for XPointer. At this moment, there are later versions ("candidate recommendations"); but the changes introduced by these new versions are not relevant enough to suppose more changes than mere one to one direct attribute translation.

The querying of relationships could be improved in the prototype with an efficient XML query language, for which there is not yet a standard.

6.4 Future work

The offer of an automatic treatment of relationships would be complete with automatic detection of citations and automatic generation of document identifiers (document plus fragment address) from detected citations. Experiments to generate document identifiers [31] in English language documents with journals show that this goal is closer with every passing day. In the prototype of legislative information, the rigid structure of this information makes citation detection possible and not too difficult. The challenge is to generalise it to other structured documents.

The services architecture proposed in chapter 4 can be expanded with more link oriented services, able to extract additional information from links (for example, statistical data about citations or relationships). Links gain relevance in digital libraries -and Internet-, due to the user's interest in finding "related information".

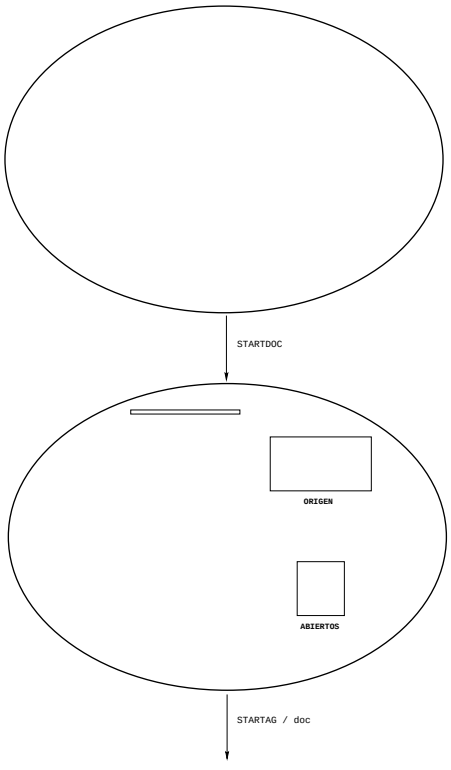
The semantic structure capture algorithm cannot work on any input document, which forces some restrictions to be chosen to impose on them. The algorithm could gain in flexibility with the use of techniques that help to differentiate the start of an element from citations to other elements of the same type inside its content.

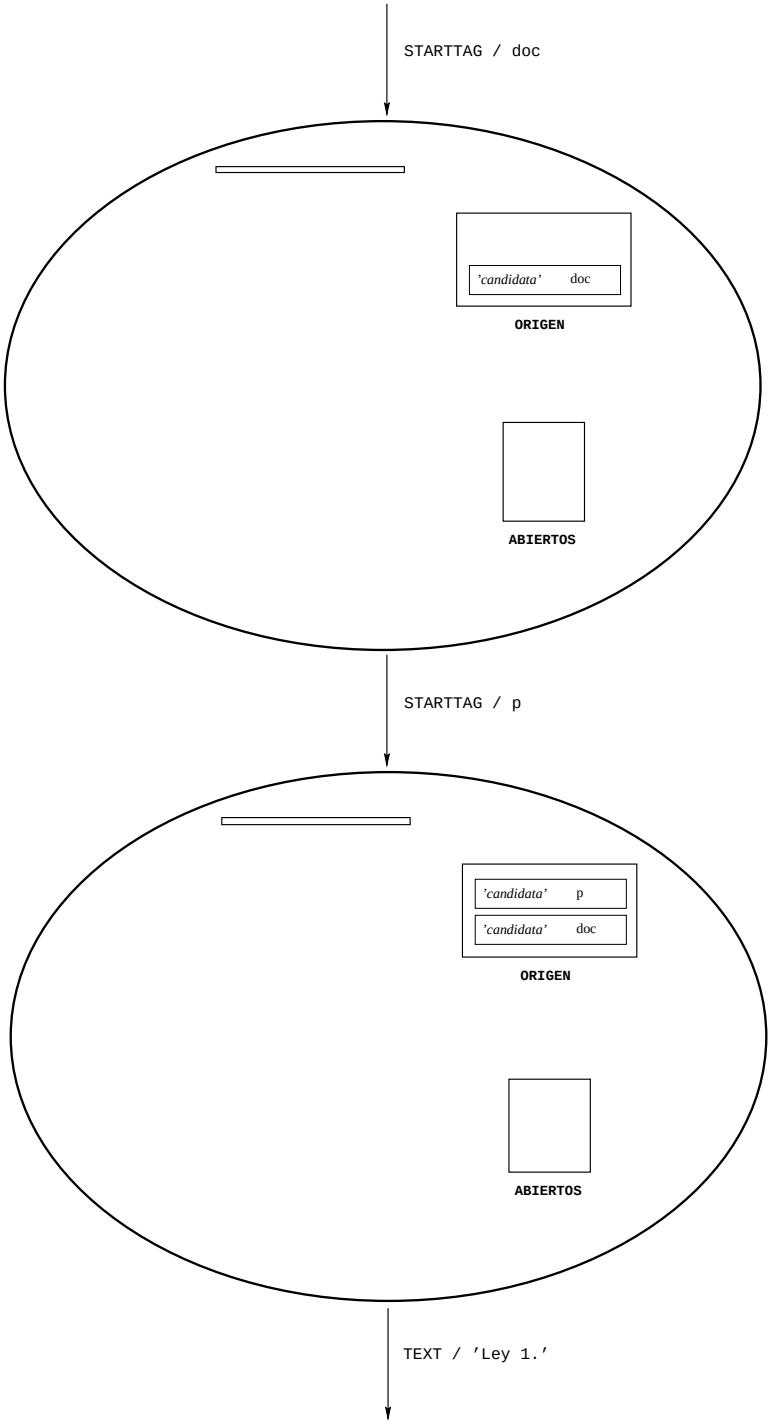
Also, version generation is limited in cases of conflictive modifications, where an automated algorithm has no means to distinguish incoherent modifications from those that should be resolved (see subsection 3.7.3 in chapter 3). The decision taken has been to leave them unresolved and to return them in a list of conflicts, that could be used in an interactive dialogue with the user who has asked for the current document version; the user could help to resolve these cases by indicating which modifications should be applied: an interaction with the user would expand the possibilities of the version generation algorithm, and this would find its place inside the user interface service.

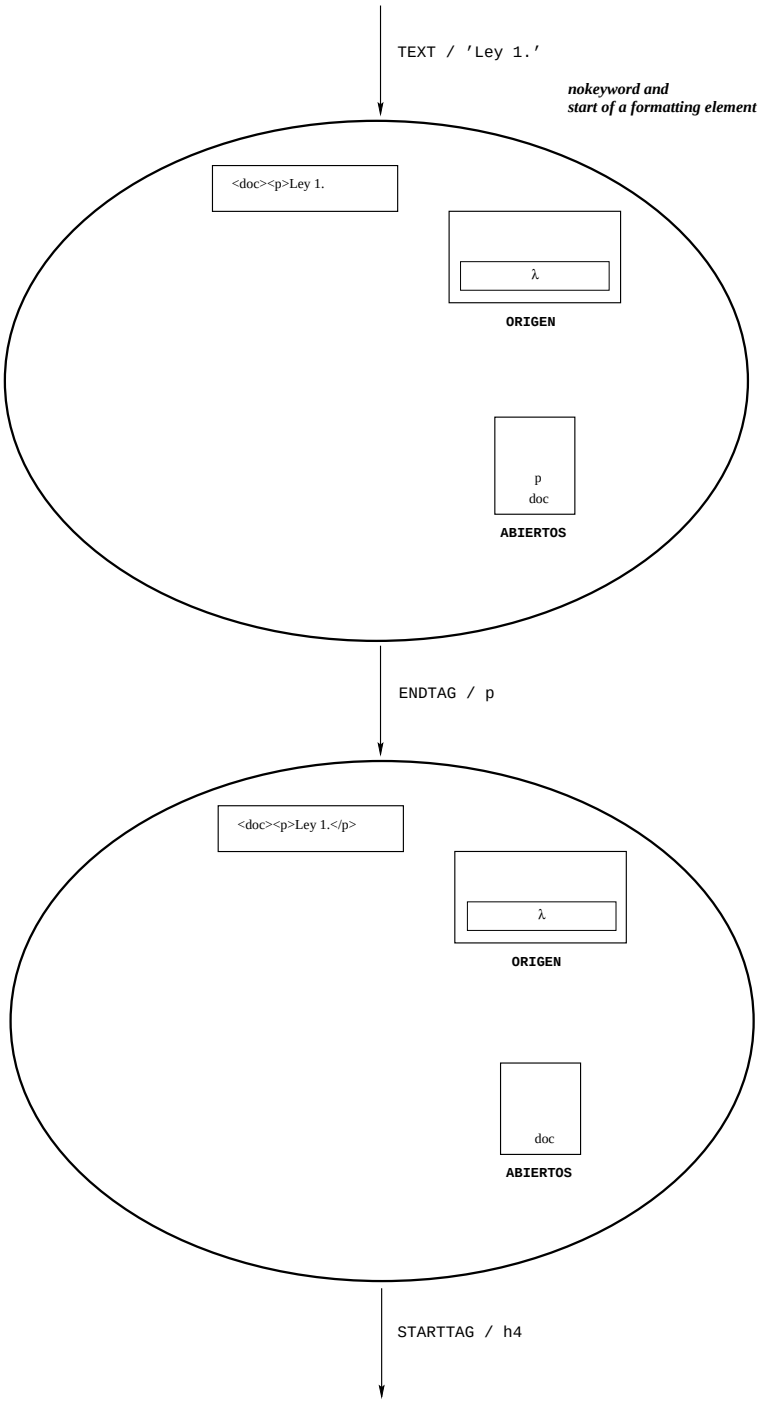
Proposed solutions for documents in digital libraries are expandable to other environments with massive document databases, such as Internet, that share many of their problems and solutions with digital libraries.

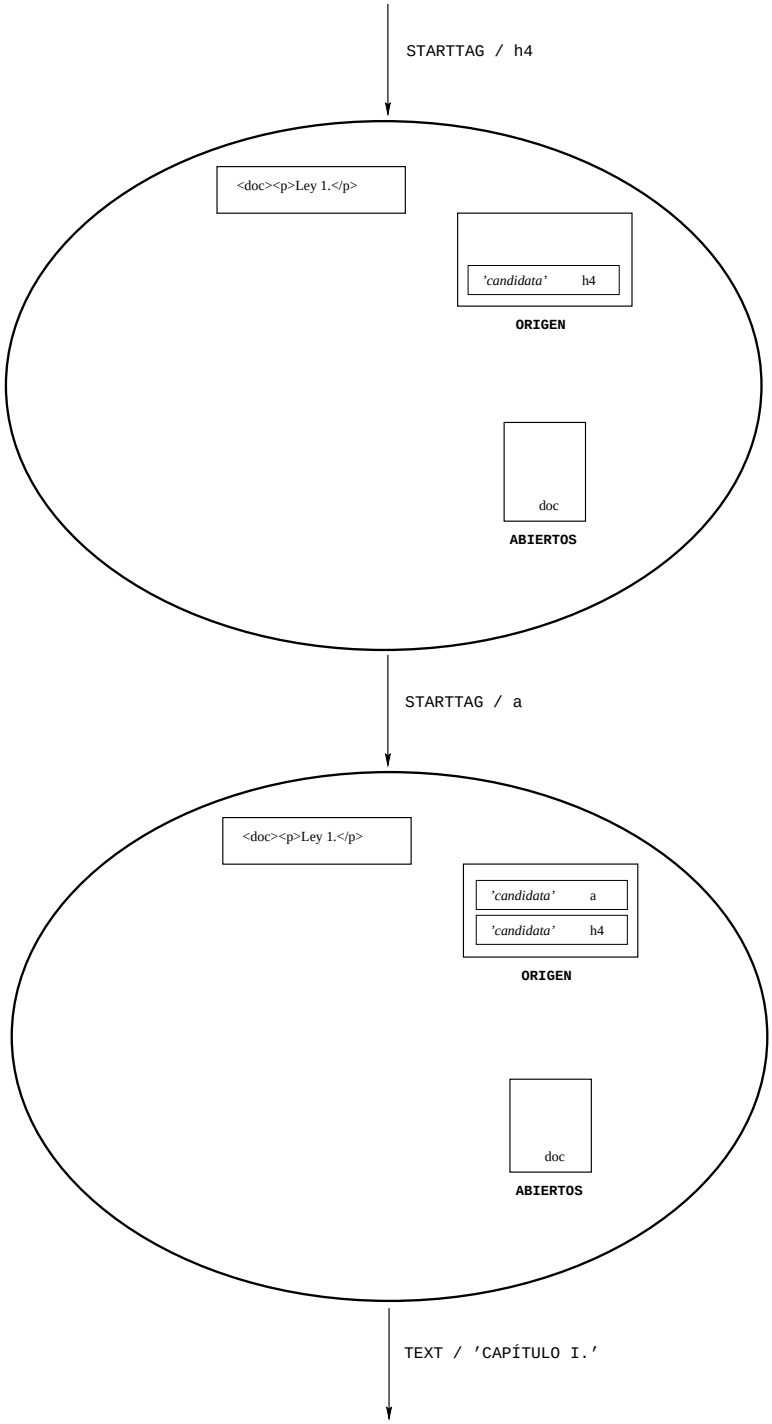
A

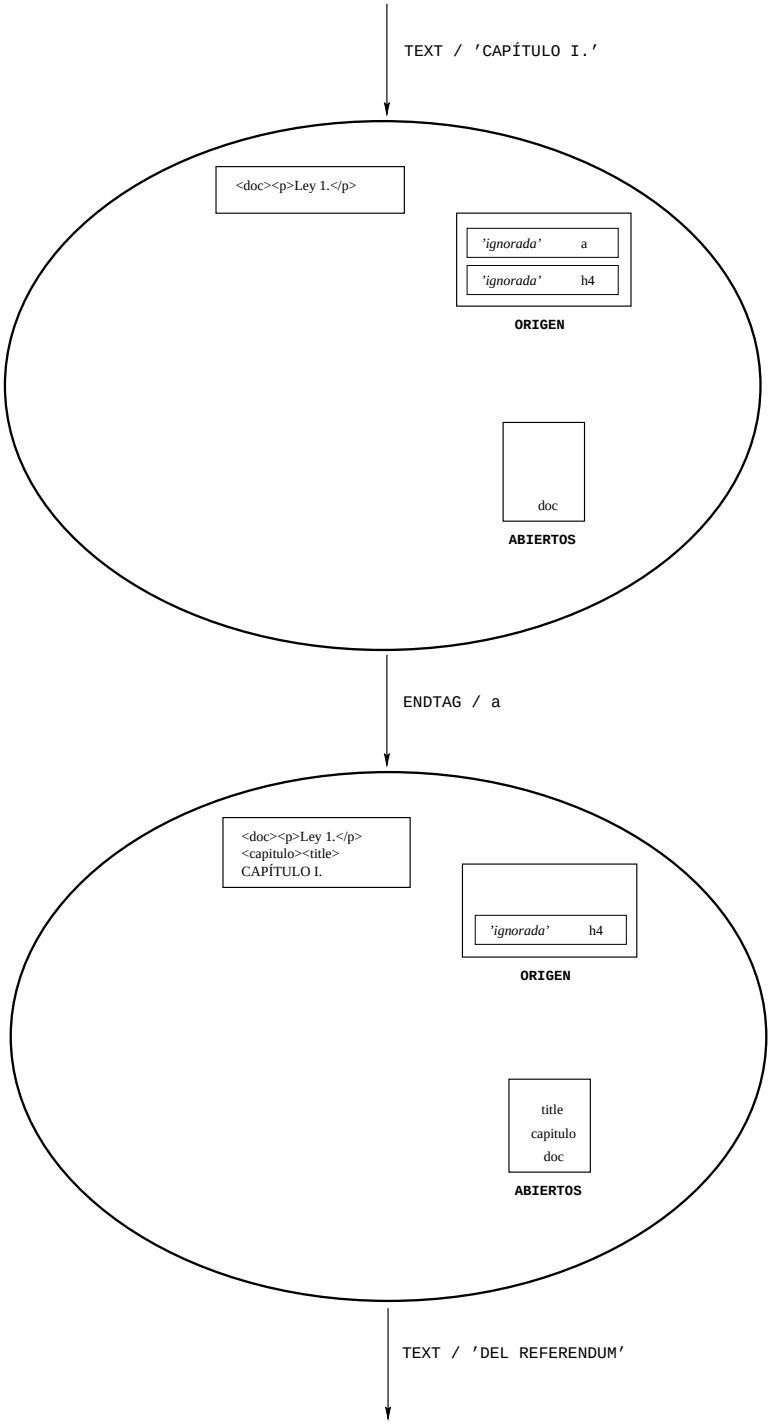
Logical structure capture evolution on an example

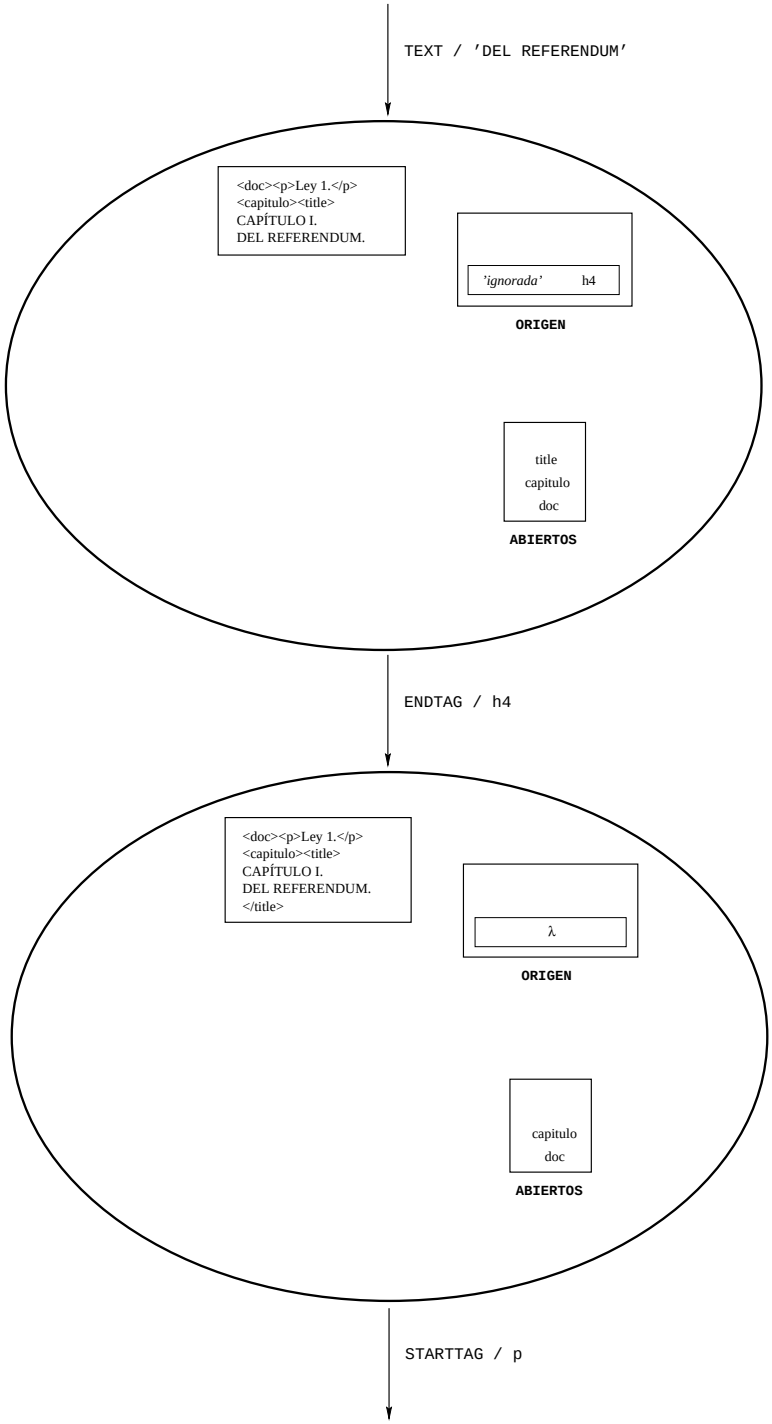


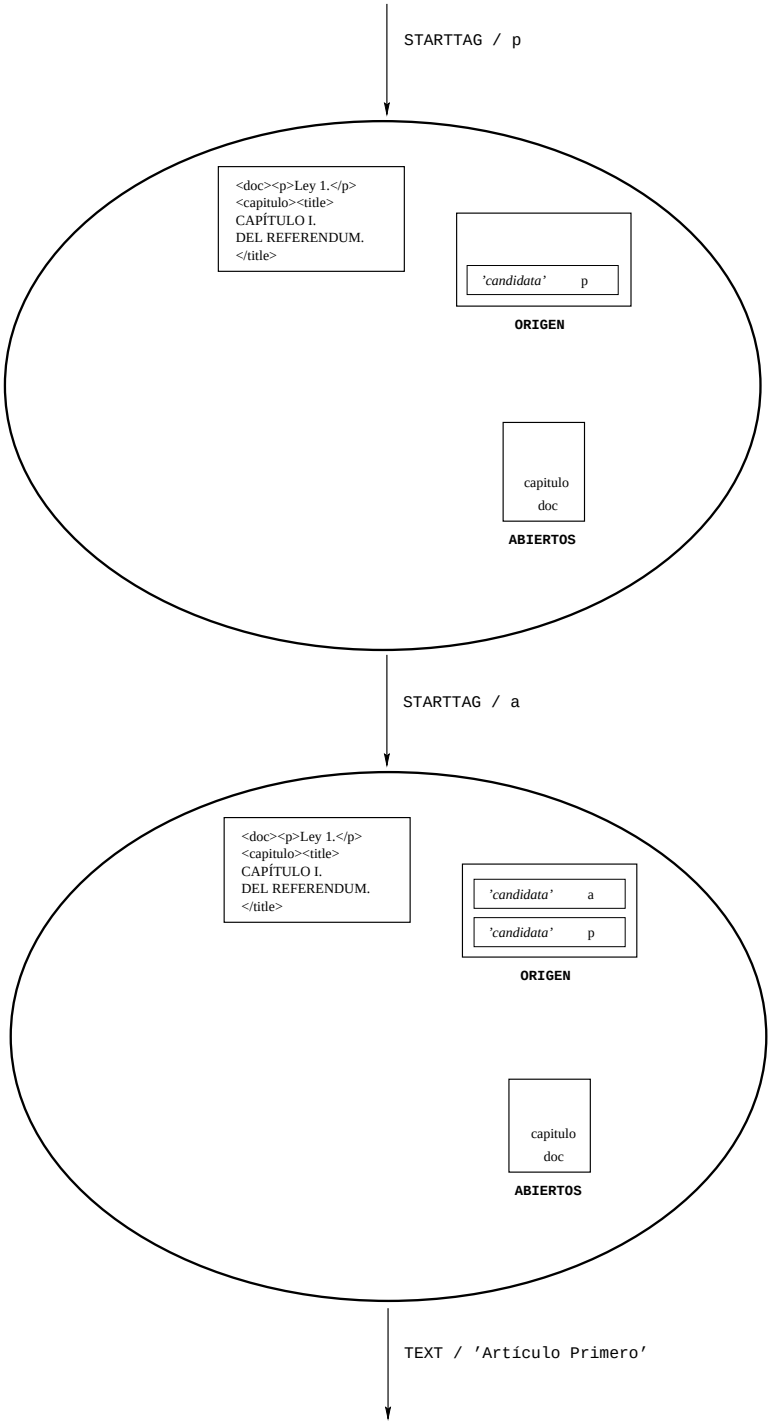

STARTTAG / doc
'candidata' doc
ORIGEN
ABIERTOS
STARTTAG / p
'candidata' p
'candidata' doc
ORIGEN
ABIERTOS
TEXT / 'Ley 1.'

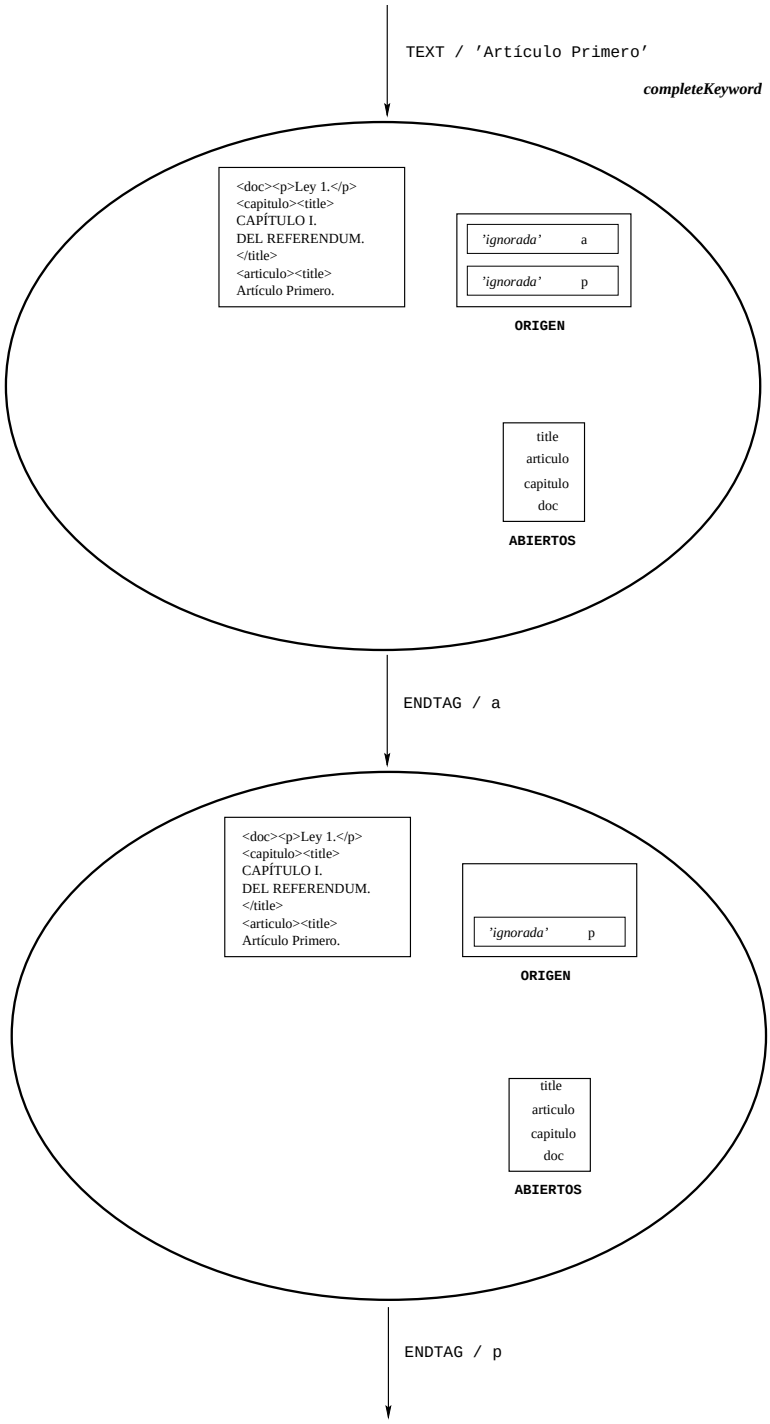

TEXT / 'Ley 1.'
nokeyword and
start of a formatting element
<doc> Ley 1.
λ
ORIGEN
p
doc
ABIERTOS
ENDTAG / p
<doc> Ley 1.
λ
ORIGEN
doc
ABIERTOS
STARTTAG / h4

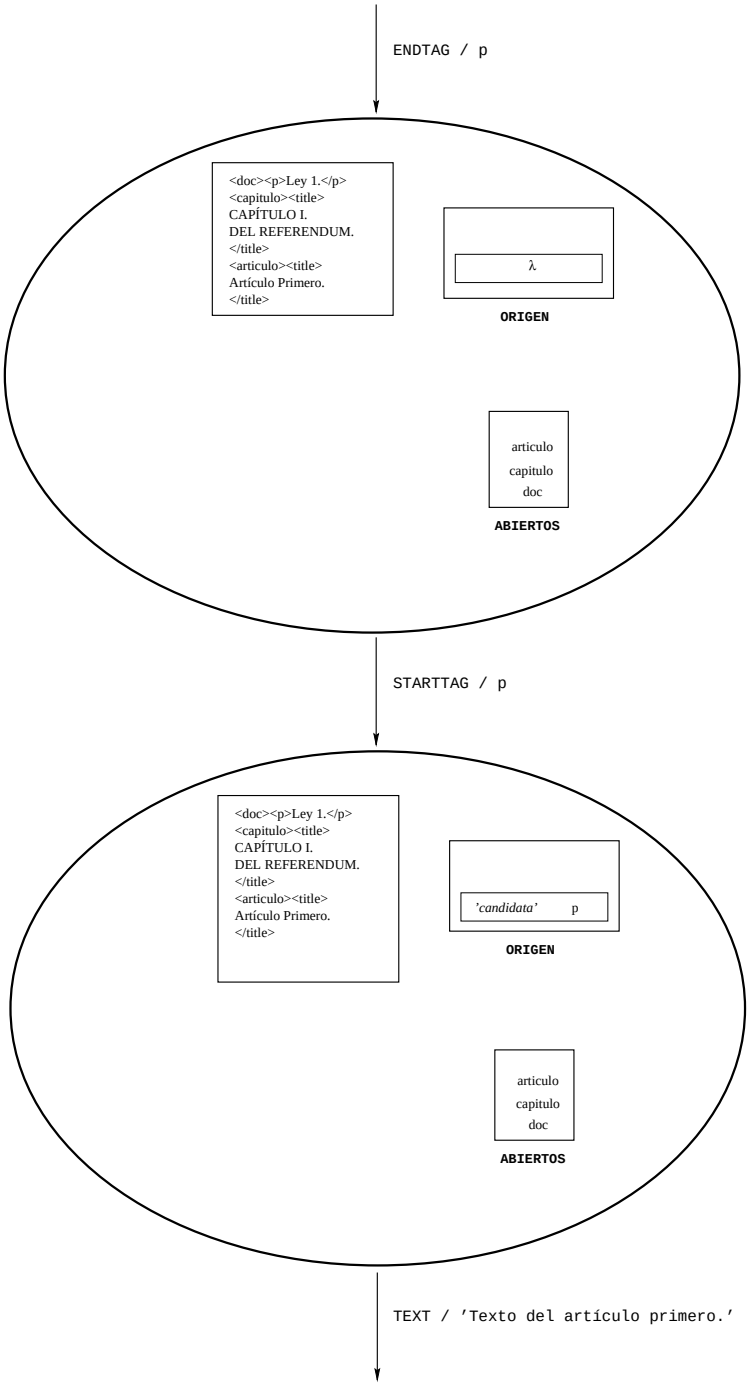

STARTTAG / h4
<doc> Ley 1.
'candidata' h4
ORIGEN
doc
ABIERTOS
STARTTAG / a
<doc> Ley 1.
'candidata' a
'candidata' h4
ORIGEN
doc
ABIERTOS
TEXT / 'CAPÍTULO I.'

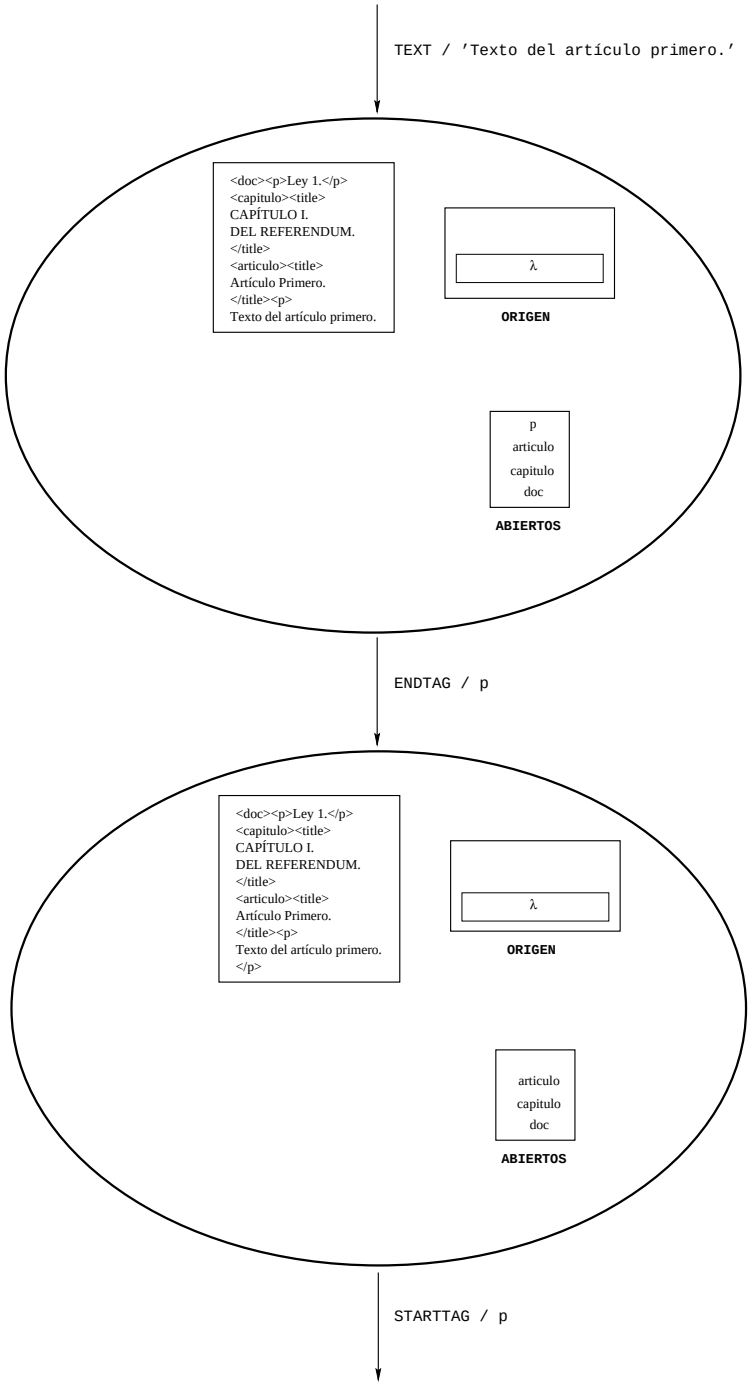

TEXT / 'CAPÍTULO I.'
<doc> Ley 1.
'ignorada' a
'ignorada' h4
ORIGEN
doc
ABIERTOS
ENDTAG / a
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
'ignorada' h4
ORIGEN
title
capitulo
doc
ABIERTOS
TEXT / 'DEL REFERENDUM'


TEXT / 'DEL REFERENDUM'
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
'ignorada'
h4
ORIGEN
title
capitulo
doc
ABIERTOS
ENDTAG / h4
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
λ
ORIGEN
capitulo
doc
ABIERTOS
STARTTAG / p

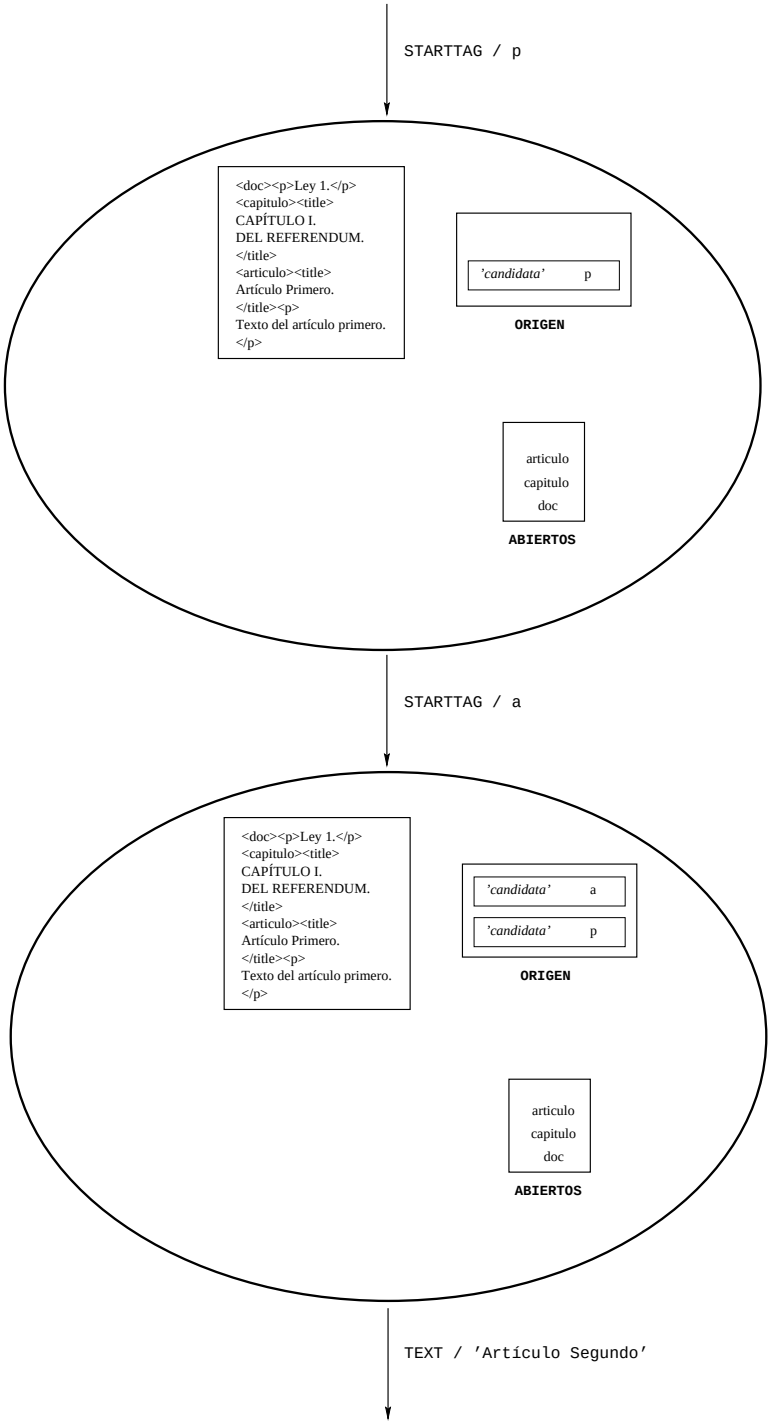

STARTTAG / p
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
'candidata' p
ORIGEN
capitulo
doc
ABIERTOS
STARTTAG / a
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
'candidata' a
'candidata' p
ORIGEN
capitulo
doc
ABIERTOS
TEXT / 'Artículo Primero'


TEXT / 'Artículo Primero'
completeKeyword
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
'ignorada' a
'ignorada' p
ORIGEN
title
articulo
capitulo
doc
ABIERTOS
ENDTAG / a
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
'ignorada' p
ORIGEN
title
articulo
capitulo
doc
ABIERTOS
ENDTAG / p


ENDTAG / p
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
λ
ORIGEN
articulo
capitulo
doc
ABIERTOS
STARTTAG / p
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
'candidata' p
ORIGEN
articulo
capitulo
doc
ABIERTOS
TEXT / 'Texto del artículo primero.'

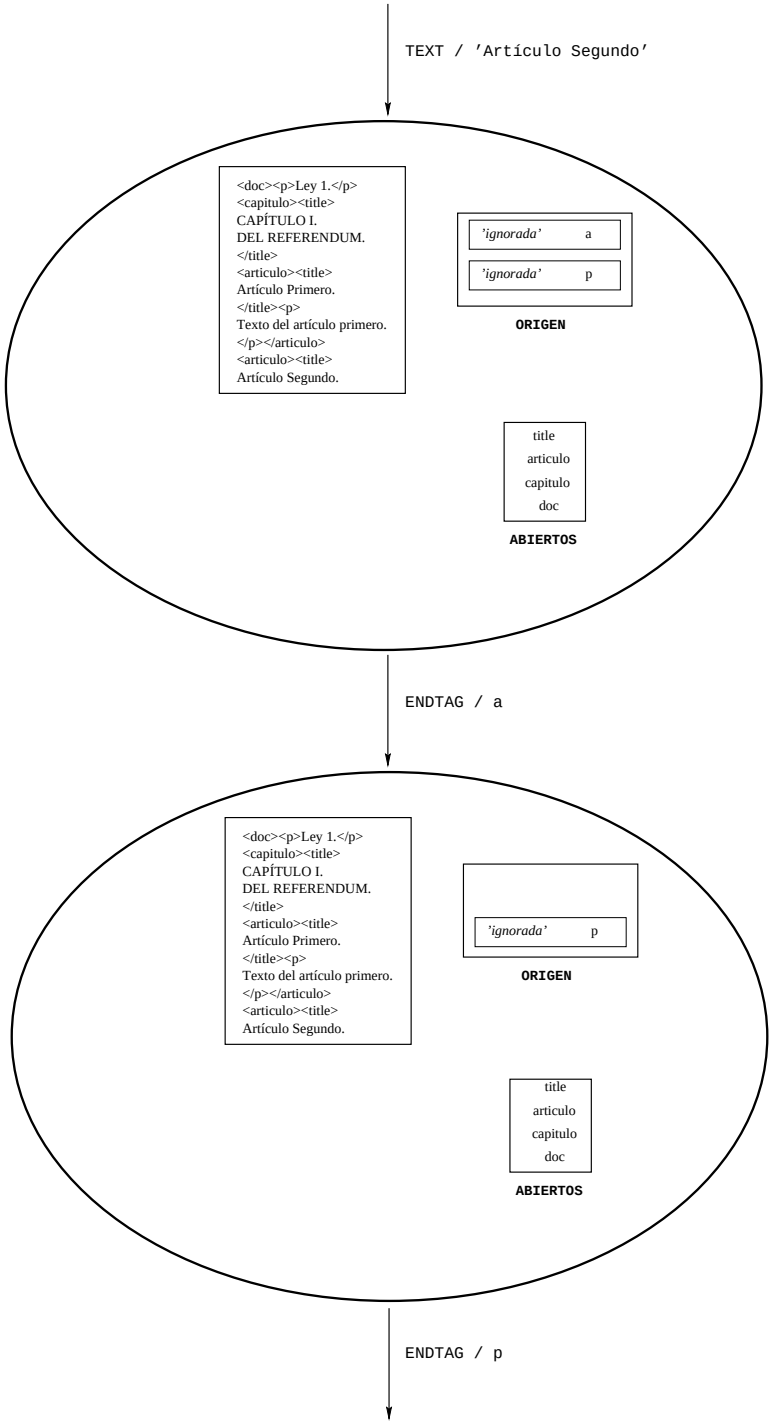

TEXT / 'Texto del artículo primero.'
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
λ
ORIGEN
p
articulo
capitulo
doc
ABIERTOS
ENDTAG / p
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.

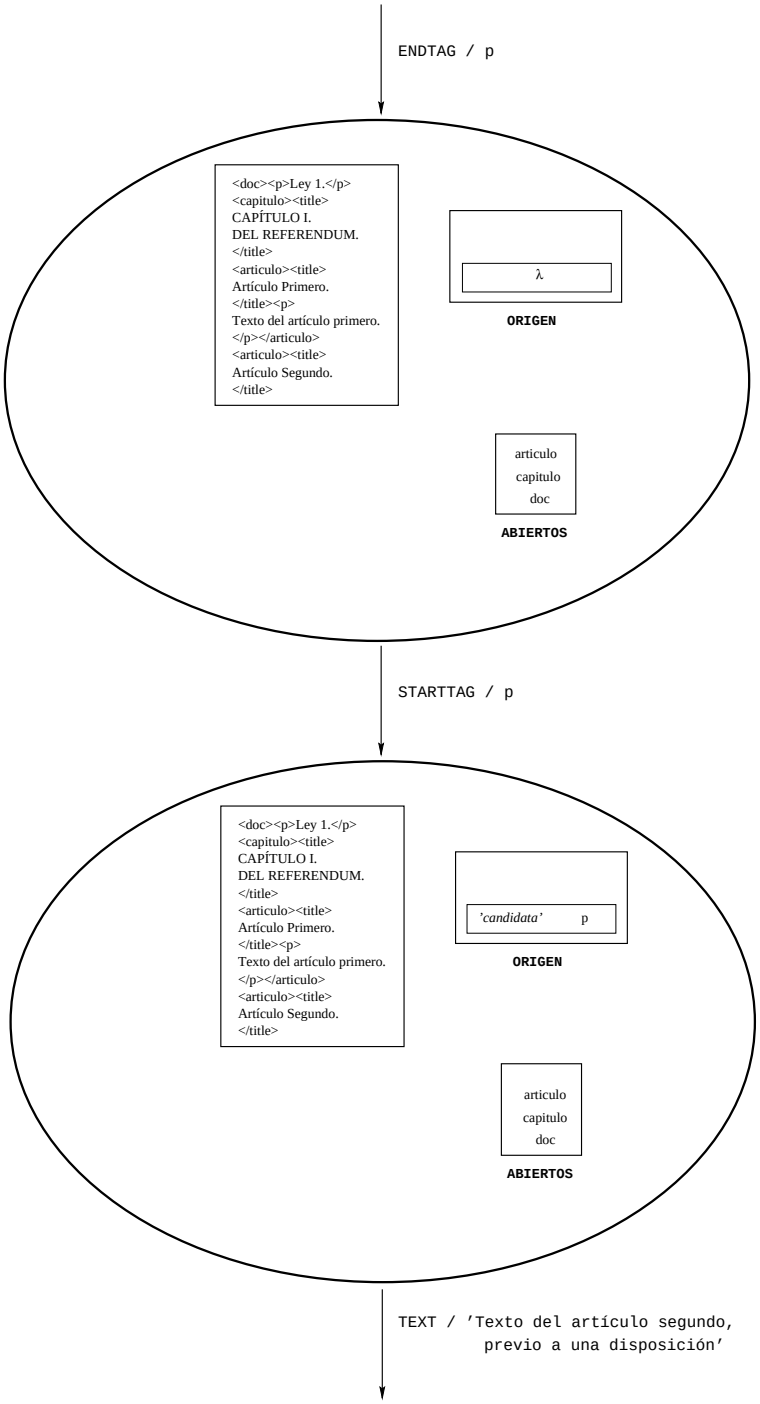
λ
ORIGEN
articulo
capitulo
doc
ABIERTOS
STARTTAG / p

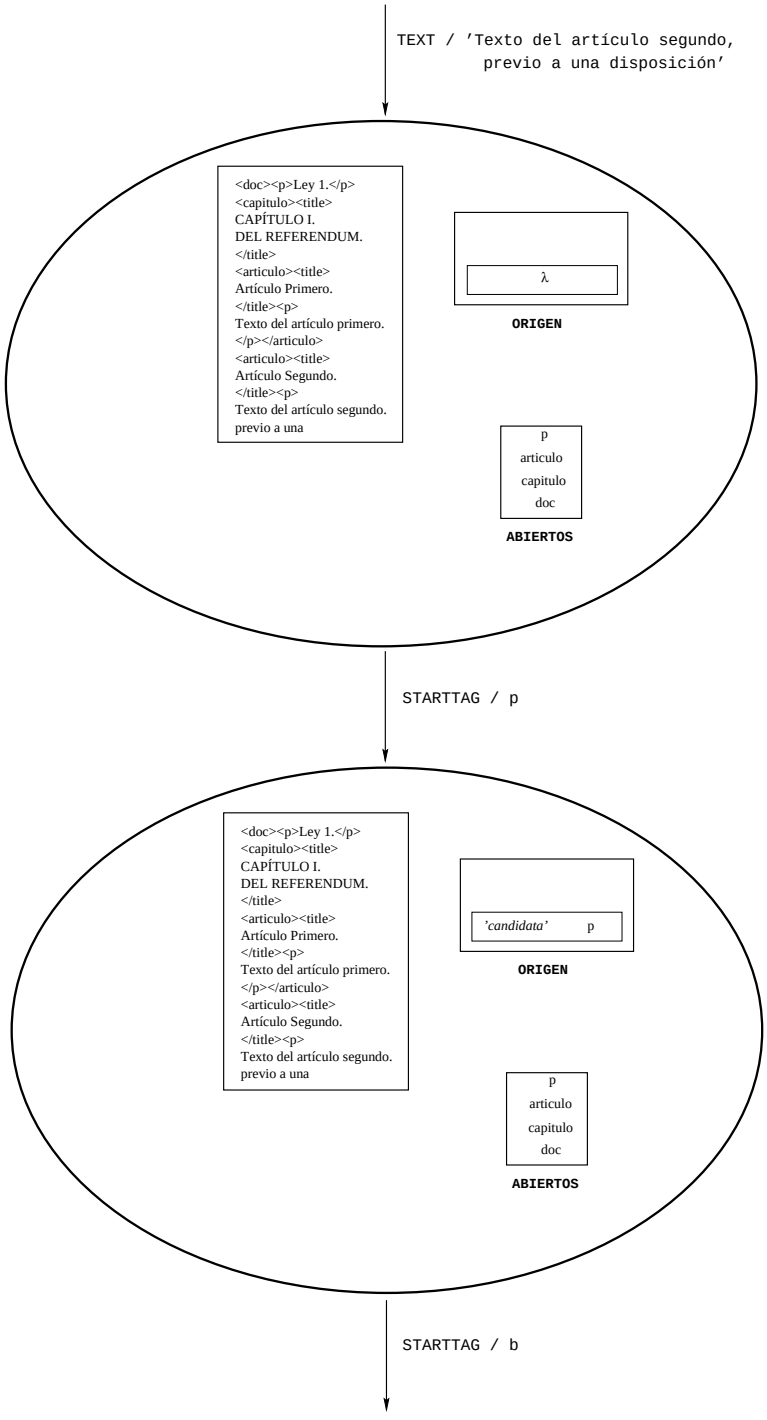

STARTTAG / p
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.

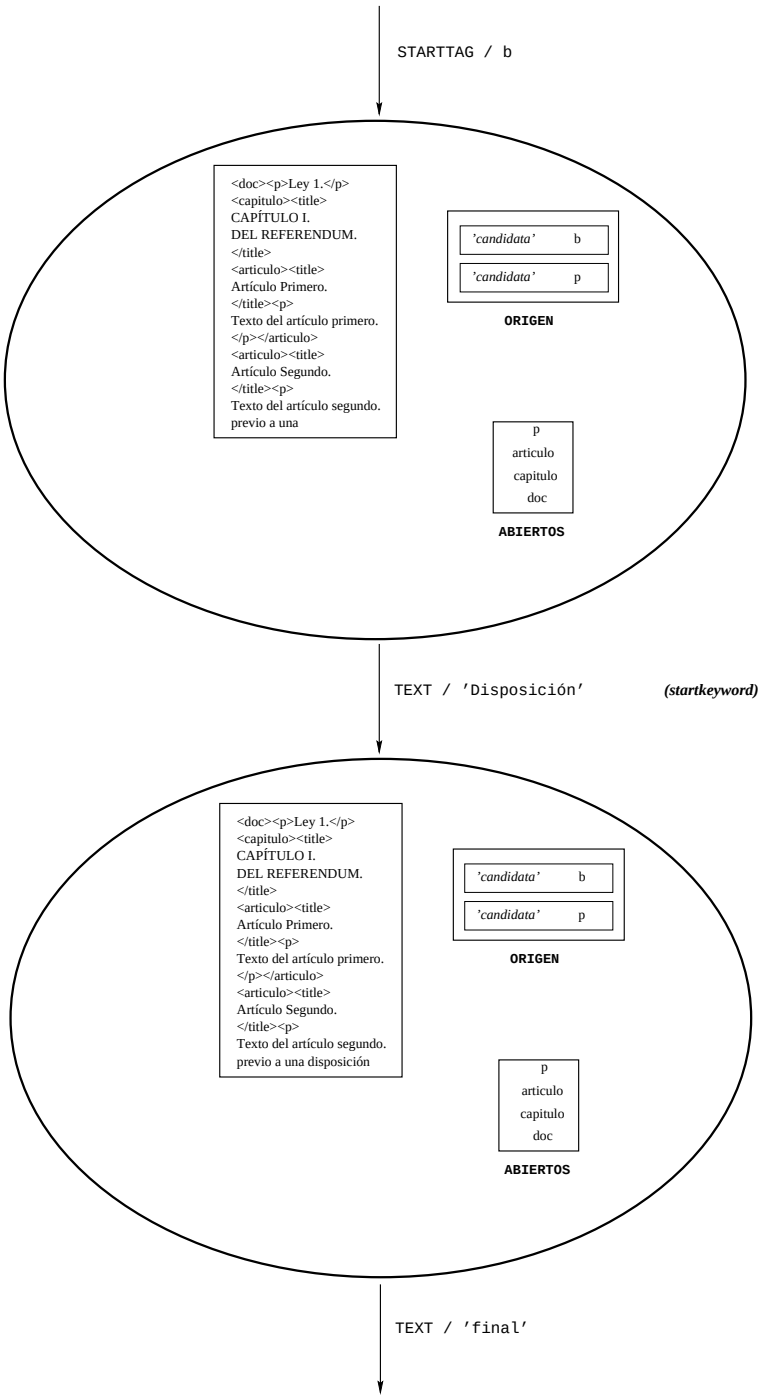
'candidata' p
ORIGEN
articulo
capitulo
doc
ABIERTOS
STARTTAG / a
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.

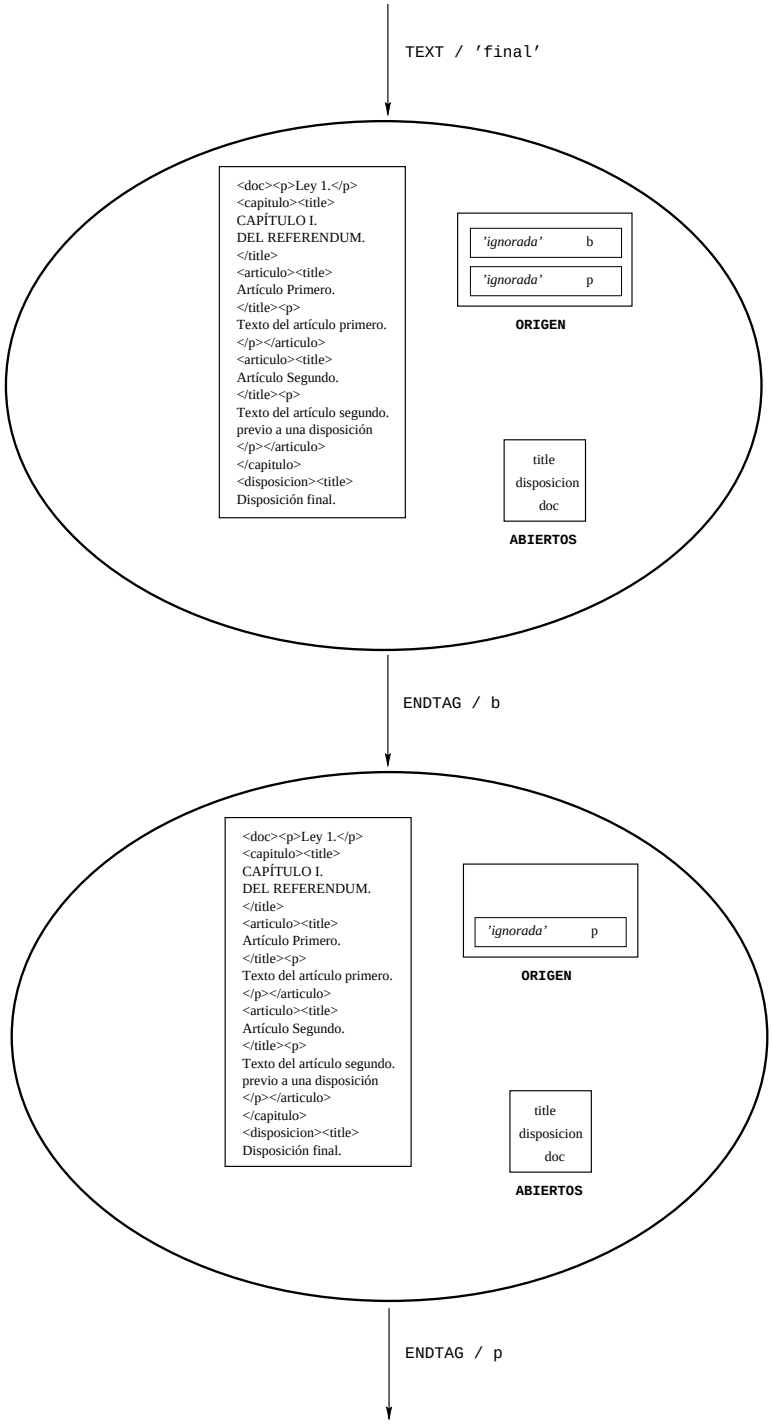
'candidata' a
'candidata' p
ORIGEN
articulo
capitulo
doc
ABIERTOS
TEXT / 'Artículo Segundo'

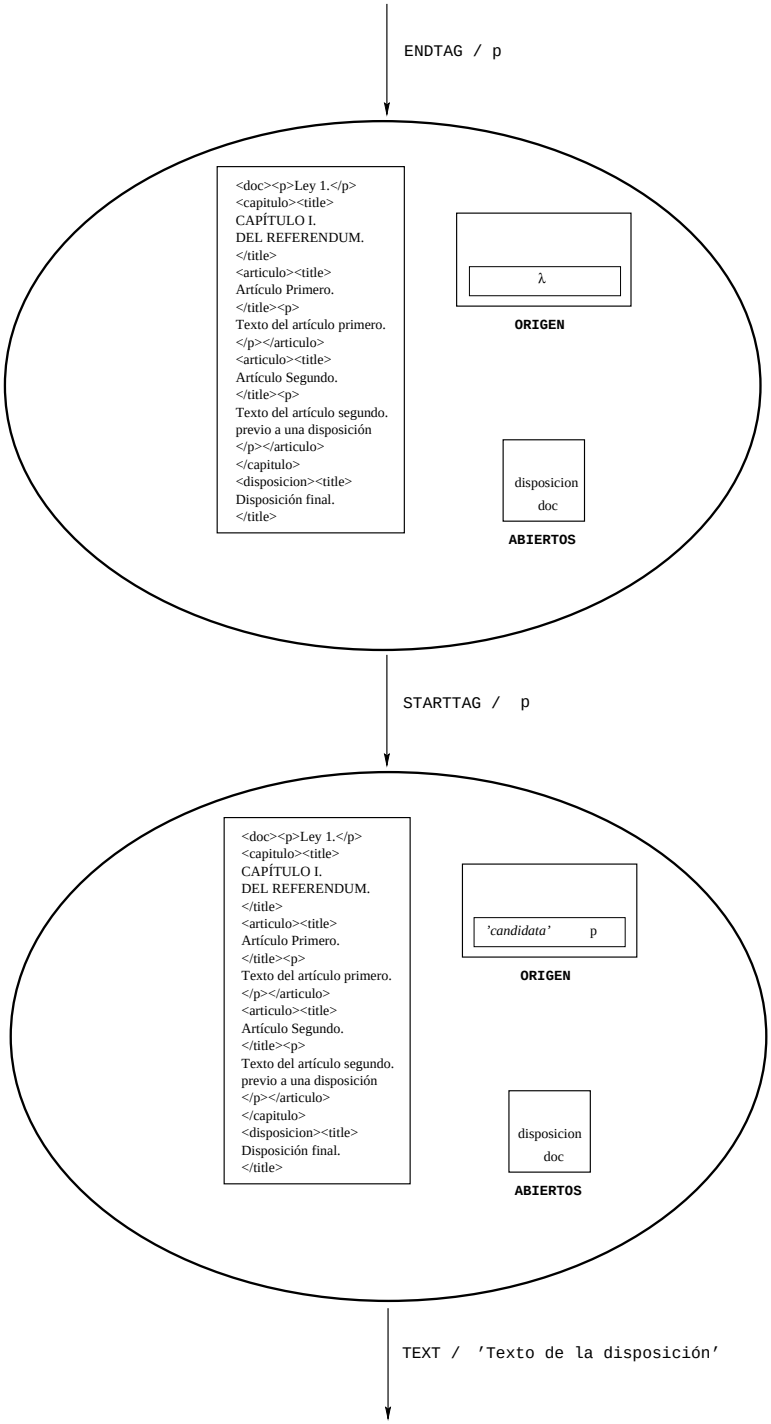

TEXT / 'Artículo Segundo'
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
'ignorada' a
'ignorada' p
ORIGEN
title
articulo
capitulo
doc
ABIERTOS
ENDTAG / a
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
'ignorada' p
ORIGEN
title
articulo
capitulo
doc
ABIERTOS
ENDTAG / p

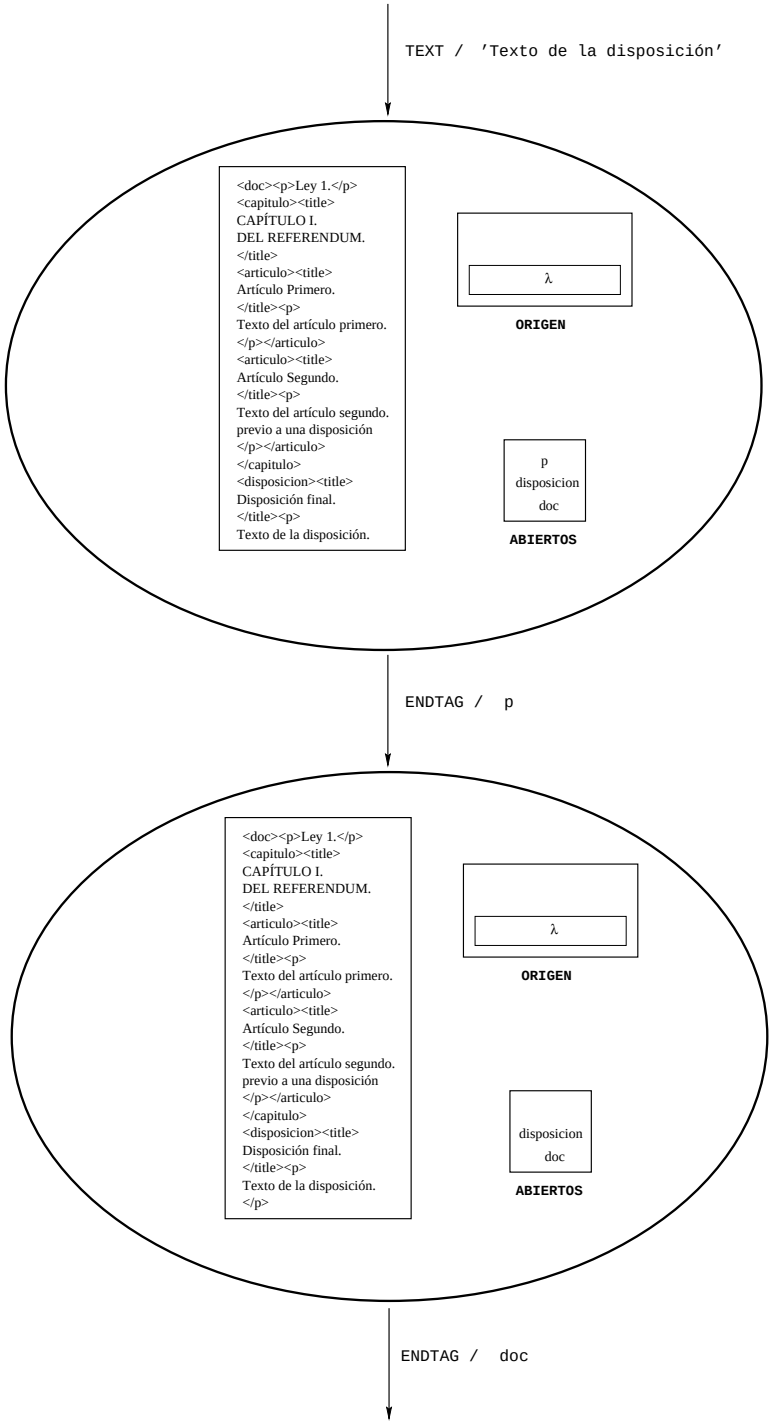

ENDTAG / p
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
λ
ORIGEN
articulo
capitulo
doc
ABIERTOS
STARTTAG / p
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
'candidata' p
ORIGEN
articulo
capitulo
doc
ABIERTOS
TEXT / 'Texto del artículo segundo,
previo a una disposición'


TEXT / 'Texto del artículo segundo,
previo a una disposición'
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una
λ
ORIGEN
p
articulo
capitulo
doc
ABIERTOS
STARTTAG / p
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una
'candidata' p
ORIGEN
p
articulo
capitulo
doc
ABIERTOS
STARTTAG / b

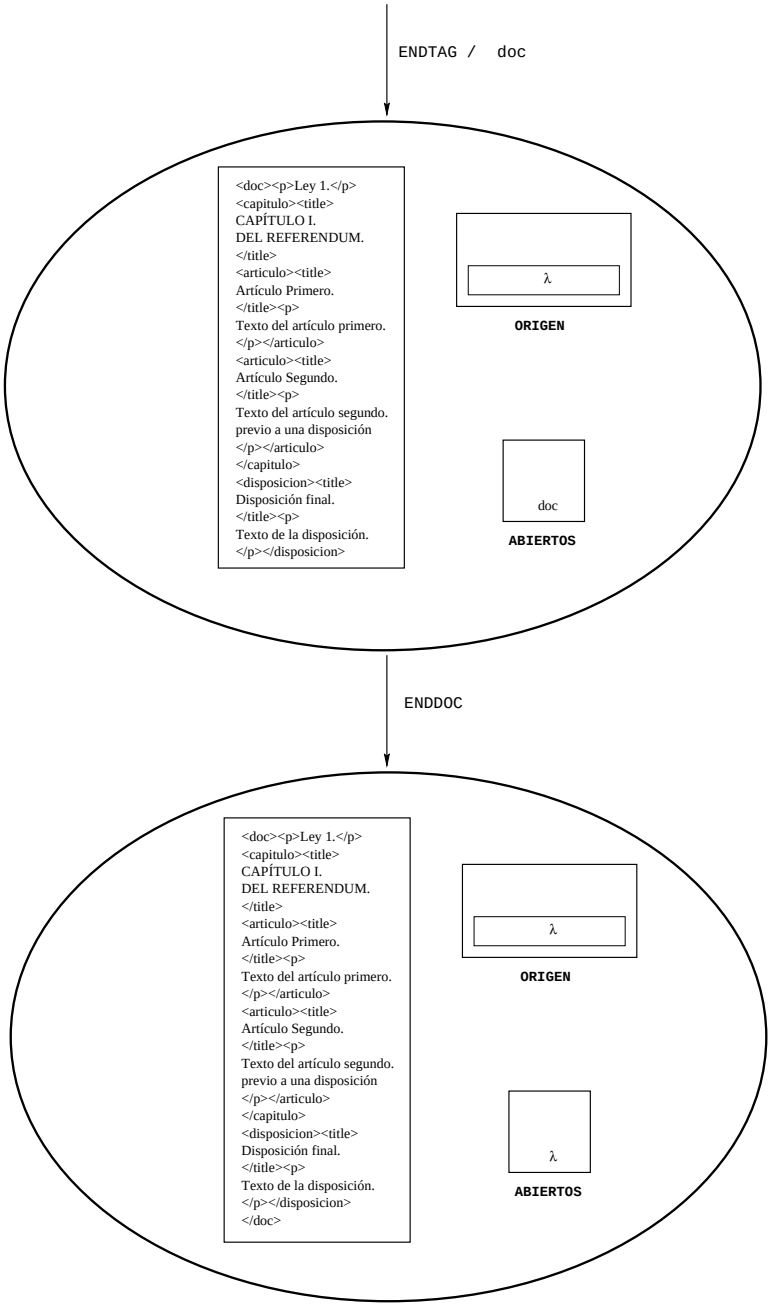

STARTTAG / b
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una
'candidata' b
'candidata' p
ORIGEN
p
articulo
capitulo
doc
ABIERTOS
TEXT / 'Disposición'
(startkeyword)
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una disposición
'candidata' b
'candidata' p
ORIGEN
p
articulo
capitulo
doc
ABIERTOS
TEXT / 'final'


TEXT / 'final'
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una disposición
</articulo>
</capitulo>
<disposicion><title>
Disposición final.
'ignorada' b
'ignorada' p
ORIGEN
title
disposicion
doc
ABIERTOS
ENDTAG / b
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una disposición
</articulo>
</capitulo>
<disposicion><title>
Disposición final.
'ignorada' p
ORIGEN
title
disposicion
doc
ABIERTOS
ENDTAG / p


ENDTAG / p
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una disposición
</articulo>
</capitulo>
<disposicion><title>
Disposición final.
</title>
λ
ORIGEN
disposicion
doc
ABIERTOS
STARTTAG / p
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una disposición
</articulo>
</capitulo>
<disposicion><title>
Disposición final.
</title>
'candidata' p
ORIGEN
disposicion
doc
ABIERTOS
TEXT / 'Texto de la disposición'


TEXT / 'Texto de la disposición'
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una disposición
</articulo>
</capitulo>
<disposicion><title>
Disposición final.
</title>
Texto de la disposición.
λ
ORIGEN
p
disposicion
doc
ABIERTOS
ENDTAG / p
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una disposición
</articulo>
</capitulo>
<disposicion><title>
Disposición final.
</title>
Texto de la disposición.

λ
ORIGEN
disposicion
doc
ABIERTOS
ENDTAG / doc


ENDTAG / doc
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una disposición
</articulo>
</capitulo>
<disposicion><title>
Disposición final.
</title>
Texto de la disposición.
</disposicion>
λ
ORIGEN
doc
ABIERTOS
ENDDOC
<doc> Ley 1.
<capitulo><title>
CAPÍTULO I.
DEL REFERENDUM.
</title>
<articulo><title>
Artículo Primero.
</title>
Texto del artículo primero.
</articulo>
<articulo><title>
Artículo Segundo.
</title>
Texto del artículo segundo.
previo a una disposición
</articulo>
</capitulo>
<disposicion><title>
Disposición final.
</title>
Texto de la disposición.
</disposicion>
</doc>
λ
ORIGEN
λ
ABIERTOS

Bibliography

[1] S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and J. Simeon. Querying documents in object databases. *International Journal on Digital Libraries*, 1(1), 1997.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for semi-structured data. *Internation Journal on Digital Libraries*, 1(1), 1997.

[3] Maristella Agosti, Roberto Colotti, and Girolamo Gradenigo. A two-level hypertext retrieval model for legal data. In *14th ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 316–325, Diparta-mento di Elettronica e Informatica, Universita'di Padova, October 1991. Chicago, IL USA.

[4] James Allan. Automatic Hypertext Link Typing. In *Hypertext'96, the Seventh ACM Conference on Hypertext*, pages 42–52, 1996.

[5] J. André, R. Furuta, and V. Quint. Structured documents: What and why? In J. André, R. Furuta, and V. Quint, editors, *Structured Documents*. Cambridge University Press, 1989.

[6] Jacques André, D. Decouchant, V. Quint, and Helene Richy. Vers un atelier editorial pour les documents structures. In *Congrès AFCET Bureautique, Document, "Groupware", Multimédia*, pages 63–72, Versailles (France), June 1993.

[7] Jacques André, Richard Furuta, and Vincent Quint, editors. *Structured documents*, volume 2 of *The Cambridge series on Electronic Publishing*, Cambridge, New Rochelle, Melbourne, 1989. Cambridge University Press.

[8] Heimburger Anneli. A Structured Link Document as a new means for composing and publishing technical customer documentation in extranets and intranets. In *Electronic Publishing'99*, 1999.

[9] AQUARELLE. The Information Network on the Cultural Heritage. http://aqua.inria.fr/Aquarelle/.

[10] William Y. Arms. Key Concepts in the Architecture of the Digital Library. *D-Lib Magazine*, July 1995.

[11] William Y. Arms, Christophe Blanchi, and Edward A. Overly. An Architecture for Information in Digital Libraries. *D-Lib Magazine*, February 1997.

[12] Timothy Arnold-Moore, Phil Anderson, and Ron Sacks-Davis. Managing a digital library of legislation. In *2nd ACM International Conference on Digital Libraries, ACM DL 1997*, pages 175–183, Philadelphia, PA, USA, July 1997. ACM Press.

[13] Timothy Arnold-Moore, Michael Fuller, Alan Kent, Ron Sacks-Davis, and Neil Sharman. Architecture of a content management server for XML document applications. In *1st International Conference on Web Information Systems Engineering (WISE 2000)*, Hong Kong, June 2000.

[14] Timothy Arnold-Moore, Michael Fuller, and Ron Sacks-Davis. Approaches for structured document management. In *Markup Technologies'99*, Philadephia, PA, USA, December 1999.

[15] Timothy Arnold-Moore, Michael Fuller, and Ron Sacks-Davis. System architectures for structured document data. *Markup Languages: Theory and Practice*, 2(1):15–44, 2000.

[16] M. Baldonado, C. Chang, L. Gravano, and A. Paepcke. The Stanford Digital Library Metadata Architecture. *International Journal of Digital Libraries*, 2(1), February 1997.

[17] Michelle Baldonado, Q. Wang, and Terry Winograd. SenseMaker: An Information-Exploration Interface Supporting the Contextual Evolution of a User's Interests. In *Proceedings of the Conference on Human Factors in Computing Systems, CHI'97*, pages 11–18, Atlanta, Ga., 1997. ACM Press, New York.

[18] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice.* Addison Wesley, 1998.

[19] Abdel Belaid and Amos David. The use of Information Retrieval Tools in Automatic Document Modeling and Recognition. In *Tenth International Workshop on Database and Expert Systems Applications (DAUD'99), Florence, Italia*, pages 522–526, September 1999.

[20] Donna Bergmark. An Architecture for Reference Linking. Presentation of the Cornell Digital Library Research Group, May 2000.

[21] Mark Bernstein. An apprentice that discovers hypertext links. In N. Streitz, A. Rizk, and J. André, editors, *Hypertext: Concepts, Systems and Applications.*, The Cambridge Series on Electronic Publishing, pages 212–223, November 1990.

[22] S. Biagioni, J.L.Borbinha, R. Ferber, P. Hansen, S. Kapidakis, L. Kovacs, F. Roos, and A. M. Vercoustre. The ERCIM Technical Reference Digital Library. In *Second European Conference on Research and Advanced Technology for Digital Libraries*, September 1998.

[23] Stefania Biagioni, Carlo Carlesi, and Donatella Castelli. Supporting retrieval by 'relation amog documents' in the ERCIM Technical Reference Digital Library. In *11th ERCIM Database Research Group Workshop on Metadata for Web Databases*, May 1998.

[24] William P. Birmigham. An agent-based architecture for digital libraries. *D-Lib Magazine*, July 1995.

[25] William P. Birminghan. An Agent-Based Architecture for Digital Libraries. *D-Lib Magazine*, July 1995.

[26] William James Blustein. *Hypertext Versions of Journal Articles: Computer-aided linking and realistic human-based evaluation.* PhD thesis, University of Western Ontario, London, Ontario, Canada, 1999.

[27] S. Bonhomme and C. Roisin. Transformations de documents électroniques. *Document Numérique*, 1(3), 1997.

[28] Stéphane Bonhomme. *Transformation de documents structurés, une combinaison des approches explicite et automatique.* PhD thesis, Université Joseph Fourier (Grenoble), December 1998.

[29] Tim Bray. Comparison of SGML and XML. World Wide Web Consortium Note 15-December-1997.

[30] Heather Brown. Standards for structured documents. *The Computer Journal*, 32(6):505–514, December 1989.

[31] Priscilla Caplan and William Y. Arms. Reference Linking for Journal Articles. *D-Lib Magazine*, 5(7/8), 1999.

[32] L. A. Carr, W. Hall, and S. Hitchcock. Link Services or Link Agents? In *9th ACM Conference on Hypertext and Hypermedia*, 1998.

[33] Leslie Carr, Wendy Hall, and David De Roure. The evolution of hypertext link services. *ACM Computing Surveys*, 31(4), December 1999.

[34] Wojciech Cellary and Geneviève Jomier. *Building an object-oriented databse system. The story of O_2.*, chapter Consistency of Versions in Object-Oriented Databases. Number 19 in The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 1992.

[35] Sudarshan S. Chawathe, Serge Abiteboul, and Jennifer Widom. Managing historical semistructured data. *Theory and Practice of Object Systems*, 5(3):143–162, August 1999.

[36] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(2), 1996.

[37] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Control*, 2(3):113–124, 1956.

[38] Martin Choquette, Daniel Poulin, and Paul Bratley. Compiling Legal Hypertexts. In Norman Revell and A. Min Tjoa, editors, *Database and Expert Systems Applications, 6th International Conference, DEXA'95*, volume 978 of *Lecture Notes in Computer Science*, pages 449–458. Springer, September 1995.

[39] Jeff Conklin. Hypertext: An introduction and survey. *IEEE Computer*, 20(9):17–41, 1987.

[40] V. Cristophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *ACM SIGMOD Conference on Management of Data*, pages 312–324, 1994.

[41] *CVS. Concurrent Versions System.* http://www.cvshome.org/docs/manual/.

[42] J. R. Davis. Creating a Networked Computer Science Technical Report Library. *D-Lib Magazine*, September 1995.

[43] J. R. Davis and C. Lagoze. A protocol and server for a distributed digital technical report library. Technical Report TR94-1418, Computer Science Department, Cornell University, 1994.

[44] J. R. Davis and Carl Lagoze. The Networked Computer Science Technical Report Library. Technical report, Cornell University, 1996.

[45] James R. Davis and Carl Lagoze. NCSTRL: Design and Deployment of a Globally Distributed Digital Library, 1999.

[46] Herbert Van de Sompel and Patrick Hochstenbach. Reference linking in a hybrid library environment. *D-Lib Magazine*, 5, April 1999.

[47] Steven J. DeRose. Expanding the Notion of Links. In Norman Meyorwitz, editor, *Proceedings of Hypertext'89*, pages 249–255, Pittsburgh, PA Baltimore, 1989. Association for Computing Machinery Press.

[48] Steven J. DeRose and David G. Durand. The TEI Hypertext Guidelines. *Computing and the Humanities*, 29(3), 1995.

[49] Steven J. DeRose, C.M. Seperberg-McQueen, and Bill Smith. Queries on Links and Hierarchies. In *Proceedings of QL'98 - The Query Languages Workshop* , Boston, december 1998.

[50] Steven J. DeRose, C.M. Seperberg-McQueen, and Bill Smith. XQuery: A unified syntax for linking and querying general XML documents. In *Proceedings of QL'98 - The Query Languages Workshop* , Boston, december 1998.

[51] Jack Doggen. FORMEX V3L Tagging the Laws: SGML Used for Complex Multilingual Documents. In *SGML'96: Celebrating a Decade of SGML*, Boston, 1996.

[52] Bob DuCharme. Links That Are More Valuable Than the Information They Link? *XML.com*, July 1998.

[53] Bob DuCharme. What XLink Can Do for Your Applications. *XML Magazine*, Spring 2000.

[54] Jacques Ducloy. DILIB, une plate-forme XML pour la génération de serveurs WWW et la veille scientifique et technique. *MicroBulletin du CNRS*, pages 3–9. http://www.loria.fr/projets/DILIB/dilib-0.2/DOC/index.html.

[55] Eulegis. http://www.eulegis.net.

[56] Nicholas Finke. TEI Extensions for Legal Text. In *Text Encoding Initiative Tenth Anniversary User Conference*, Providence, Rhode Island, USA, November 1997.

[57] Edward A. Fox and Robert K. France. Architecture of an Expert System for Composite Document Analysis, Representation, and Retrieval. *International Journal of Approximate Reasoning*, 1(2):151–175, April 1987.

[58] R. Furuta and P. D. Stotts. Object structures in paper documents and hypertexts. In *Workshop on Object-Oriented Document Manipulation (WOODMAN'89)*, Rennes, France, May 1989.

[59] Richard Furuta. Concepts and models for structured documents. In J. André, R. Furuta, and V. Quint, editors, *Structured Documents*, pages 7–38. Cambridge University Press, 1989.

[60] Richard Furuta and P. David Stotts. Specifying structured document transformations. In J.C. van Vliet, editor, *Document Manipulation and Typography*, pages 109–120, Nice (France), April 1988.

[61] George H.Brett II. An Integrated System for Distributed Information Services. *D-Lib Magazine*, December 1996.

[62] Rosa Maria Di Giorgi and Roberta Nannucci. Hypertext systems for the law. In *Informatique et droit/ Computers and law*, Montreal, 1992.

[63] L. Gravano, C.C.K.Chang, H. Garcia-Molina, and A. Paepcke. STARTS. Stanford Protocol Proposal for Internet Retrieval and Search. In *ACM SIGMOD Conference on Management of Data*, 1997.

[64] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. The effectiveness of *GlOSS* for the text-database discovery problem. In *Proceedings of the International Conference on Management of Data*, May 1994.

[65] Georg Haider, Cecilia Magnusson Sjöberg, Gerald Quirchmay, and Verena Sebald. The Comparative Part of the Corpus Legis Project - Using SGML for Intelligent Information Retrieval of Legal Documents. In A. Niku-Lari., editor, *EXPERSYS-96, Artificial Intelligence Applications.*, Technology Transfer Series, pages 181–186, 1996.

[66] Frans C. Heeman. Granularity in structured documents. *Electronic Publishing*, 5(3):143–155, September 1992.

[67] S. Hitchcock, L. Carr, S. Harris, J.M.N. Hey, and W. Hall. Citation Linking: Improving Access to Online Journals. In *Second ACM International Conference on Digital Libraries*, pages 115–122, 1997.

[68] Steve Hitchcock. Linking Electronic Journals: Lessons from the Open Journal Project. *D-Lib Magazine*, December 1998.

[69] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation.* Addison Wesley, 1979.

[70] T. Hu. *New methods for robust and efficient recognition of the logical structures in documents.* PhD thesis, Université de Fribourg, 1994.

[71] Karen Hunter. Adding Value by Adding Links. *Journal of Electronic Publishing*, 3, March 1998. http://www.press.umich.edu/jep/03-03/hunter.html.

[72] IFLA Study Group on the Functional Requirements for Bibliographic Recrods. Functional requirements for bibliographic records. Deutsche Bibliotek, Frankfurt-am-Main, 1997. http://www.ifla.org/VII/s13/frbr/frbr.pdf.

[73] Indecs Home Page. http://www.indecs.org/.

[74] La infopista jurídica. http://www.juridica.com/.

[75] International Organization for Standardization, Geneve. *Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML) (ISO 8879:1986)*, 1986.

[76] Eila Kuikka and Martti Penttonen. Tansformation of structured documents with the use of grammar. In *Electronic Publishing*, volume 6, pages 373–383, dec 1993.

[77] Carl Lagoze and David Fielding. Defining Collections in Distributed Digital Libraries. *D-Lib Magazine*, November 1998.

[78] La Ley - Web Site. http://www.laley.net.

[79] LEGGIO - Noticias Jurídicas. http://www.leggio.com/.

[80] Légifrance. L'essentiel du droit français. http://www.legifrance.gouv.fr.

[81] Legislación Básica del Estado - Centro de Información Administrativa - MAP - España. http://www.igsap.map.es/cia/dispo/Ibe.htm.

[82] Barry M. Leiner. The NCSTRL Approach to Open architecture for the Confederated Digital Library. *D-Lib Magazine*, December 1998.

[83] Seung-Jin Lim and Yiu-Kai Ng. WebView: A Tool for Retrieving Internal Structures and Extracting Information from HTML Documents. In *Sixth International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 71–78, Hsinchu, Taiwan, April 1999. IEEE Computer Society.

[84] G. Lindén. *Structured Document Transformations*. PhD thesis, Dept. of Computer Science, University of Helsinki, 1997.

[85] Philippe Martin and L. Alpay. Conceptual structures and structured documents. In Peter W. Edlund, Gerard Ellis, and Graham Mann, editors, *Conceptual Structures: Knowledge Representation as Interlingua, 4th International Conference on Conceptual Structures, ICCS '96,*, volume 1115 of *Lecture Notes in Computer Science*, pages 145–159, Sydney, Australia, August 1996. Springer.

[86] Guido De Mets. Consleg Interleaf: SGML Applied in Legislation. In *SGML'96: Celebrating a Decade of SGML*, Boston, 1996.

[87] National Institute of Information Standards, http://sunsite.berkeley.edu/SICI. *SICI: Serial Item and Contribution Identifier Standard*, ANSI/NISO Z39.56 Version 2 edition.

[88] Peter J. Nürnberg, Richard Furuta, John J. Leggett, Catherine C. Marshall, and Frank M. Shipman III. Digital libraries: Issues and architectures. In *Digital Libraries 95*. Center for the Study of Digital Libraries, Texas A&M University, 1995.

[89] Association of Research Libraries. Definition and Purposes of a Digital Library, October 1995.

[90] Andreas Paepcke. Digital libraries: Searching is not enough. *D-Lib Magazine*, May 1996.

[91] Andreas Paepcke, Che Chuan K. Chang, Héctor García Molina, and Terry Winograd. Interoperability for Digital Libraries Worldwide. *Communications of the ACM*, April 1998.

[92] H. Van Dyke Parunak. Don't Link Me In: Set Based Hypermedia for Taxonomic Reasoning. In *Proceedings of the Third ACM Conference on Hypertext*, pages 233–242, San Antonio, Texas, USA, December 1991.

[93] H. Van Dyke Parunak. *Hypertext/Hyermedia Handbook*, chapter Ordering the information graph, pages 299–325. Intertext Publications, 1991.

[94] Norman Paskin. DOI: Current Status and Outlook May 1999. *D-Lib Magazine*, May 1999. http://www.dlib.org/dlib/may99/05paskin.html.

[95] James Powell and Edward A. Fox. Multilingual Federated Searching Across Heterogeneous Collections. *D-Lib Magazine*, September 1998.

[96] S. Ranwez and M. Crampes. Conceptual documents and hypertext documents are two different forms of virtual documents. In *Workshop on Virtual Documents, Hypertext Functionnality and the Web, Eight International World Wide Web Conference, Toronto, Canada*, 1999.

[97] Antoine Rizk and Dale Sutcliffe. Distributed link service in the AQUARELLE project. In Mark Bernstein, Les Carr, and Kasper Østerbye, editors, *8th ACM Conference on Hypertext and Hypermedia*. ACM, 1997.

[98] Martin Roscheisen, Michelle Baldonado, Kevin Chang, Luis Gravano, Steven Kechpel, and Andreas Paepcke. The Stanford InfoBus and Its Service Layers, August 1997. http://www-diglib.stanford.edu.

[99] R. Sacks-Davis, T. Arnold-Moore, and J. Zobel. Database systems for structured documents. In *International Symposium on Advanced Database Technologies and Their Implementation.*, pages 272–283, Nara, Japan, October 1994. Invited paper.

[100] Bruce Schatz, William Mischo, Timothy Cole, Ann Bishop, Susan Harum, Eric Johnson, Laura Neumann, Hsinchun Chen, and Dorbin Ng. Federated Search of Scientific Literature. *Computer*, 32(2), February 1999.

[101] Cecilia Magnusson Sjöberg. DTD development for the legal domain. In *Swedis SGML 97*, 1997. http://info.admin.kth.se/SGML/.

[102] D. Smith and M. Lopez. Information extraction for semistructured documents. In *Workshop on Management of Semi-structured Data*, Tucson, Arizona, 1997.

[103] C. M. Sperber-McQueen and Lou Burnard, editors. *Guidelines for Electronic Text Encoding and Interchange*. ALLC/ACH/ACL Text Encoding Initiative, 1994.

[104] Stanford Digital Libraries Project. http://www-diglib.stanford.edu.

[105] Steve B. Cousins. A task-oriented interface to a digital library. In *CHI 97 Conference Companion*, pages 103–104, 1996.

[106] K. Summers. Toward a taxonomy of logical document structures. In *Electronic Publishing and the Information Superhighway: Proceedings of the Dartmouth Institute for Advanced Graduate Studies (DAGS '95)*, pages 124–133. Boston, 1995.

[107] Kristen Maria Summers. *Automatic Discovery of Logical Document Structure*. PhD thesis, Cornell University, aug 1998.

[108] Janet Verbyla. Unlinking the Link. *ACM Computing Surveys*, 31(4):34–, December 1999.

[109] Anne-Marie Vercoustre and François Paradis. Reuse of Linked Documents through Virtual Document Prescriptions. *Lecture Notes in Computer Science. Lecture Notes in Artificial Intelligence*, May 1998.

[110] W3C, the World Wide Web Consortium. *Cascading Style Sheets, level 2 CSS2 Specification. W3C Recommendation 12-May-1998.*

[111] W3C, the World Wide Web Consortium. *Extensible Stylesheet Language (XSL). Version 1.0.* W3C Working Draft 27-March-2000.

[112] W3C, the World Wide Web Consortium. *HTML 4.01 Specification.*

[113] W3C, the World Wide Web Consortium. *XHTML[tm] 1.0: The Extensible HyperText Markup Language. A Reformulation of HTML 4 in XML 1.0.* W3C Recommendation 26-January-2000.

[114] W3C, the World Wide Web Consortium. *Namespaces in XML*, January 1999. W3C Recommendation. http://www.w3.org/TR/REC-xml-names.

[115] W3C, the World Wide Web Consortium. *XML Path Language (XPath)*, November 1999. W3C Recommendation. http://www.w3.org/TR/1999/xpath.

[116] W3C, the World Wide Web Consortium. *XML Pointer Language (XPointer)*, December 1999. W3C Working Draft. http://www.w3.org/TR/xptr.

[117] W3C, the World Wide Web Consortium. *Extensible Markup Language (XML) 1.0 (Second Edition)*, October 2000. W3C Recommendation. http://www.w3.org/TR/REC-xml.

[118] W3C, the World Wide Web Consortium. *Resource Description Framework (RDF) Schema Specification 1.0*, March 2000. W3C Candidate Recommendation 27 March 2000. http://www.w3.org/TR/rdf-schema.

[119] W3C, the World Wide Web Consortium. *XML Linking Language (XLink)*, February 2000. W3C Working Draft 21-February-2000. http://www.w3.org/TR/2000/WD-xlink-20000221.

[120] Norman Walsh and Leonard Muellner. *DocBook: The Definitive Guide.* O'Reilly, 1st edition, October 1999.

[121] Eve Wilson. Links and structures in hypertext databases for law. In Antoine Rizk, Norbert A. Streitz, and J. André, editors, *European Conference on Hypertext, ECHT'90*, The Cambridge Series on Electronic Publishing, pages 194–211, Paris (France), 1990. Cambridge University Press.

[122] XSilfide (Serveur Interactif pour la Langue Francaise, son Identité, sa Diffusion et son Etude). http://www.loria.fr/projets/XSilfide/.

[123] Yi Xu. *An Incremental Apporach to Document Structure Recognition.* PhD thesis, GMD - Forschungszentrum Informationstechnik GmbH, 1998.

[124] Z39.50 Maintenance Agency. *ANSI/NISO Z39.50-1995. Information Retrieval (Z39.50): Application Service Definition and Protocol Specification*, July 1995.