

Technische Universität Hamburg-Harburg

**Link-Preserving Import of Hypermedia Document
Networks into a Content Management System**

Student Project Work

as a requirement of master's programme in
Information and Media Technologies

Submitted by:

Maria Ginsburg
Student No.: 20519

Submitted to:

Prof. Dr. Joachim W. Schmidt

Project Advisor:

Dipl.-Inf. Axel Wienberg

Hamburg, Germany

February 2003

Abstract

Subject domain: Hypermedia, Content Management

Problem definition: When stored inside an object-oriented Content Management System (CMS) like the CoreMedia Research Platform, a set of multimedia documents with embedded hyperlinks needs to be represented as a collection of structured objects with associations. The task of the student project is to design a suitable representation framework for the objects in the CMS, and to implement the import step for converting a Hypermedia Document Network represented as a web site into objects inside the CMS, while extracting and preserving structural link information.

Task limitation description: While the information stored inside the CMS has to be sufficient to query the link structure, and to regenerate a website representation of the document network (with possibly changed URLs), the export is not part of the implementation work. Also, the framework should be general enough to cover a wide range of hypermedia formats, but the prototype only has to parse HTML.

Acknowledgements

My special thanks go to Prof. Dr. Joachim W. Schmidt and research assistant Dipl.-Inf. Axel Wienberg at Software Systems Department of the Technical University of Hamburg-Harburg for providing the opportunity to work on this student project, for their essential and kind advice, encouragement and guidance throughout this student project work.

1. Introduction.....	5
1.1 Motivation.....	5
1.2 Task description.....	6
2. Hypermedia Structure	7
2.1 Hypermedia Elements.....	7
2.1.1 HTML Structures	7
2.1.2 SMIL structures.....	9
2.1.3. Macromedia Authoring tools' Presentation.....	10
2.2 Links.....	12
2.3. Hypermedia structure model.....	15
3. Object Structure in CMS.....	16
4. Architectural Analysis	17
4.1 Requirements	17
4.1.1. Requirements for the Import Tool.	17
4.1.2. Requirements for the Document Structure.	23
4.2 Document Structure	24
4.3 Importer Tool Architecture	29
5. Design	34
5.1. Design of XLink DTD	34
5.2. Design of the Specifications of the Components.....	38
5.2.1. Importer.....	38
5.2.2. Parser.....	41
5.2.3. Allocator.....	42
5.2.4. Class Diagram.....	45
6. Prototypical Implementation.....	46
6.1. Importer.....	46
6.2. Parser.....	48
6.3. Allocator.....	50
7. Conclusion	51
7.1. Related work	51
7.2. Summary and Possible Further Development	51
Bibliography:	53

1. Introduction

1.1 Motivation

The rapid growth of the Internet, the availability of advanced platforms for application development and streaming technology have enhanced the development and usage of rich media content – complex structures that contain different types of pictures, video, sound, textual information. In addition, new business models are evolving around trading and delivery of digital assets. Today, media companies are also realizing the necessity of managing and digitizing their assets in order to reduce costs and generate revenue. These new ways of doing business require expanding management capabilities for digital content. They are provided by content management systems (CMS) allowing creation, deployment and administration of rich media content.

Content management systems provide such key advantages as:

- quick response to changes of e-business processes
- automation of content publication
- separation of content and its presentation layout

Content management systems support the following functions for content creation and delivery:

- workflow functionality
- consistent control of design using templates
- centralized content repositories
- version control
- content adaptation for different users, browsers, devices
- dynamic rendering and publication of content structure

These functionalities make CMS a key tool for content creation and administration within information systems in order to reduce delivery time and cut down production cost.

Therefore the number and complexity of CMS-based Web portals, E-commerce applications, media editions etc. is constantly growing.

From another point of view, content management technology is a rather young compared to the development of content creation and authoring techniques. Many companies, such as Macromedia, Adobe, Microsoft, etc., are developing authoring tools for different kinds of media widely recognized among professional designers and developers of rich media content. This recognition is a major reason for the large number of hypermedia systems still produced with these tools. Although, providing a powerful and easy-to-use development environment, authoring tools do not provide further management functions, so content created and designed outside a CMS still requires CMS functionality in order to stay up-to-date, to allow group work, testing, staging etc.

1.2 Task description

The problem of import of already created hypermedia structures into CMS arises. Storing a hypermedia document network as one binary object would not be suitable as limited CMS functionalities could be applied to it, for example version control would only be available for the complete object, but modification of its structure or staging would be restricted. Therefore the internal structure of hypermedia content is crucial for the import process. The task of the project is to design a framework for the structured representation of a hypermedia structure in CMSs like the CoreMedia Research Platform (CRP). The development of the framework is a result of research on the following problems:

- which common properties hypermedia of different formats has and how it can be structured
- how relations between hypermedia documents are modeled
- which requirements CMS imposes on the description of the links and which functions of CMS can be used after the import of a hypermedia structure

As a result, the framework should be implemented for converting Hypermedia Document Network represented as a Web-site into objects inside CRP, while extracting and preserving structural link information.

2. Hypermedia Structure

“In Hypermedia applications, we can access and link together a rich collection of multimedia resources” [Son, J.-B., 1998]. Hypermedia is a style of building systems for organizing, structuring and accessing information around a network of multimedia nodes connected together by links [Conklin, 1987]. As the framework being developed for the import of hypermedia structure has to be generic, i.e. it allows the import of a wide range of formats of hypermedia, it is important to find out to which extent different structures support the notion of nodes and links. The Dexter model, which introduced these terms, was based on hypertext. Therefore, HTML structures can be mapped to the model more easily, whereas so-called fourth-generation systems create more complicated structures.

2.1 Hypermedia Elements

The section analyses different hypermedia formats in order to investigate their internal structure and determine how their structural elements can be represented as a set of nodes and interrelated links.

2.1.1 HTML Structures

The term Hypermedia is mostly used as a synonym for the term Hypertext, since many hypertext systems allows including media elements of different formats. For example, HTML structures were initially modeled only by hypertext documents. The latest versions of HTML allow combination of textual documents with graphics, sound, and moving images. This content complexity does not allow the determination of *node* as just a textual document in a simple HTML structure, composed only with text documents.

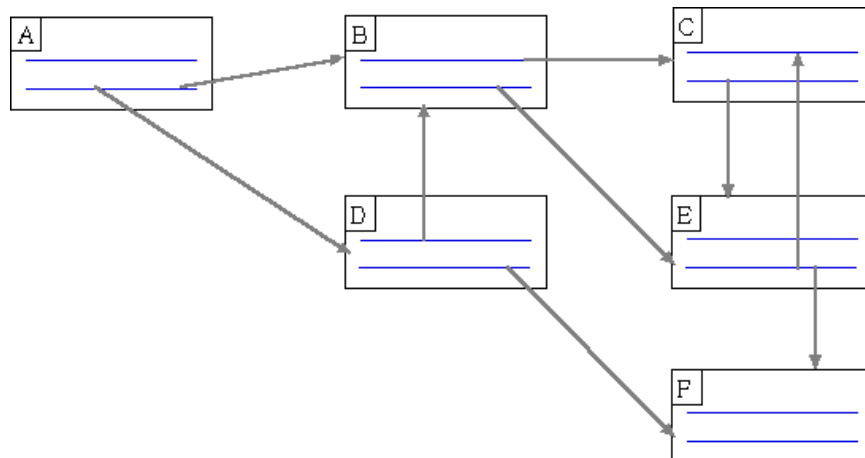


Fig. 2-1 Example of a hypertext structure [Son, J.-B., 1998]

Figure 2-1, which is a modified reproduction of Nielsen [Nielsen,1990], shows a small structure with six nodes and nine links. Once users start by reading the text marked A, the hypertext structure gives two options to continue: B or D. Assuming that users decide to

go to B, they can then jump to C or to E, and from E they can go to C or to F. Many more different paths than in this example can be created.

Figure 2-1 shows so-called external structure of hypertext [Slatin, 1990].

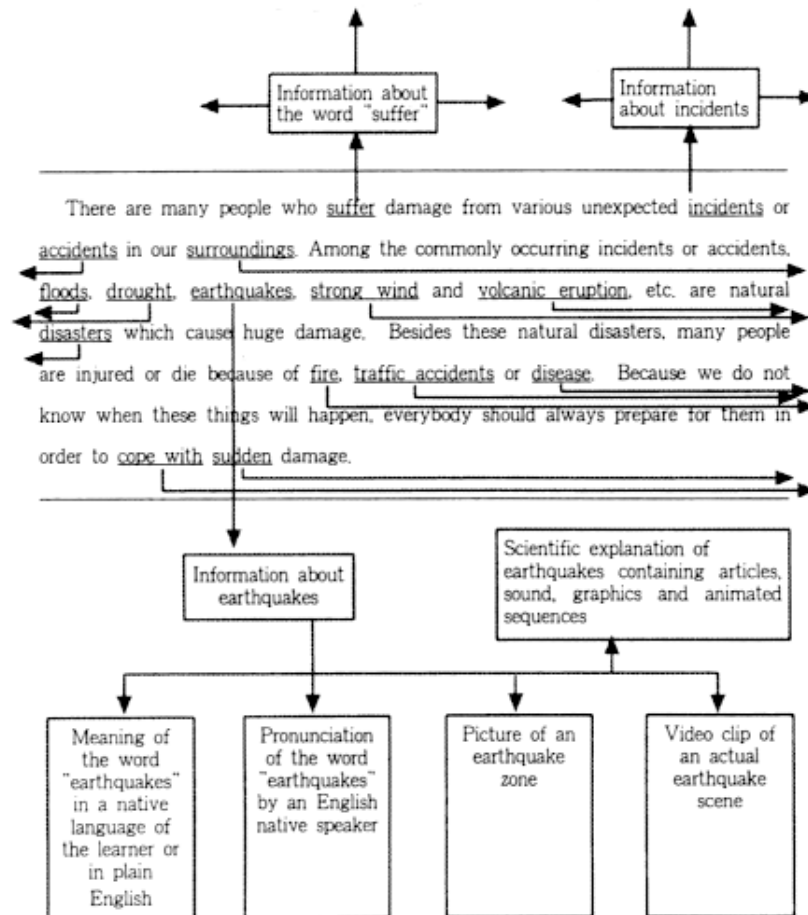


Fig. 2-2 Example of complexity of the hypermedia systems [Son, J.-B., 1998]

It presents the navigation lines, by which user browses the content from document to document. The internal structure of a particular document could be more complex, i.e. a document might contain a media element embedded in text. The document in figure 2-2 describes the word “earthquake”. It consists of textual information (meaning of the word), sound (pronunciation), graphics (picture of earthquake zone), and moving image (video clip). The presentation of these elements is embedded on the page and a user can access them by clicking on the text element “earthquake” in source document and viewing the target document.

There are different approaches towards complex documents. Philip Seyer [Seyer, 1991] proposed to consider them as a node which might have complex structure: "A node is usually a small collection of data organized around a single topic. Nodes can contain various kinds of data: graphics, audio, video, computer animated images, film clips of

animated scenes, digital sound and other kinds of information." But from a content management perspective, this approach seems to be not optimal, as media elements may require to be modified or retrieved independently from the document they are embedded in. These operations require the separation of these elements and thus keeping the linking between them. Therefore, HTML structure should be modified to more complex one (see Figure 2-3).

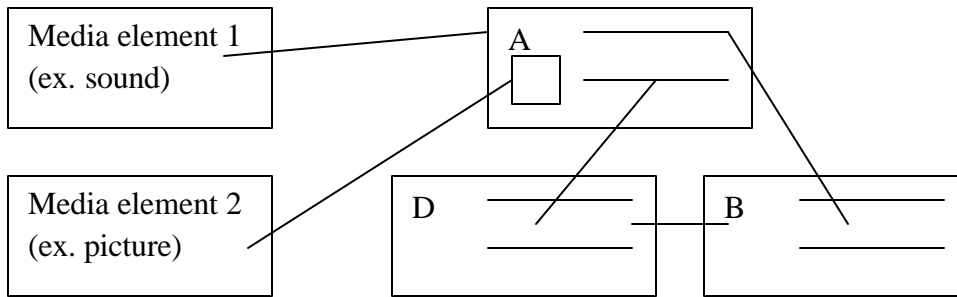


Fig.2-3 HTML document structure in terms of nodes and links

Here documents A, B, D and media elements 1, 2 are seen as independent nodes in a network, where the relations between nodes (embedded/non-embedded) are defined by the behavior properties of the links (see section 2.2 “Links”). That allows defining a node as following: node is multimedia element or a textual document.

```
Node := Media Element | Textual Document
```

2.1.2 SMIL structures

The Synchronized Multimedia Integration Language allows simple authoring of interactive audiovisual presentations. SMIL is typically used for "rich media"/multimedia presentations which integrate streaming audio and video with images, text or any other media type. SMIL is an HTML-like language [SMIL 2.0, 2001]. SMIL provides the following functionalities:

- positioning of media elements on the screen,
- synchronizing the presentation of time-dependent elements,
- displaying media according to user preferences, language, bit-rate.

SMIL is an XML-defined format, and its XML structure is time-based. This means, the primary semantic impact of composition and order of XML code is on the timing of the document. This contrasts to typical XML and HTML documents. A component's position in SMIL structure says *when* it appears, whereas a component's position in HTML and typical XML structure says *where* it appears [Rutledge, van, 1999]. It contains links to media elements which are synchronized or substitute each other by tag <switch>. These elements and description file can be seen as nodes of hypermedia being imported.

A SMIL file is taken as node and treated as document for storing in a CMS, because it contains all the necessary timing information for playing the multimedia presentation and can be the subject of modification inside CMS.

Complex content, such as hypertext structure, can be integrated within a SMIL presentation in the same way such as images or video fragments and other media by referring to it using a URL. SMIL can, for example, be used to define the synchronized display of a number of individual HTML pages by specifying the timing and screen location of each HTML page. The hierarchical structure of such a SMIL document would reflect the temporal structure of the overall presentation instead of the text flow of the various HTML pages.

Taking into account these facts, the node structure for SMIL presentation can be as following:

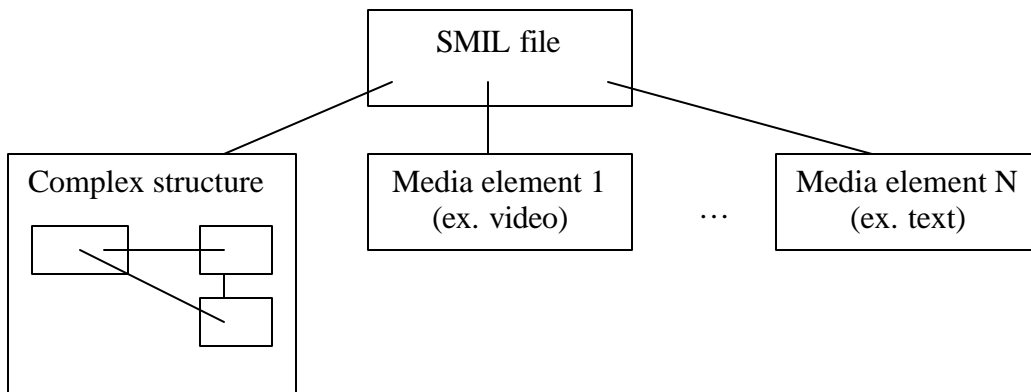


Fig. 2-4 SMIL document structure in terms of nodes and links

2.1.3. Macromedia Authoring tools' Presentation

Macromedia developed a new generation of multimedia authoring tools, Macromedia Flash and Director. Whereas early authoring tools were the basis for the Dexter model with its notations of nodes and links (for example HyperCard, PowerPoint), the multimedia presentations made by means of Flash or Director are much more difficult to map to this model. These authoring tools use different compression technology and are applicable to different types of multimedia presentations, but have a similar internal structure. Macromedia Director is taken as an example.

The concept of Director presentation goes beyond card stacks and pages of its predecessors. Instead, it uses a *timeline* which doesn't require sequential movement. The unit of timeline is a *frame* which consists of *channels* with media elements (*sprites*). The content of channels defines what is shown on the stage at a certain moment of time. It can be changed from frame to frame or stay the same waiting for some event, for example the execution of a Lingo script. Scripting language can control the appearance of the sprites on the stage, for example changing one element to another and playback head movement (jumping to another frame, looping etc). This complex behavior makes it impossible to

take frames as nodes (in analogy to cards), since the content of stage can be determined only at run-time.

Another idea is to use *Score* information (description of timeline playback) in analogy to a SMIL file, describing which components where and when appear in a presentation, and media elements as simple nodes. Here the problem is, that Director stores and refers to its “actors” (media elements which participate in the presentation) as cast members, which means that all the elements are stored in Director’s own library. There is no possibility to disassemble the library in order to import elements separately, especially if the library is internal and stored in the same file with the Score information. Video files are exceptions. They are not imported into the library and are stored as external files.

Director presentation (*movie*) can reference other movies in different ways. It is done by Lingo script, but from different places, for example movie script (“on the end of the movie 1 go to movie 2”), sprite script (“on mouse click go to movie 2”), and with different methods (“go to” and “play”). A movie can also reference or contain other hypermedia structures like HTML, SMIL documents, Flash movies.

As one can see, it is difficult to map Director hypermedia structure into nodes and links model of fine granularity as it is possible for HTML documents. It can be done on the high level of generalization (see Figure 2-5):

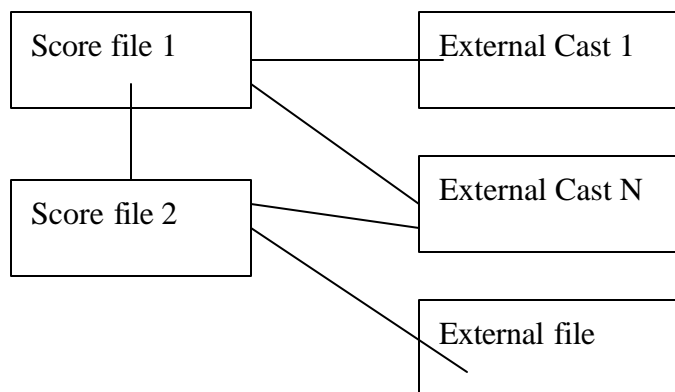


Fig. 2-5 Macromedia Director presentation structure in terms of nodes and links

Hypermedia produced in such authoring tools as Flash or Director is probably the most complicated structure. Many other structures such as for example an Adobe Premiere project can be mapped to the model similar to the SMIL one.

Analysis of different hypermedia structures leads to a conclusion that all of them can be considered as a network of nodes with interrelated links, but the granularity, or elements which can be seen as a node, depends on the type of the structure.

2.2 Links

The subject of the previous subsection was the node structure of different types of hypermedia. In order to create the complete model of hypermedia the link structure should be analyzed. This chapter surveys the taxonomy of hyperlinks, their properties and behavior in order to create a model of the link structure suitable for importing hypermedia into CMS.

Hypermedia, like all other information systems, contains various types of relationships between elements of information. Examples of typical relationships include

- similarity in meaning or context (*Bush* relates to *Hypermedia*),
- similarity in logical sequence (*Chapter 3* follows *Chapter 2*),
- similarity in temporal sequence (*Video 4* starts 5 seconds after *Video 3*),
- similarity in containment (*Chapter 4* contains *Section 4.2*).

Hypermedia reflects these relationships as links connecting the various information elements, so that these links can be used to navigate within the information space. [Son, J.-B.,1998]

Within one hypermedia structure *remote* links connect the current document, or node, with another one, whereas *local* links connect pieces of information within a node. For instance, a journal article structured as a single hypertext node might have a table of contents at the beginning where each entry is a link to that section of the article.

Different taxonomies of links can be created in order to analyze their utilization. One of these taxonomies can be based on the type of information relationships. In particular relationships are divided into:

- those based on the organization of the information space (structural links)
These links reflect information structure, for example in a book they form linear structure (from chapter to chapter) and hierarchical one (table of content).
- those related to the content of the information space (associative and referential links)
An associative link is an instantiation of a semantic relationship between information elements. They are based on the meaning of different information components. The example could be a cross-referencing within books (“for more information on *X* refer to *Y*”).
A referential link provides a link between an item of information and an elaboration or explanation of that information.

Most hypermedia systems do not provide a mechanism for differentiating between these link types, therefore the more useful link taxonomy is one based on the mechanics of links. They are

- the number of sources and destinations for links (single-source single-destination, multiple-source single-destination, etc.),
- the directionality of links (unidirectional, bidirectional),
- the anchoring mechanism (generic links, dynamic links, etc.).

Links with multiple sources and destinations (endpoints) connect a set of related nodes. The user might choose between different available destinations. When he initiates the traversal of a link with multiple endpoints, he can be prompted to choose one of the options by dialog box or pop-up menu.

The direction of a link is defined by the author's expectation how users follow the link. Readers of a link directed from A to B, for example, are expected to proceed to B after having read A. With link directionality, hyperlinks can be either uni- or bidirectional. Some hypermedia systems, such as the World Wide Web, support unidirectional links only. The usual corollary of unidirectional links is that they cannot be traversed "backwards" from destination to source. This means that it is impossible to see which links point to a particular node. Such information is very useful if one wishes to know how many times an article is "cited" or if one wishes to move a node and notify all the links pointing to it of the new destination.

Generic links link any occurrence of an anchor in any source document to a specific anchor in a destination document. Generic links greatly improve authoring efficiency (a link only needs to be created once, and would even be added to new documents) and maintenance (only one link has to be changed if the destination changed). A trivial example of generic links is a dictionary lookup facility - every appearance of a dictionary term in any document would be linked to the definition of the term. [OHP, 1996]

Dynamic Links link specific anchor in a source document to a number of dynamically computed destinations (for example by using a text retrieval search). [OHP, 1996]

An **uni-directional one-to-one link** can be used as a basis for more complex linking assuming that:

- A bi-directional link is equivalent to two uni-directional links that point at each other.
- n -ary links are equivalent to a collection of uni-directional links that share a common source point.

By this assumption a first approximation of definition of link model can be made: a link can be seen as an ordered pair of two nodes in hypermedia network, where the order defines the directionality of the link.

For the import of hypermedia into a CMS it is not sufficient only to know that two nodes are connected. Since a document can be modified or rendered for publishing, there are at minimum two decisions that must be made as a link is constructed between two nodes: what will be the point, the "explicit object" that activates the link in a source document, and what will be the point in a destination node.

Therefore the notation of position is vital in order to obtain a proper specification of a hyperlink. In accordance with the Dexter model, a designated and uniquely addressable location within a media object is defined as *anchor*.

Determining an anchor in media objects requires a suitable measuring system. For example, counting words in a text can be applied to determine a specific location within text; a coordinate system helps to determine areas within a picture. For those media that are arranged both in time and space, for example video, notation of frame, time or even fuzzy notation, as “shortly before”, can be applied. The only requirement for a measuring system is the uniqueness of the location address.

For hypertext structures, a link goes from a certain anchor in a source node, such as word or button, towards a whole node or an anchor within a node. In other structures, such as Macromedia Director presentation, the whole node can refer to an anchor in another node or a whole node. That requires a new term *endpoint* based on the anchor notation to be introduced. For the purpose of this project, a endpoint shall be defined as an anchor in a certain node or the node itself.

Consequently, link model definition is refined as an ordered pair of two locations. But this model does not reflect a behavioral aspect of link what will be important when a hypermedia structure is exported. Therefore link should contain some description element (arc) for link properties, such as link title, type, behavioral attributes.

The title property is useful to increase users’ understanding of links and to recognize and indicate the destination page and so reduce disorientation.

The wide range of link types can be classified, for example:

- hypertext links [Grabinger et al., 1992a],
- sequential links [Grabinger et al., 1992b],
- relational links [Grabinger et al., 1993a],
- support links [Grabinger et al., 1993b].

Link typing enables the designer of hypertexts to characterize links according to a predefined set of types. These might be rhetorical classifications: ‘comment’, ‘amplification’, ‘refutation’, and so on [Landow, 1989]. They might also be type of linked data: movie, sound file, text document, etc.

Behavioral attributes suggest to applications how the referred resource is associated with the current node. They determine:

- how the content should be displayed when the link is activated;
- whether the link should be traversed automatically or whether a specific user request is required.

These attributes are application dependent, however, and applications are free to ignore the suggestions.

The final definition of a link is: **A link is a structure containing a pair of endpoints, called source endpoint and target endpoint (local or remote), and a description of relation between two resources** (see Figure 2-6).

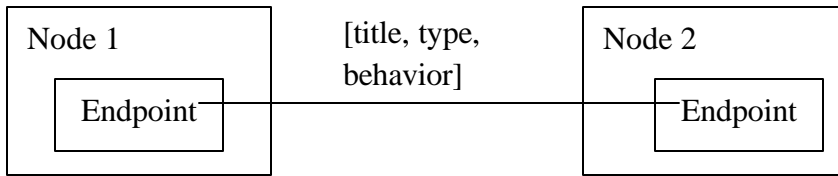


Fig. 2-6 Example of a link between two nodes

2.3. Hypermedia structure model

To summarize the analysis of hypermedia structure its formal model is created (see Figure 2-7).

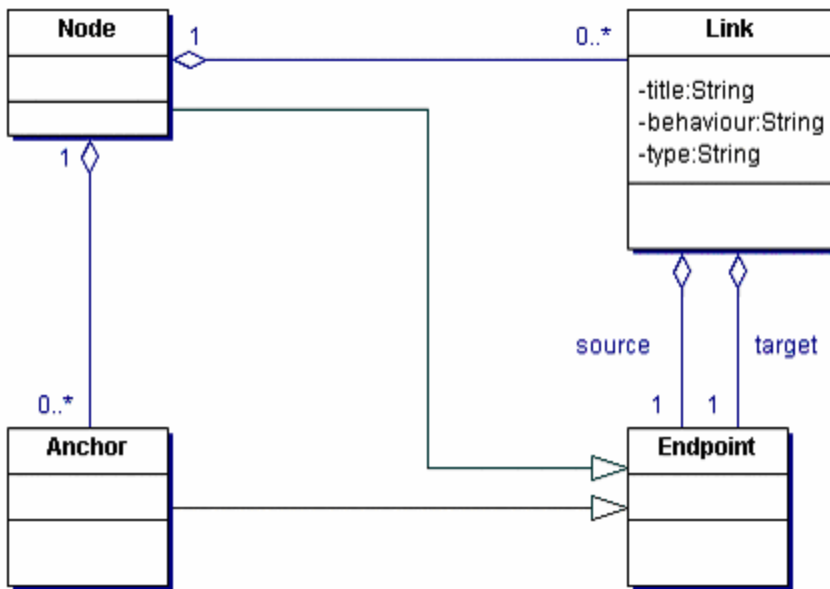


Fig. 2-7 Hypermedia structure model

In hypermedia structure links are stored inside nodes. A node contains an arbitrary number of links. An anchor is a unique point inside a certain node. The number of anchors in a node is arbitrary. A link consists of two endpoints, source and target, and contains properties describing link behavior, title, and type. Endpoint is either an anchor or a node itself and arc is a link properties description.

3. Object Structure in CMS

In the previous chapter, a formal model of a hypermedia structure of an arbitral format was created. This structure is the basis for processing a multimedia document network during its import into CMS. Now, it is important to investigate the structure of the CMS into which documents will be imported, in order to clarify the requirements for the mapping framework.

As described in the task, import should be done in CoreMedia Research Platform. It uses an object-oriented meta-model to express the content.

Figure 3-1 shows simplified UML class diagram for the CRP meta-model. Available classes of properties include atomic properties such as string and integer properties, media properties bound to application specific binary values, and link properties bound to collections of links to other objects. Each object has a declared type. A type declares properties, and can inherit further properties from a single parent type [Wienberg A., 2002].

A declaration of a link property includes the expected type for targets of the contained links, cardinality and uniqueness constraints, and may indicate that the link collection is keyed or ordered, turning it into a map or a list, respectively. The available link collection types therefore correspond to those in most modern programming languages: simple link, link set, link list, link map.

Important here is that links between the objects are expressed explicitly as references to objects.

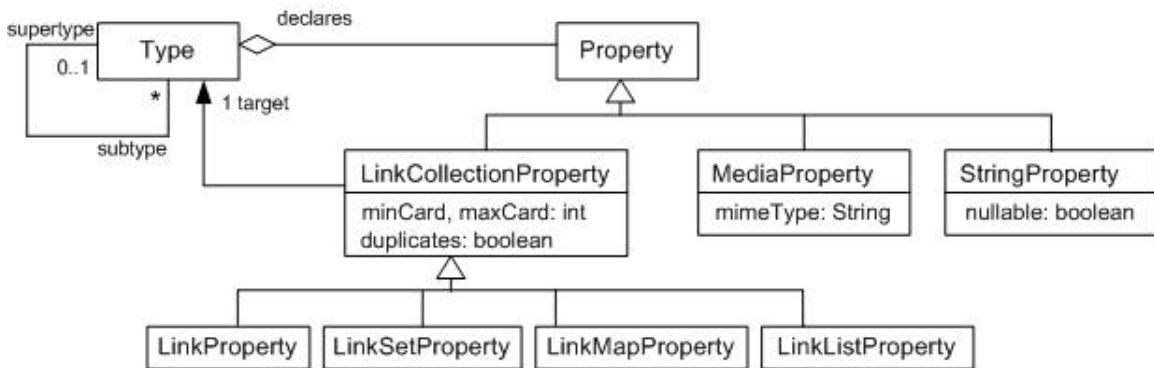


Fig. 3-1 Simplified UML class diagram for the CRP meta-model

4. Architectural Analysis

The architectural analysis creates a model which is an abstraction of the Importer Tool to be designed. The model is based on the use cases and the requirements to its functionality to create the architecture of the tool. As results of this phase, the specification of document structure and the specification of component structure are obtained.

4.1 Requirements

In order to create the architecture of the Importing Tool requirements to its structure and functionality are identified.

4.1.1. Requirements for the Import Tool.

This subsection analyzes use cases in order to determine functional requirements to Importer Tool. Non-functional requirements are investigated.

4.1.1.1. Use Cases

In order to find out which requirements CRP imposes on the import process, a number of use cases will be analyzed.

Import of hypermedia structure can be classified as continuous and incremental import. Continuous import is the import of the whole structure, starting from some initial node and finishing when the last node is imported. Incremental import is the addition of new nodes to already imported structure; it is a special case of continuous import (see section 5.2 Design of the Specifications of the Components).

Figure 4-1 shows use cases of document processing in CMS. It contains:

- Import of the document structure
- Edit of a document
- Modification of a link
- Moving a document
- Deletion a document

Further, activity diagrams refine these functions.

Figures 4-2 and 4-3 show activity diagram of continuous and incremental imports. Three main functions can be deduced from them:

- parse a document structure
- add a document to CRP
- link documents

The designed framework should refine these functions.

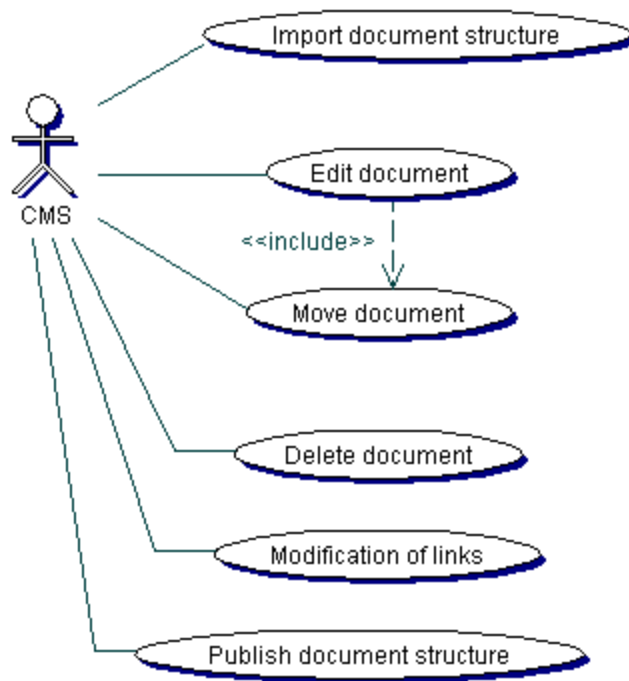


Fig. 4-1 Use cases of document processing in CMS

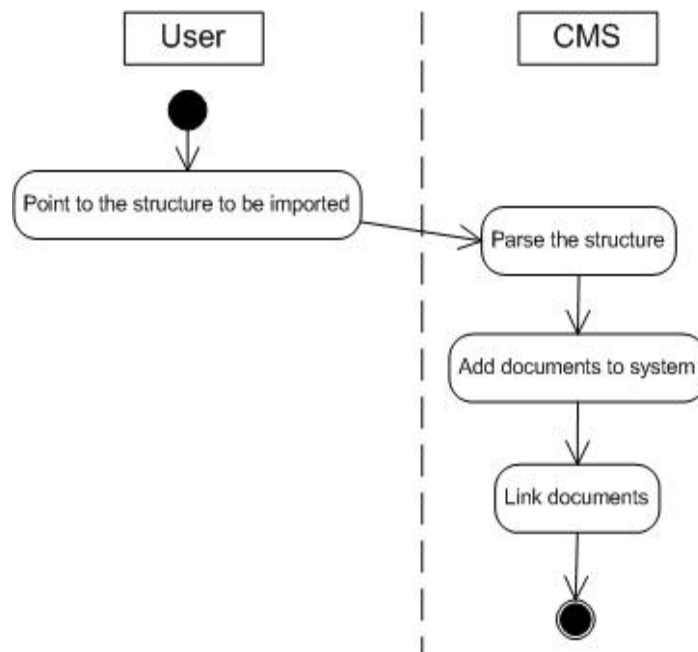


Fig. 4-2 Continuous import

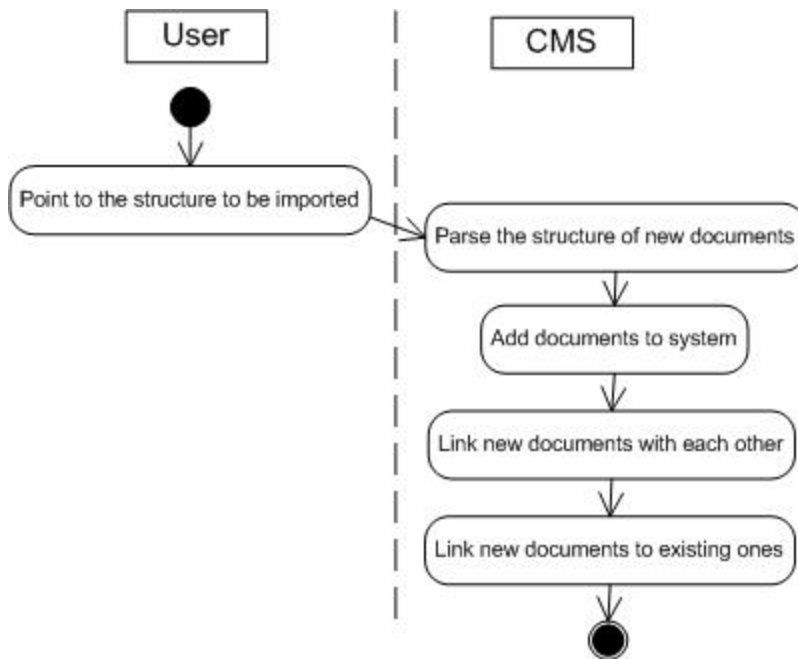


Fig. 4-3 Incremental import

Figures 4-4, 4-5, 4-6, 4-7 show core functions of CRP that might be applied to the imported hypermedia structure during its lifecycle after its import – editing, moving, deletion of a document, modification of a link and publication of the whole structure.

Editing of documents (see Figure 4-4) requires two actors: a CMS and an external editor. The external editor is a content-dependent application that provides tools for modification of a specific document. Modification of a document can concern both changing the content and changing the links. Therefore the newly edited document should be re-imported to the CMS so that the CMS’ link structure can be updated and if there is a link to a new document it can be added to the CMS. In this case, incremental import is required.

Moving of a document (see Figure 4-5) implies changing its location inside a CMS and changing endpoints of the links that refer to this document. In the special case of CRP, changing link endpoints is not necessary, since a link between documents is a reference and doesn’t depend on its location.

The deletion of a document (see Figure 4-6) requires validation of the links referring to it. To define these links, CRP uses a reverse link set (a dynamic link according to 2.2) containing the links pointing to the current document.

Modification of a link (see Figure 4-7) implies the change of the link endpoint which requires changing document reference inside CMS. If a user modified a link by changing its property, it may lead to changes in the document template or in some structure that describes these properties.

Publication of a document structure (see Figure 47) requires rendering of the link structure corresponding to the new location. So original links inside a document belonging to this structure should be substituted with the new URL according to the changes that probably occurred (editing, moving). The loss of information during the substitution of the links should be minimized. Therefore, during the parsing of the document structure it is necessary to extract not only the information about which documents are connected, but also information about link properties.

As one can see from the use case, CRP provides the methods for moving and editing documents, modification links, and publication of the document structure on the basis of its link structure. So the import tool has to preserve original link structure according to CRP's meta-model (see Document Structure Requirements).

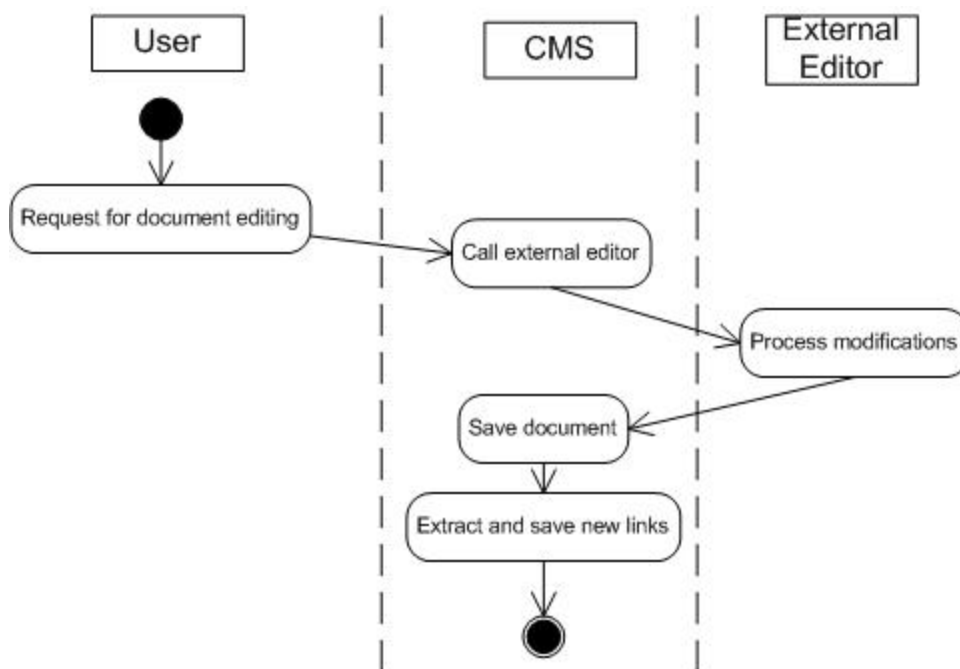


Fig. 4-4 Editing a document

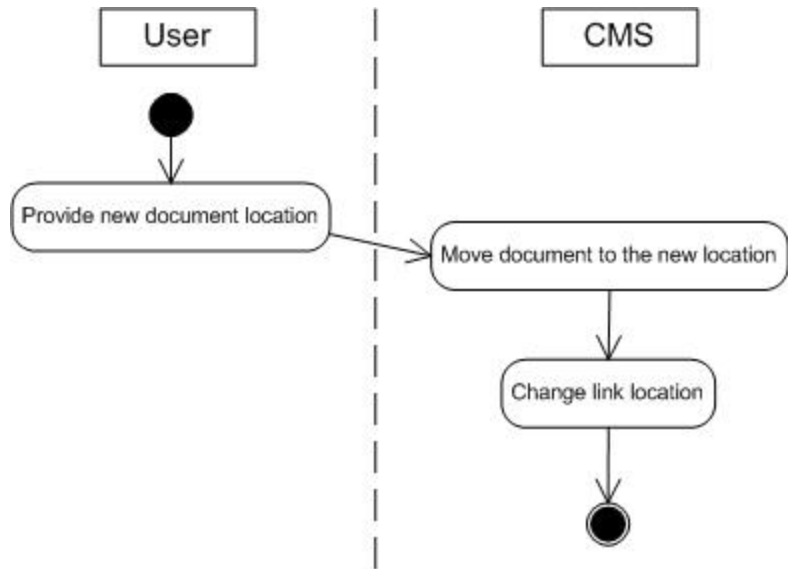


Fig. 4-5 Moving a document

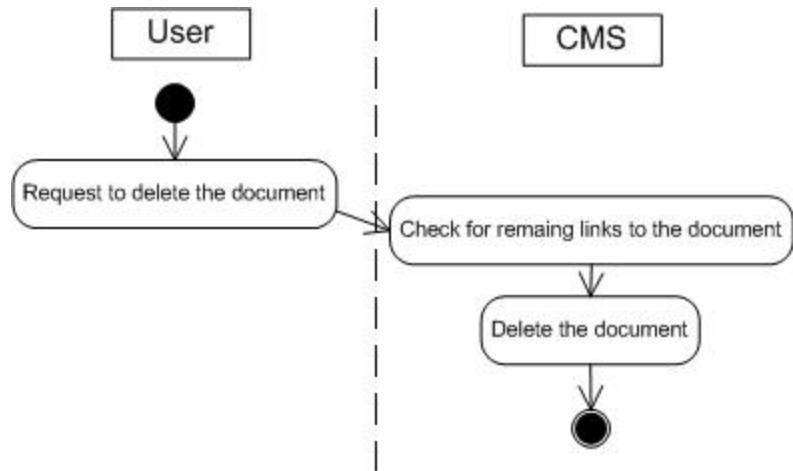


Fig. 4-6 Delete a document

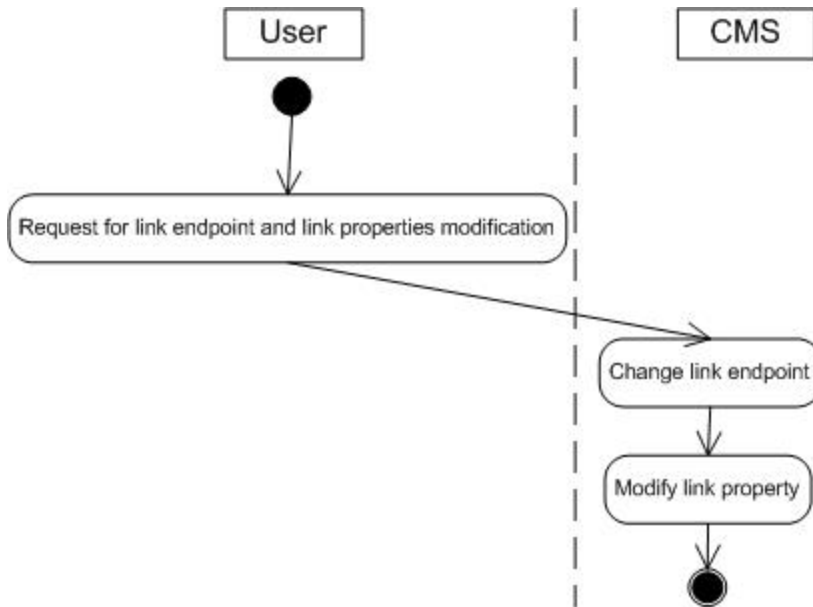


Fig. 4-7 Modification of a link

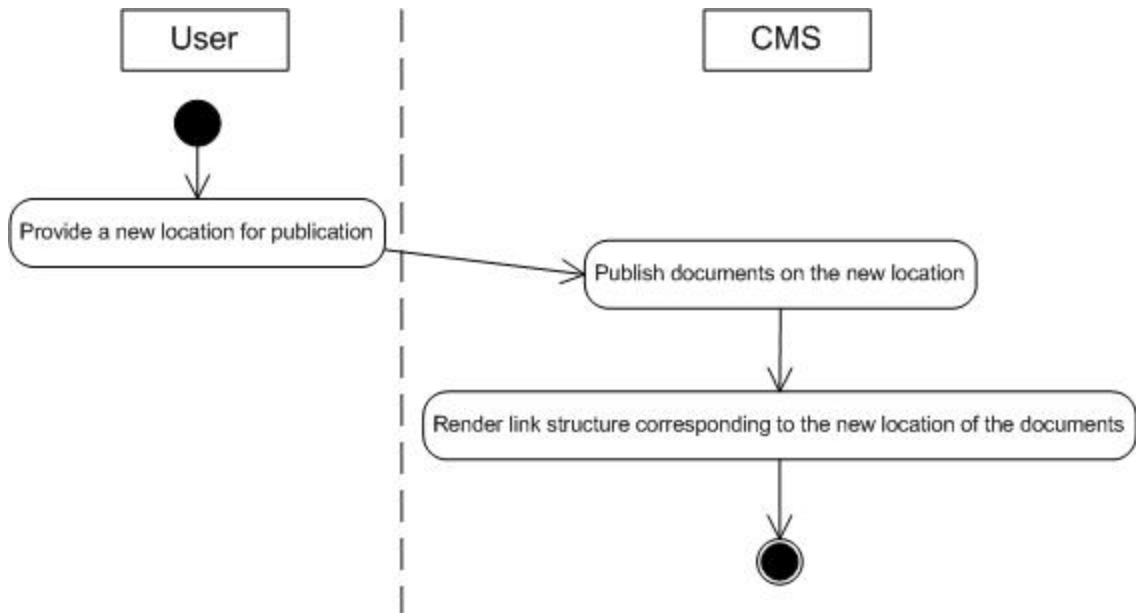


Fig. 4-8 Publication

4.1.1.2. Non-Functional Requirements

Interruptability. The interruption of the import of the document structure should not cause failure of the whole process. It should not require the repetition of the whole import process. Interrupted continuous import leads to the following incremental import. It should add the part of the structure that was not imported because of the failure to the already imported one. In order to meet the requirement of interruptability

- import of the whole structure should be idempotent by atomic import of separate documents. This is provided by introduction of incremental import.
- import of one document should meet the requirement of transactionality

Transactionality. The import of a document should conform the properties of transactions :

- Atomicity: when a separate document is being imported to CMS, the result should be either a) the whole imported document is stored in CMS or, in case of failure, b) it can be imported by the next round of incremental import. Imported documents keep the full information. Documents only containing partial information are supposed to be re-imported.
- Consistency: the effects of the import must preserve required system properties. For example, newly imported document shouldn't refer to a document that doesn't exist.
- Isolation: it is assumed that there will be no two importing processes running at the same time, therefore isolation property is not critical for this case.
- Durability: once import is committed, the change must persist, except in the face of a truly catastrophic failure.

Modularity. In order to provide flexibility, i.e. to adapt import for different hypermedia formats, modular structure is required.

4.1.2. Requirements for the Document Structure.

In previous chapters the formal model of hypermedia structure, being imported, and the meta-model of CRP were investigated. The document structure should be created accordingly to the CRP meta-model so that it can reflect the structure of hypermedia document network, meaning that it has to allow:

- storing set of documents (nodes),
- defining anchors inside a document,
- preserving link structure, so that it reflects original relations between the documents and allows further editing, moving the documents and publishing the whole structure with changed URLs.

The decision on the document structure should bridge the difference between the two models in link storage. Hypermedia documents define links inside themselves. That allows determining the starting point of a link inside a document and the link properties.

At the same time, object-oriented CRP expresses links explicitly by the references between objects. The questions to be answered while designing the document structure are:

- how the start anchor of a link can be determined,
- how the end anchor of a link can be determined,
- how the properties of a link can be stored so that it would be possible to associate a corresponding sets of properties and links during the further structure regeneration.

4.2 Document Structure

Based on the requirements of CRP described earlier, the document structure can be developed.

As it was already said, CRP organizes its information by resources of different types. A resource of *default document* type is taken as a base for storage of an imported document. Each instance of this resource type defines a node. It is identified by ID number CRP automatically assigns to each object. This resource should contain the properties to describe:

- the content of the document
- the locations of the links inside the document
- the links to other documents or anchors in documents
- the properties of the links

The content of the imported document is stored as a media object in media property (*content*) bound to application specific binary value.

Links between documents are the references between objects expressed by the link collection property (*links*). As each document might be connected with many documents, link set, link list or link map should be used.

Link collection property defines that the current document refers to another one. But it contains no information about:

- whether it refers to the whole document or a part. If it refers to the part of a document, it is impossible to determine the position of the part.
- The kind of the relation between documents. For example, it is possible to differentiate the link by creating different link collections for every type. But with the growing number of link properties the structure can become bulky.

XLink structure is used to describe links between documents. It uses XML syntax to create structures that can describe links of different complexity. It allows expressing links that reside in a location separate from the linked resources with a description of its behavior (see section 5.1 Design of Xlink DTD). Xlink structure that describes the links of the document is stored as a media object in media property (*xlink*).

The problem of definition of a certain location inside a document is divided into two sub-problems (see Figure 4-8):

1. Definition of a link source endpoint in case it is an anchor in a source document
2. Definition of a link target endpoint in case it is an anchor in a target document

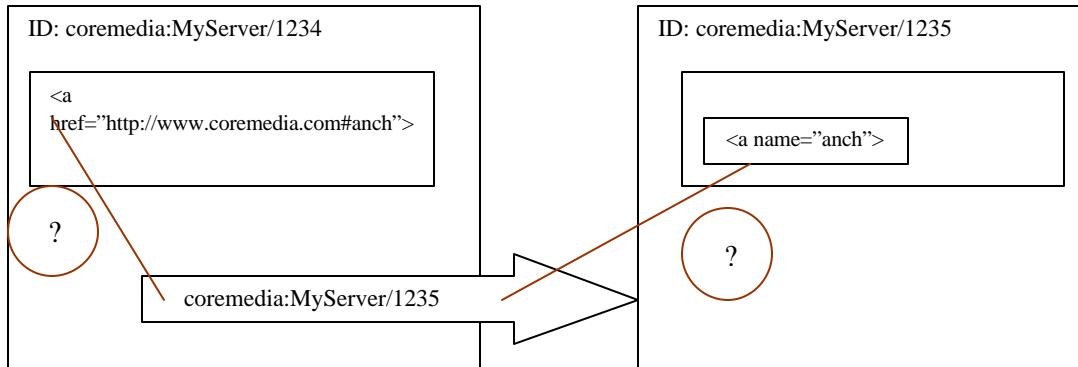


Fig. 4-8 Representation of nodes relationship inside CRP

Starting from the second problem: as a link collection cannot identify whether the link refers to the whole document or to only its part, an anchor description should be done in XLink structure in following forms:

- *coremedia: MyServer/1235#anch*, if the target endpoint is an anchor
- *coremedia: MyServer/1235*, if the target is the whole document,

where *MyServer/1235* is the ID of the referred document inside CRP. Since different hypermedia formats have to be taken into account, it cannot be assumed that anchors are always marked as in HTML (e.g., SMIL element can refer to a particular part of video file by time span or number of the frame). Therefore *anch* is defined as some unique identification of an anchor inside a document. The particular expression of *anch* is content dependent (frame, time span for time-dependent media, coordinates for graphics etc.). It is required that it uniquely identifies the location of the link endpoint inside a document.

This architectural decision raises a further problem to be solved:

- The editing of the target document might require adjustment of locations of anchors, even though it doesn't know which targeted anchors it contains. For example, a document could refer to a 10th frame of a video sequence. After edit of the video new frames are added or some frames were cut, so that 10th frame doesn't contain the same information. Therefore a link to the video should be deleted as dangling or adjusted corresponding to the new location. It is possible if all the links that point to this document are known and could be adjusted. CRP allows determining reverse links and XLink structure provides information about anchors. This process will not be implemented in the framework and is the subject of further development.

To define link's source endpoint in a source document several solutions were offered:

1. To change its structure by substitution the original links by the ID of the link while parsing the document.

In this case, each link is assigned to some unique ID by which link collection is keyed. A parsing tool should be created to map uniquely the point in a document marked by ID and the corresponding link during the link structure regeneration or

the editing of the document can. This solution excludes the necessity to store XLink structure in CMS, since all the link properties are described inside the document.

The following objection could be raised:

- Internal structure of the original file is changed at least two times, during import and during publication; it can be required also for modification of the links.

This is not preferable, since often changes of internal structure could corrupt its content.

2. Treat the source endpoints as anchors.

That means, these points can be described as a target anchor with some anchor ID, where ID can be:

1. Byte position in the document. In this case, editing of each document content will influence the position of the link
2. Number of the link in file. Assuming that links are not added or removed this reference is unique and stable.
3. ID definition is content-dependent like in the solution with target anchors. E.g. in Director format – <frame, sprite>, HTML – name of the link or <paragraph, word>.

This solution is preferable, since it doesn't change the original document and provides flexibility of separation of content and link structure.

In this case, XLink structure should describe not only the target endpoint, but also the source one. Since it will be stored with the document which links it describes, it is enough to define the starting point as *#anch*, without document ID. A link in link collection if keyed by *anch* of link departure point can be uniquely related to the description of its property in XLink structure.

Technically, anchor ID for a source document is a string of location description, which is understandable by interpreter, corresponding to this type of file (number of frame in video, name of the anchor in HTML etc). For some kinds of document the ID is created artificially or cannot be created at all, e.g. Director project doesn't determine link to element in external library explicitly.

That makes us define 3 levels of service that Import Tool can support:

1. Store links that reflects only relations between documents and don't identify endpoints. This level allows performing query over document structure.
2. Links identify endpoints, but they cannot be removed or added. In this case anchor IDs are tightly related with the number of links or their structure. Change of one anchor ID can lead to changes in all others endpoints.
3. Links identify endpoints, but they can be removed or added. In this case anchor IDs are not related with the number of links or their structure. Endpoint can be found by names, query etc.

The instance of document type can look as follows (see Figure 4-9):

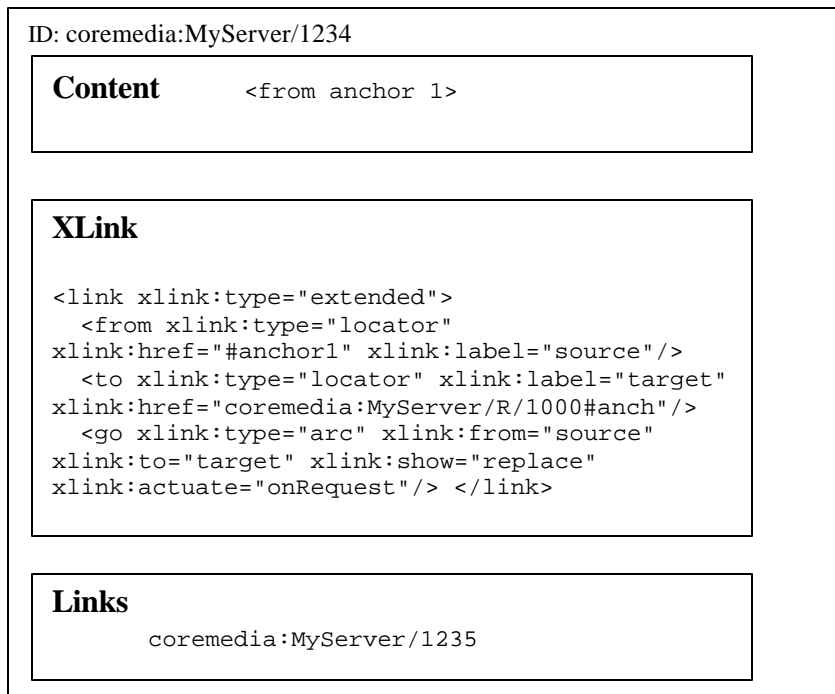


Fig. 4-9 Example of *default document* type resource. For XLink DTD see section 5.1

The combination of Link set and XLink structure looks redundant, since both provide information on which documents the current document refers to. Link set is CRP standard type which allows the CMS to perform functions such as link check, request of documents that refer to the particular document, etc. It provides the object-oriented association between documents. XLink structure plays a role of link property descriptor, defining anchors and the behavior of link. That is necessary for further export. The CMS has no own function to process XLink structure, hence a model of modification and export modules should be created; that is a subject for a future work.

In order to support import the process, some “service” properties should be added into the structure of a document.

- URL. String property that stores the initial URL of the document. It is necessary, since during the incremental import the importer should find out which documents of the structure being processed have been already imported. The easiest way is to compare the URL of the document to the URL of the imported documents (details see below).
- ImportSet. Index property which is necessary to perform search among imported documents (details see below).

The structure of resource of *default document* type is depicted in Figure 4-10.

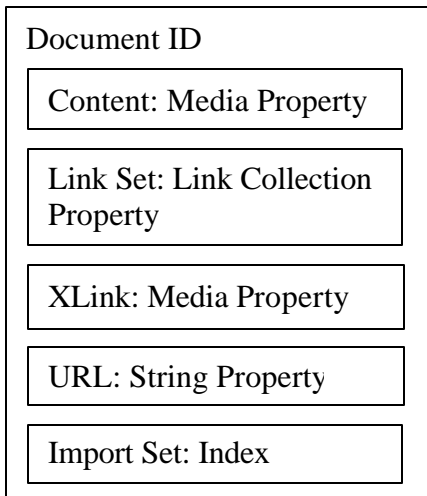


Fig. 4-10 Default document structure

To perform a search among imported documents, a resource of *ImportSet* type should be created. It contains the references to all imported documents and creates their index by *URL* property.

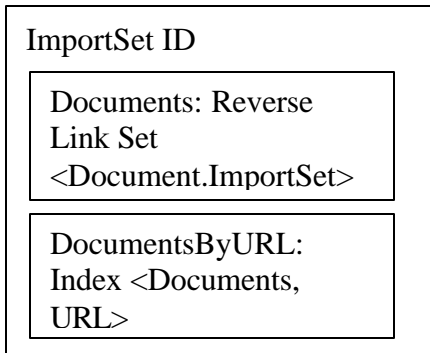


Fig. 4-11 ImportSet structure

The general document structure is represented by the following diagram:

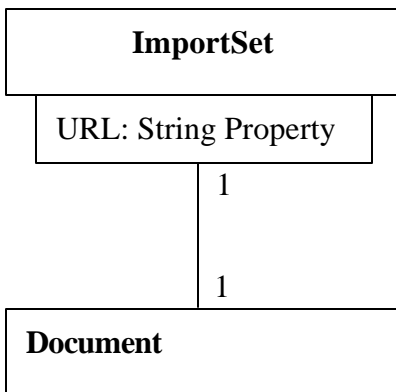


Fig. 4-12 Butch diagram for general document structure

4.3 Importer Tool Architecture

Importer Tool (IT) transfers original hypermedia structure placed in World Wide Web or in file system of a computer into the document structure as described in the previous chapter. The general algorithm of its work is depicted on Figure 4-13.

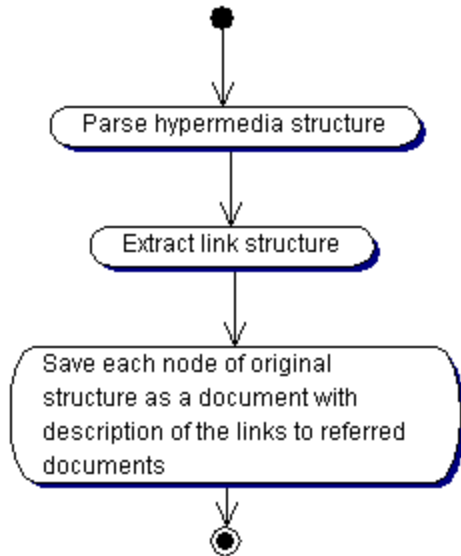


Fig. 4-13 General algorithm of Importer Tool

Parsing a hypermedia structure implies analysis of each document in the network in order to figure out which documents it refers to and how the links can be described. As it was said before, original content of a file will not be changed, IT should only extract and save the information about the link structure. Thus it is possible to separate the content stored as Media property in *default document* in CRP, and link structure described by Link Collection and XLink properties.

IT operates on the two levels of the Dexter model. This model defines *storage* and *within- component* layers of hypermedia. The storage layer describes a “database” composed of node/link network. The within- component layer concerns the internal structure of the nodes. Similarly, IT is divided into the two components. The *Importer* module focuses on the mechanism of mapping the hypermedia structure defined in chapter 2 to the CRP document structure. It treats the nodes as a generic container of data. Importer makes no difference between text, graphic or animation nodes. For each node of the original document network it fills up the properties of a corresponding document in CRP.

Extracting the link information from a node is a content-specific operation. It is the responsibility of the *Parser* module. It works with the certain hypermedia type and understands its format. The structure of Parser is not elaborated during the framework

design, since for each format it is different. The communication with Importer and the format of information they exchange should be specified in order to provide plug-in flexibility. Thus Parser can be treated as a “black box”, which processes somehow a file of certain format and gives the description of its links as an output. This description is taken as incoming information for Importer. It analyses it and stores in the corresponding properties of the document in CRP.

Two solutions for Parser – Importer communication are offered:

1. Link description formed by Parser is an XLink structure with links in form of reference to other documents in the terms of CRP, i.e. pointing to the ID of a document inside CRP. Importer stores it in *XLink* property and extracts pairs <source endpoint, target endpoint> from this structure in order to form *Link Set*. In order to form this description Parser should apply to CRP each time it finds a link in source document to allocate an empty document for the target document. Allocation is done by the third module, *Allocator*, which acts as “service” module for both Parser and Importer. By the given ID of the document it returns URL stored in *URL* property and, vice versa, by given URL returns the ID. Importer needs reference information provided by Allocator in case of incremental import to find out which documents linked with the document being imported are already in the system.

The components model of this solution can be depicted as following:

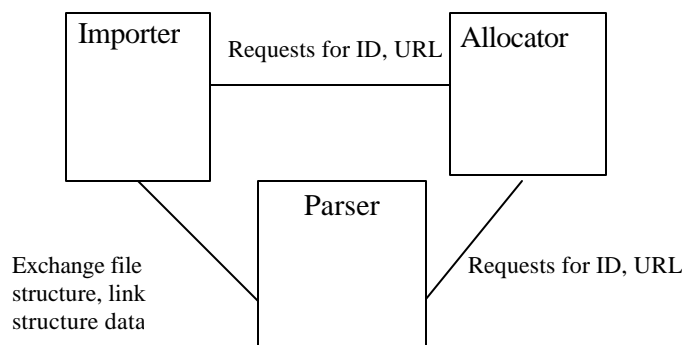


Fig. 4-14 Component model for the first architectural solution

Advantages:

- From XLink received from Parser, Importer can directly extract the information about the ID of the document where the current node should be imported and can directly form Link Set.

Disadvantages:

- Parser requires the information from CRP about document ID. Therefore, it can not be seen as an independent component and does not have the desired plug-in flexibility.

2. Link description formed by Parser is an XLink structure with links in form of the absolute URLs. Importer extracts pairs <source endpoint, target endpoint (URL)> from this structure. In order to form *Link Set*, it asks *Allocator* to resolve the URL into the document ID. Importer changes the XLink structure received from Parser by substituting URLs with corresponding IDs and stores this structure in *XLink* property of the

document. Still, Importer needs reference information provided by Allocator for incremental import.

The components model of this solution can be depicted as following:

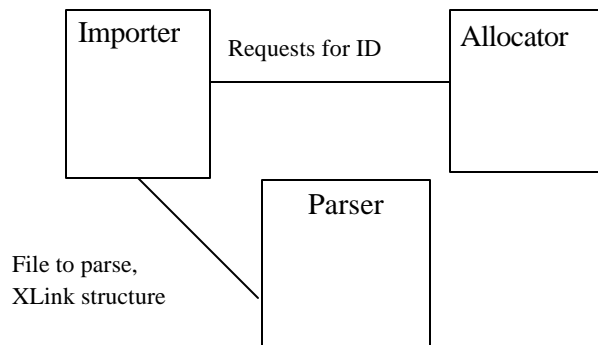


Fig. 4-15 Component model for the second architectural solution

Advantages:

- Parser is an independent component and does not require internal information of CRP
- Importer does not need to ask Allocator for resolution of ID of a document into corresponding URL

Disadvantages:

- Importer needs to modify XLink structure

Obviously, the second solution requires Importer to provide more functionality. Though, it provides more independency and flexibility to Parser realization. The second solution is chosen for the further design and implementation.

Thus, the functionality and the format of exchange information of each component can be defined.

Importer

Functions:

- Initiates both continuous and incremental import. Before the import starts, it has to define which type of import is required
- Goes through the hypermedia structure treating it as a cyclic graph which nodes are subject to call Parser
- Saves node content in Content property of the document
- Saves XLink structure in XLink property of the document
- Analyses and modifies XLink structure in order to form Link Set property of the document

Incoming data:

- Initial file of hypermedia structure to be imported (for example, index.htm in HTML)
- Base URL to determine file structure in CRP
- XLink structure
- ID of a document from Allocator

Outgoing data:

- A file to be parsed as an Input Stream

Parser

Functions:

- Analyses the given Input Stream in order to find the links
- Defines the start endpoint of the link in the source document as an anchor (if it is applicable for a given hypermedia format)
- Defines the endpoint of the link in the target document as an anchor or as a whole document, resolves relative URLs into absolute ones
- Forms XLink structure

Incoming data:

- A file to be parsed as an Input Stream
- The base URL

Outgoing data:

- XLink structure

Allocator

Functions:

- Find ID of the document by given URL
- Allocate an empty document with given URL so that file structure, in which hypermedia is organized, is preserved in CRP

Incoming data:

- Requests for ID

Outgoing data:

- ID

The sequence diagram depicts the components interaction scenario (Figure 4-16)

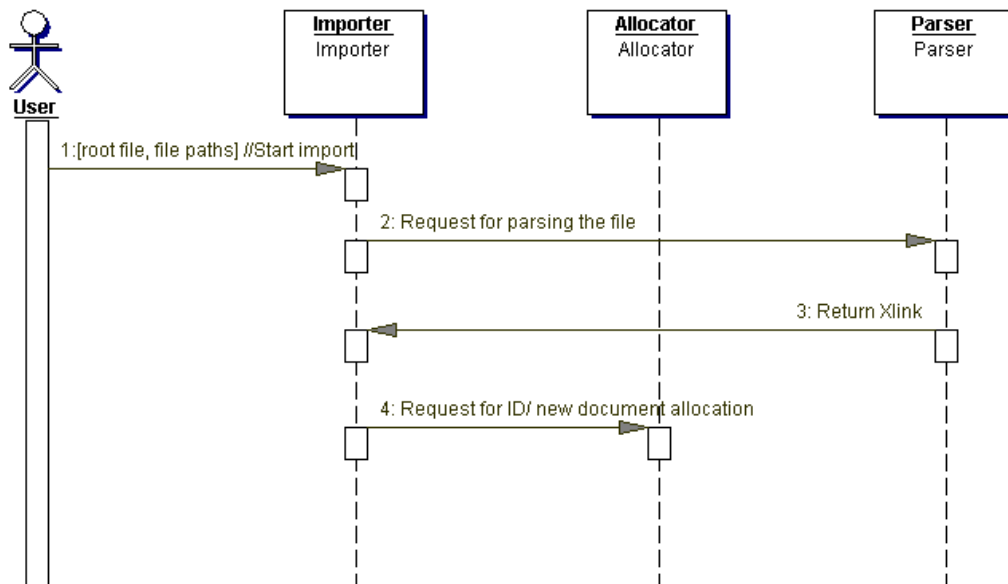


Fig. 4-16 Sequence diagram of the component interaction

Important notes:

- It is important how Importer, Parser and Allocator treat URL. In order to initiate import, Importer receives absolute path of the initial file of hypermedia and transfers it to Parser. The link Parser extracts from a document could be both relative and absolute. All the relative URLs should be resolved in Parser into absolute ones and transferred to Allocator to allocate a document with the absolute URL stored in the URL property.
- The initial file is not always a root file of hypermedia structure (like for example index.htm in HTML). As long as incremental import occurs, it can be any file in this structure. Allocator basing on the URL of initial file can not define correctly file structure to be preserved in CRP. Therefore, the base URL of the original hypermedia should be given as a parameter to Allocator.

The components of the developed specification map the hypermedia structure of arbitrary format into CRP. Their architecture matches the requirements of modularity and their functionality matches the requirements of interruptability. In order to meet the requirements of transactionality the following transactions of importing process are defined:

- allocation of the empty document in CRP
- filling up the document properties: content, Link Set, XLink

5. Design

Design phase formalizes the document specification and the specification of component structure created in the previous chapter in order to develop a framework of the Importer Tool.

5.1. Design of XLink DTD

In subsection 4.2 the structure of document was determined. The properties of the links contained in a document are stored in XLink structure. In order to standardize the usage of XLink for link description in CMS, Document Type Definition for XLink structure is fixed.

XLink recommendation defines the following terminology [XLink recommendation - 27 June 2001].

Local resource is an XML element that participates in a link by virtue of having as its parent, or being itself, a linking element.

Any resource or resource portion that participates in a link by virtue of being addressed with a URI reference is considered a **remote resource**, even if it is in the same XML document as the link, or even inside the same linking element.

Basing on these definitions XLink defines outbound, inbound and third-party arcs.

Outbound – arc which starting resource is local and ending resource is remote.

Inbound – arc which starting resource is remote and ending resource is local.

Third-party – neither starting resource nor ending one is local.

XLink offers two types of links:

Simple links offer syntax for outbound link with exactly two participating resources (the examples could be A and IMG links in HTML)

Extended links offer full XLink functionality, such as inbound and third-party arcs, as well as links that have arbitrary numbers of participating resources. As a result, their structure can be fairly complex, including elements for pointing to remote resources, elements for containing local resources, elements for specifying arc traversal rules, and elements for specifying human-readable resource and arc titles [XLink 1.0, 2001].

As it was defined in the architectural analysis phase, none of the two link endpoints in document, which will be described in XLink structure in IT, belong to it. That means that IT processes third-party links. Therefore, extended type of links is chosen.

A link is described by source endpoint (*from* –element), target endpoint (*to*-element) and arc, giving the information about how to traverse a pair of resources, including the direction of traversal and possibly application-behavior information (*go*-element).

From and *to* elements are of the locator-type, by which XLink indicates remote resources participating in a link. Locator attribute *href* supplies the data that allows an XLink application to find a remote resource (or resource fragment). *Href* attribute of the *from*-element should contain only reference to a location inside the current resource (particularly, anchor ID), since its base URL is the URL of the current resource. *Href*-attribute of the *to*-element is different for XLink structure formed by Parser and those stored in XLink property of the imported document. For Parser *href*-attribute value is an absolute URL of the node. Importer substitute this value with reference to CRP resource indicating the location inside it (*coremedia:CRP Document ID#anch*).

Part of the DTD defined description of *from*- and *to*- elements:

```
<!ELEMENT from EMPTY>
<!ATTLIST from
xlink:type (locator) #FIXED "locator"
xlink:href CDATA #REQUIRED
xlink:label CDATA #FIXED "source">

<!ELEMENT to EMPTY>
<!ATTLIST to
xlink:type (locator) #FIXED "locator"
xlink:href CDATA #REQUIRED
xlink:label CDATA #FIXED "target">
```

Go-element is of the arc type, by which rules for traversing among extended link's participating resources are indicated. Its *from*- and *to*- attributes indicate the direction of the link. *Title* attribute describes the meaning of a link or a resource in a human-readable fashion. *Go*-element contains two behavior attributes *show* and *actuate*. *Show*-attribute describes how target endpoint should be presented on the link traversal. It should contain the following values:

"new" - an application traversing to the ending resource should load it in a new window, frame, pane, or other relevant presentation context.

This is similar to the effect achieved by the following HTML fragment:

```
<A HREF="http://www.example.org" target="_blank">...</A>
```

"replace" - an application traversing to the ending resource should load the resource in the same window, frame, pane, or other relevant presentation context in which the starting resource was loaded.

This is similar to the effect achieved by the following HTML fragment:

```
<A HREF="http://www.example.org" target="_self">...</A>
```

"embed" - an application traversing to the ending resource should load its presentation in place of the presentation of the starting resource.

This is similar to the effect achieved by the following HTML fragment

```
<IMG SRC="http://www.example.org/smiley.gif">
```

"other" - the behavior of an application traversing to the ending resource is unconstrained by XLink specification. The application should look for other markup present in the link to determine the appropriate behavior.

"none" - the behavior of an application traversing to the ending resource is unconstrained by this specification. No other markup is present to help the application determine the appropriate behavior. [XLink 1.0, 2001]

Actuate - attribute describes the time when traversal from starting endpoint to target endpoint should occur. It should contain the following values:

"onLoad" - an application should traverse to the ending resource immediately on loading the starting resource. This is similar to the effect typically achieved by the following HTML fragment, when the user agent is configured to display images:

```
<IMG SRC="http://www.example.org/smiley.gif" >
```

"onRequest" - an application should traverse from the starting resource to the ending resource only on a post-loading event triggered for the purpose of traversal. An example of such an event might be when a user clicks on the presentation of the starting resource, or a software module finishes a countdown that precedes a redirect.

"other" - the behavior of an application traversing to the ending resource is unconstrained by this specification. The application should look for other markup present in the link to determine the appropriate behavior.

"none" - the behavior of an application traversing to the ending resource is unconstrained by this specification. No other markup is present to help the application determine the appropriate behavior. [XLink 1.0, 2001].

DTD of Xlink structure:

```
<?xml version="1.0"?>
<!ELEMENT document (links)>
<!ATTLIST document
id          CDATA          #REQUIRED
url         CDATA          #REQUIRED>
<!ELEMENT links (link*)>
<!ELEMENT link (from, to, go)>
<!ATTLIST link
xmlns:xlink CDATA          #FIXED
           "http://www.w3.org/1999/xlink"
xlink:type  (extended)    #FIXED          "extended">

<!ELEMENT from EMPTY>
<!ATTLIST from
xlink:type  (locator)    #FIXED          "locator"
xlink:href  CDATA        #REQUIRED
xlink:label CDATA        #FIXED          "source">

<!ELEMENT to EMPTY>
<!ATTLIST to
xlink:type  (locator)    #FIXED          "locator"
xlink:href  CDATA        #REQUIRED
xlink:label CDATA        #FIXED          "target">
```

```

<!ELEMENT go EMPTY>
<!ATTLIST go
xlink:type (arc) #FIXED "arc"
xlink:titel CDATA #IMPLIED
xlink:from CDATA #FIXED "source"
xlink:from CDATA #FIXED "target"
xlink:show (new
            |replace
            |embed
            |other
            |none) # IMPLIED
xlink:actuate (onLoad
              |onRequest
              |other
              |none)" #IMPLIED>

```

Following is an example of XLink structure stored in CRP corresponding to the DTD:

```

<document id="myServer/R/1234" url="file:/C:/Test/htm/ NSIDC.htm">
  <links>
    <link xmlns:xlink=http://www.w3.org/1999/xlink
xlink:type="extended">
      <from xlink:type="locator" xlink:href="9971"
xlink:label="source" />
      <to xlink:type="locator"
xlink:href="coremedia:myServer/R/1235" xlink:label="target" />
      <go xlink:type="arc" xlink:from="source" xlink:to="target"
xlink:show="new" xlink:actuate="onRequest" />
    </link>
    <link xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="extended">
      <from xlink:type="locator" xlink:href="10635"
xlink:label="source" />
      <to xlink:type="locator"
xlink:href="coremedia:myServer/R/1237" xlink:label="target" />
      <go xlink:type="arc" xlink:from="source" xlink:to="target"
xlink:actuate="onRequest" />
    </link>
  </links>
</document>

```

The attributes which values are fixed by DTD can be omitted. That leads to the simplified structure:

```

<document id="myServer/R/1234" url="file:/C:/Test/htm/ NSIDC.htm">
  <links>
    <link>
      <from xlink:href="9971" />
      <to xlink:href="coremedia:myServer/R/1235" />
      <go xlink:show="new" xlink:actuate="onRequest" />
    </link>
    <link>
      <from xlink:href="10635"/>
      <to xlink:href="coremedia:myServer/R/1237"/>
      <go xlink:actuate="onRequest" />
    </link>
  </links>
</document>

```

```
</link>
</links>
</document>
```

5.2. Design of the Specifications of the Components.

This subsection specifies algorithms of components identified in subsection 4.3 and their interfaces.

5.2.1. Importer

Importer initiates import by receiving an initial file as a first node of hypermedia structure to be imported. Then, all the nodes should be transferred to Parser to get link information of the node. Importer treats document network as a cyclic graph and uses breadth- first algorithm to traverse it as a tree structure. It keeps a set of the nodes which have to be traversed *Que* and a list of nodes which have been seen *V*.

The list with *to-be-traversed* nodes acts like a queue (FIFO). *Que* and *V* are required in order to avoid visiting a node twice as some of the links can form cycles.

Following is the algorithm of traversing a hypermedia structure:

```
choose start node x
Set V = {x}
List Que = {x}
while  $\exists v \in Que$ 
    choose first node v from Que
    import v
    for each neighbor w
        if  $w \notin V$ 
            add w to the end of Que
            add w to V
    delete v from Que
end
```

Where the *neighbor* is a node to which the current node refers, i.e. current node contains a link to this document. The link information is received from Parser.

The following structure is taken as an example:

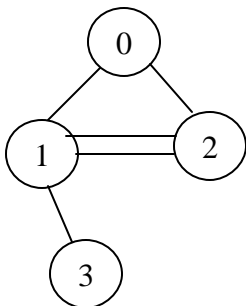


Fig. 5-1 Example of hypermedia graph

Step 0. Initialization. Importer receives the initial file and initiates the lists.

$V=\{0\}$, $Que=\{0\}$

Step 2. Importer imports node 0, adds neighbors of the node 0, node 1 and node 2, into V and Que and deletes 0 from the Que

$V=\{0, 1, 2\}$ $Que=\{1, 2\}$

Step 3. Importer takes the first node in the Que , node 1, imports it and deletes it from Que . Node 3 is added to Que and V . Node 2 is ignored because it is already in V .

$V=\{0, 1, 2, 3\}$ $Que=\{2, 3\}$

Step 4. Node 2 is imported and deleted from Que . Its neighbor, node 1, is already in V .

Step 5. Node 3 is imported and deleted from Que . End of the algorithm.

This example illustrates the continuous import. The incremental import is the special case of continuous one. It should be possible to reconstruct the list, where nodes, which are already in CRP, are marked as visited. In this case algorithm will process only those nodes which were not imported during the continuous import (or previous incremental one). The following terms are used to reconstruct the lists:

- Imported node: the node is imported if CRP contains document which URL property value equals the absolute URL of the node and property Content is not empty.
- Allocated document: the node is allocated if CRP contains document which URL property value equals the absolute URL of the node and property Content is empty.
- Not imported node: the node is not imported if CRP doesn't contain document which URL property value equals the absolute URL of the node.

Reconstructed list of to-be- *traversed* nodes includes only allocated nodes; list of nodes that have been seen includes both imported and allocated nodes.

Following is the general algorithm of Importer for continuous import (see Figure 5-2). For incremental import initialization phase contains reconstruction of the list.

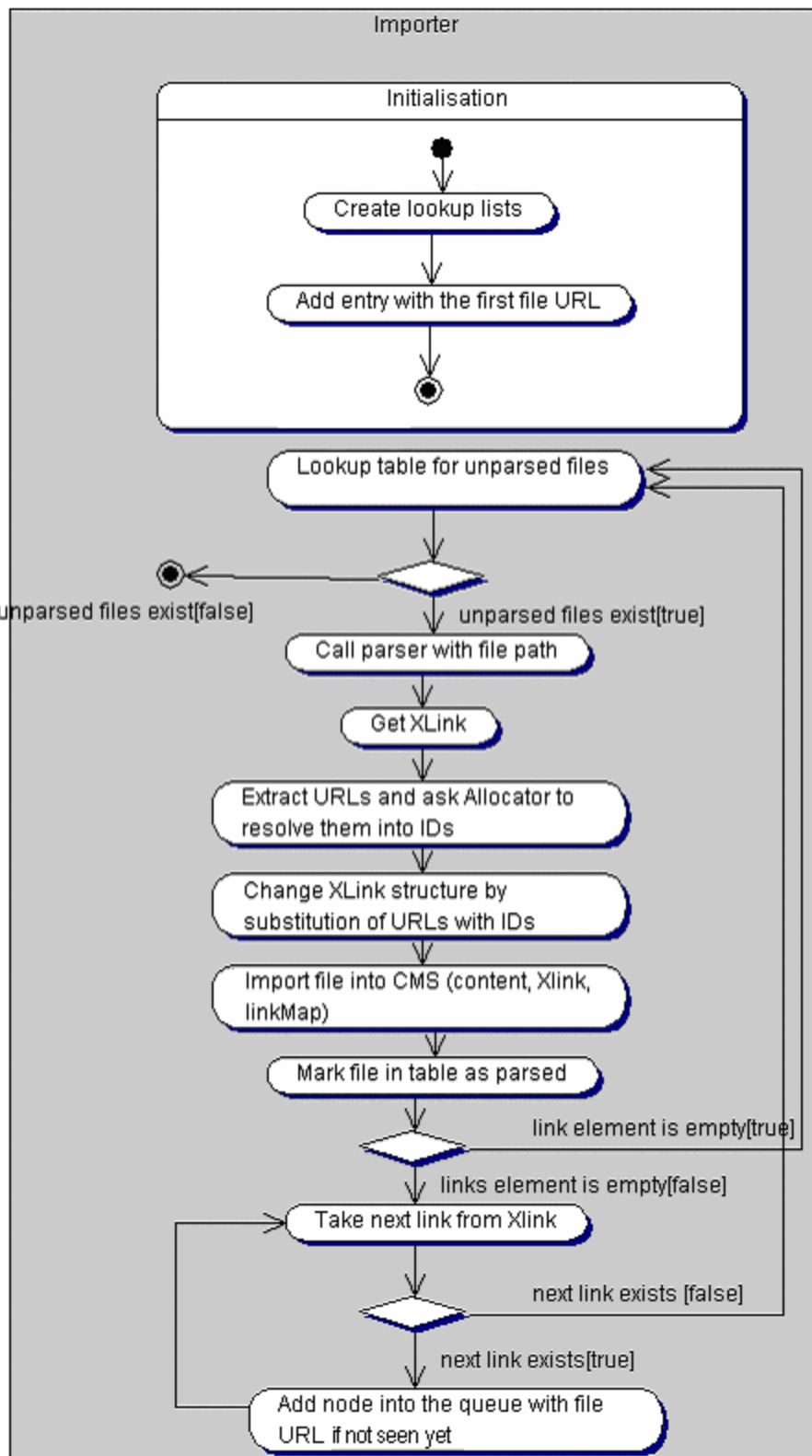


Fig. 5-2 General algorithm of Importer for continuous import

On the step *Get XLink* Importer receives the XLink structure from Parser, from which it extracts URL of the current document and the map of the links containing the entry <#anch, target URL>. Importer asks Allocator to resolve all the URLs into document IDs. It imports the current node to the document with known ID. Next step is to place content of the file into Content property, XLink structure into Xlink property, map of the links into Links property.

The queue consists of the absolute URLs of a nodes.

Importer interface specification:

Constructor: Importer (String workspace, String configuration, String rootDirectory, String rootPath, String user, String password)

workspace – identify the working workspace

configuration – identify the configuration

user/password – login to WebDav

rootPath – URL of the initial file in the structure to be imported

rootDirectory – base URL

Methods: void initImport() – creates resource of default Document and ImportSet types if continuous import is required, otherwise, for incremental import, reconstructs the table of import.

Void importFiles() – imports the hypermedia structure into CRP

5.2.2. Parser

Design of the framework does not include details about how the analysis of certain hypermedia format should be done. This phase concerns only the requirements which Importer imposes on Parser.

Parser receives from Importer the input stream of the file to be parsed and the default base URL, extracts information on links it contains and forms XLink structure to be returned to Importer. For each link, it resolves its URL into absolute one. Fig 5-3 depicts the generic algorithm of Parser.

Operations of Parser require knowledge of the media format. Implementation of its interface should be done for each media format to handle it correctly.

Parser interface specification:

Constructor: HTMLParser (String rootDirectory, String filePath, InputStream in)

rootDirectory – base URL

filePath – URL of the file to be parsed

allocator – instance of Allocator

Methods: Document getLinks() – parses the file and returns XLink document with the description of the extracted links.

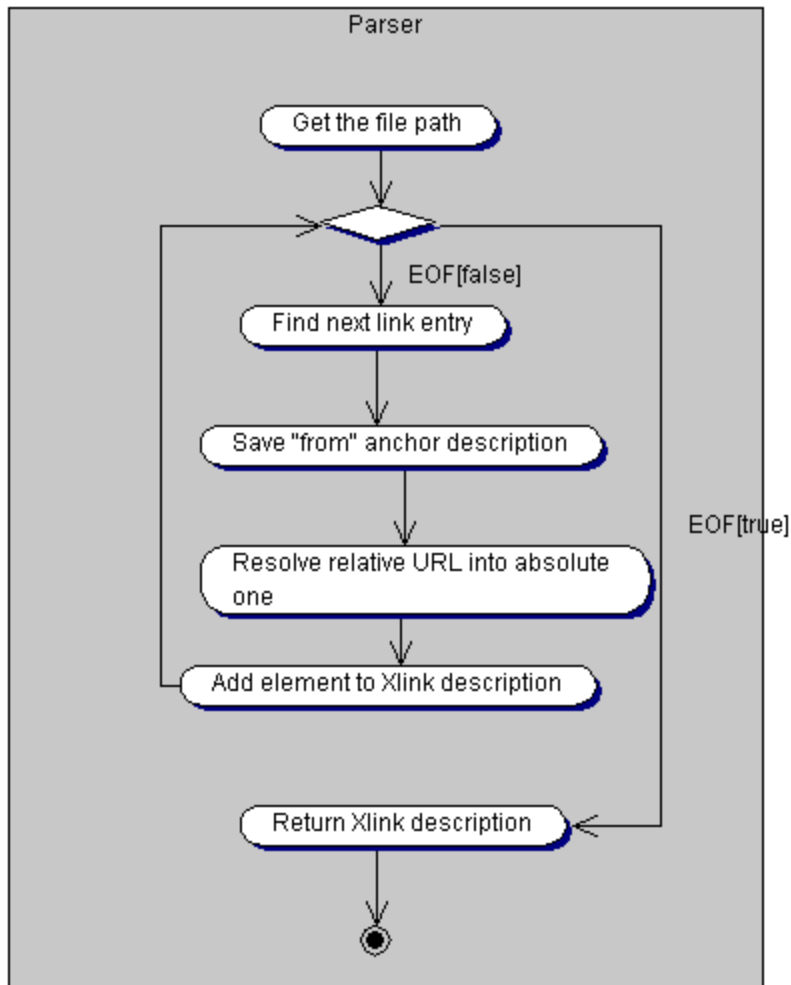


Fig. 5-3 Generic algorithm of Parser

5.2.3. Allocator

Allocator provides referencing services for Importer, mapping URL, kept in URL property, with the corresponding ID of documents in CRP. It gives an URL of a document by its CRP ID for Importer (see Figure 5-4). In case a document with the requested URL doesn't exist, Allocator reserves one (see Figure 5-5). The allocated document contains corresponding URL in URL property and the reference to ImportSet properties. Content, Links, XLink properties are empty.

Preserving the file structure of the original hypermedia in CRP introduces the problem of definition of the folder in which an allocated document should be stored. Allocator

identifies the folder structure by comparing the base URL with the absolute URL of the file. Files, which parent URL (URL without name of the file) equals the base one, are stored in the root directory. For others Allocator creates the folder tree accordingly to the directories, indicated in file's URL. For example, base URL is <http://www.planen-wohnen.de/>. The document, where the file <http://www.planen-wohnen.de/index.html> will be imported, is stored in the root directory, while <http://www.planen-wohnen.de/main/pics/1.gif> is stored under root/main/pics. This defines the scope of hypermedia structure that will be imported. Only those files will be imported whose URL contains the base URL. For other links, those which point to resources situated on other server, the documents will be allocated in the folder Outcome, but never imported. This is done in order to define the borders of imported structure and to provide opportunity to manage the links pointing “outside the scope”.

A folder in CRP is a resource that contains the references to other resources. The root folder identified by WebDAV Configuration.

Allocator interface specification:

Constructor:

Allocator (String workspace, String config, String user, String password)

workspace – current workspace

config – WebDav configuration

user/password – login to WebDav

Methods:

String IDByUrl(String Url) – returns ID by known URL of a file

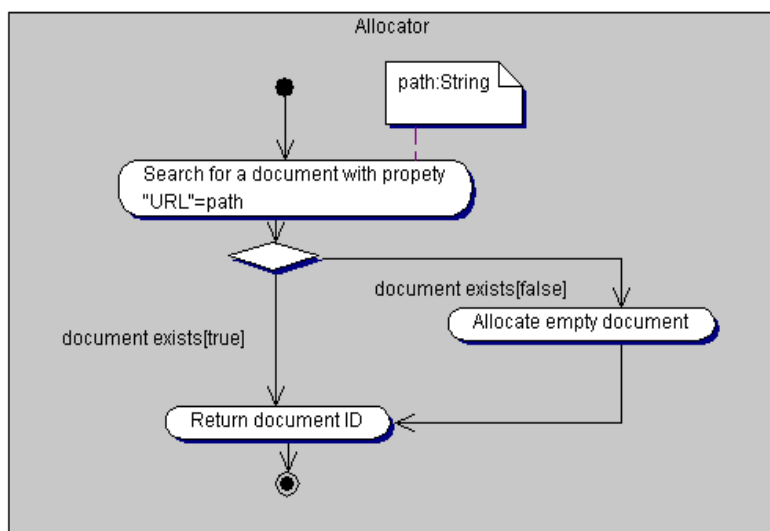


Fig. 5-4 Search of URL algorithm for Allocator

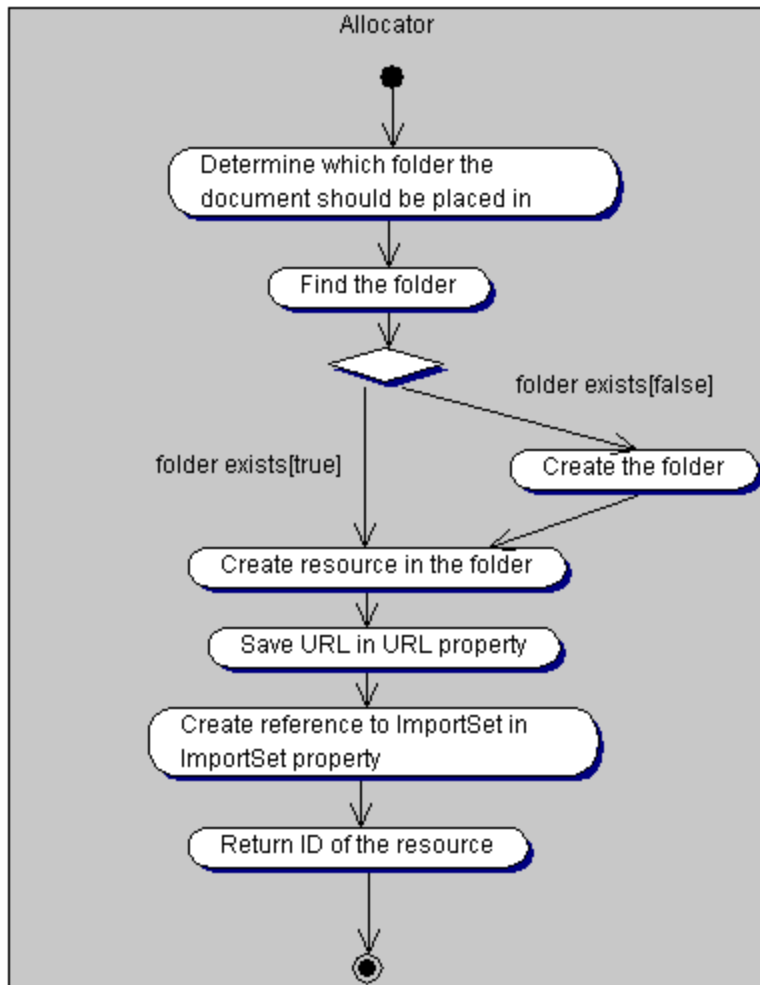


Fig. 5-5 Allocation of a new document algorithm

5.2.4. Class Diagram

Figure 5-6 represents the class diagram of IT:

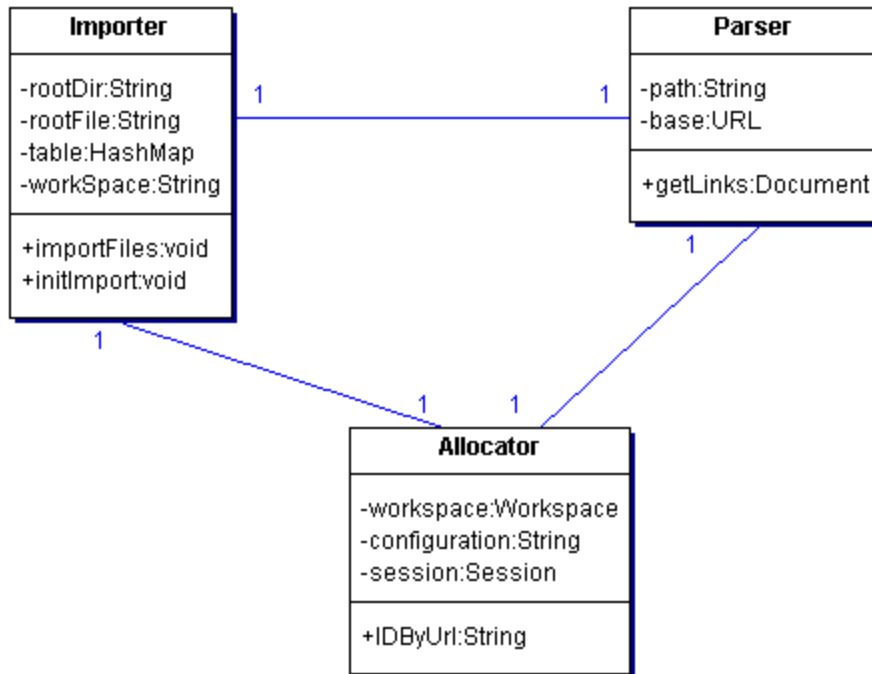


Fig. 5-6 Importer Tool class diagram

6. Prototypical Implementation

The chapter concerns the implementation of the framework designed during the previous development phases. It describes the classes and the methods used by the Importer, Allocator and Parser components.

6.1. Importer

Three classes were used in order to implement the component Importer:

- Importer.
 - Initiates import
 - Traverses hypermedia content structure
 - Calls Parser for each file to parse
 - Parses Xlink structure
 - Calls class ImportTest to import file
- ImportUtil. Utility class. Performs the operations with CRP.
 - Initializes import by creating types and ImportSet
 - Imports files onto CRP
- FilePathParser. Utility class. Processes URLs.
 - Determine name, extension and parent directory from the URL
 - Defines Mime-type of the file
 - Defines directories in which file should be stored in CRP

Method specification	Return value	Description
Class Importer		
Constructor Importer(String worksp,String configuration,String rootDir,String rootPath,String user, String password)		
public void initImport()		For continuous import – initializes import calling ImportTest, creates new lists with initial file entry. For incremental import – restores the lists.
public void importFiles()		Takes nodes in queue, calls Parser for them, imports files and adds entries into the table
private String findUnparsedURL()	Returns URL of the file to be parsed	Takes first element in the queue
private String askAllocator(String URL)	CRP ID of the document	Requests Allocator for ID of the document
private void markParsed(String URL)		Deletes imported node from the queue
private String getIDfromXlink()	Returns CRP ID of the document allocated for the file,	Analyses XLink structure in order to find URL the current node, asks Allocator to resolve it into CRP ID

	to which given XLink structure belongs to	
private HashMap[] getLinkMap()	Returns two lists: Hashmap[0] – list of the IDs, keyed by <i>from</i> endpoints Hashmap[1] – list of the URLs, keyed by <i>from</i> endpoints.	Analyses XLink structure in order to form maps of the links. Map entry format <# from anch, to anch> Changes the XLink structure by substitution of URLs with IDs
private boolean isToParse(String filePath)	True – file is to be given to Parser Otherwise - false	Determines the border of hypermedia content to be imported. In the prototype in cuts off the files which belong to another server then one indicated in base URL
Class ImportUtil		
Constructor ImportTest(String worksp, String config)		Initializes the import
public void init(String baseURL)		Create types and ImportSet
public boolean isInit()	True – incremental import False – continuous import	Defines which type of import should be provided by existence of the ImportSet resource
public void importFile(URLMedia Object mo, String extension, String sVhID, Document doc, HashMap map)		Fills up the properties of the document with corresponding ID
private void createTypes() throws MimeParseException		Creates resource types Default Document and ImportSet with properties (see 4.2. Document Structure)
private Resource getRoot()	Resource of folder type	Defines the root folder in CRP
Class FilePathParser		
Constructor FilePathParser(String path)		
public String getExtension()	Extension of the file	
public String getName()	Local name of the file	
public String getMimeType()	Mime-type of the file	

public String getSubType()	Subtype of mime- type	
public boolean isExists()	True – the file exists False – the file doesn't exist	Check if the link is dangling
public String getFileDirectory()	Parent directory	For example, for http://www.planen-wohnen.de/index.htm returns http://www.planen-wohnen.de
public ArrayList getDirectiries(String baseUrl)	List of the directories in which the file should be stored	For example, for http://www.planen-wohnen.de/main/pics/gif/index.htm with the base URL For example, for http://www.planen-wohnen.de/ the list contains{ main, pics, gif }

Initial file URL should be in the following formats:

file:///disc:/folder/folder/.../filename.extension - for local files

http://server/folder/folder/.../ filename.extension – for remote files

Base URL should be in the following formats:

file:///disc:/folder/folder/.../ - for local files

http://server/folder/folder/.../ – for remote files

6.2. Parser

An HTML parser was created as a prototype. It is based on the HTML Parser provided by javax.swing.text.html.parser package. In order to implement the component Parser three classes were used:

- HTMLParser.
 - Takes a file to be analyzed and creates XLink structure with the link description
- ParserGetter. Utility class.
 - Defines parser to traverse HTML structure
- PageSaver.
 - Analyses HTML document

Method specification	Return value	Description
Class HTMLParser implements <i>Parser</i>		
Constructor HTMLParser(String base, String filepath, java.io.InputStream in)		
public Document getLinks()	XLink structure	Creates XLink structure with the link description
private void linkResolver(String link)		Resolves a relative URL into absolute one
static Reader getReader(String uri) throws	Reader on the HTML data	

IOException		
Class PageSaver extends javax.swing.text.html.parser.HTMLToolkit.ParserCallback		
public void handleStartTag(HTML.Tag tag, MutableAttributeSet attributes, int position)		Overrides: handleStartTag in class HTMLToolkit.ParserCallback This function is called by the parser when it has recognized a start tag
public void handleSimpleTag(HTML.Tag tag, MutableAttributeSet attributes, int position)		Overrides: handleSimpleTag in class HTMLToolkit.ParserCallback This function is called by the parser when it has recognized a start tag
public HashMap getLinkTable()	Map of the links with URLs as values keyed by start point of links in file	Table entry format: <#anch, URL, show, actuate> anch – key value. Position of the link in the file URL – link URL in the form it presented in HTML Show, actuate – behavioral attributes

The table below presents HTML tags which can contain links and should be analyzed:

Tag	Attribute	Type of Tag	Behavioral attributes
A	HREF	wellformed	show: new, if target = _blank show: replace, if target = _self show: other, otherwise actuate: onRequest
AREA	HREF	simple	show: new, if target = _blank show: replace, if target = _self show: other, otherwise actuate: onRequest
LINK	HREF	simple	show: new, if target = _blank show: replace, if target = _self show: other, otherwise actuate: onRequest
FRAME	SCR	simple	show: other actuate: onLoad
INPUT	SCR	simple	show: other actuate: onLoad
BODY	BACKGROUND	wellformed	show: other actuate: onLoad
TABLE	BACKGROUND	simple	show: other actuate: onLoad
TD	BACKGROUND	simple	show: other actuate: onLoad
TH	BACKGROUND	simple	show: other actuate: onLoad
FORM	BACKGROUND	wellformed	show: other

			actuate: onLoad
	ACTION		show: new, if target = _blank show: replace, if target = _self show: other, otherwise actuate: onLoad
IMG	SCR, LOWSCR,USEM AP	simple	show: embed actuate: onLoad
META	CONTENT	simple	show: new actuate: other

6.3. Allocator

Component Allocator contains one class:

- Allocator
 - Identifies correspondence of absolute file URL and its corresponding CRP ID
 - Allocates resource of default document type
 - Reconstructs table for incremental import

Method specification	Return value	Description
Class Allocator		
Constructor Allocator(String currWorkspace, String config, String user, String password)		
public String IDByUrl(String Url)	ID	Searches for the document with the value of property URL equals given one and returns ID of the resource
public String allocate(String baseUrl, String filePath)	ID	Creates folders in which a document should be allocated, allocate document and fills up properties URL and ImportSet
public List[] restoreTable()	List[0] - <i>visited</i> list List[1] - <i>to-be-traversed</i> list	Goes through the resources which refer to ImportSet in order to reconstruct <i>to-be-traversed</i> and <i>visited</i> lists
private String[] findDirectory(ArrayList newList)	String[1] – name of the directory	Determines which directories from those where the file should be stored are already exist. IMPORTANT: There should be no files and directories with the same name

7. Conclusion

7.1. Related work

Media Foundation Beans (MFB) provides support and plug-in mechanism for analyzing and generation of links between non-embedded media-objects and their children, transform such links to child media objects into parent/child relationships between media entity EJBs, and manage these relations in case participants are moved or altered. [EMB 1.0 PD]

MFB disassembles media content into segments in order to describe dependencies between media files and creates a dependency graph. The EBNF grammar illustrates the relation between media objects and media segments:

```
Media: = {MediaSegment}  
MediaSegment: = content [childLocation]
```

Unlike in the developed IT, link information is stored together with the content in media segments. That imposes restriction on link modification and limits disassembly to non-embedded media-objects.

During regenerating the original media content, MFB assembles segments into one using the link information inside the content. IT supposes to have two flows of data for regenerator tool: content and link information.

7.2. Summary and Possible Further Development

The development of the Importer Tool framework was performed in several phases. During the Analysis phase, the subject domain was investigated. That required the analysis of different hypermedia formats in order to create a model of their structure. A formalized approach to hypermedia content was made, assuming that each hypermedia structure can be considered as a network of nodes with interrelated links, but the granularity, or elements which can be seen as a node, depends on the type of the structure. Another assumption was made concerning the definition of link endpoints: for each hypermedia format it is possible to determine the location specifier (anchor ID) of a departure endpoint. This is not always the case, for example, for Macromedia Director presentation. Therefore three levels of “service” for imported documents were suggested. Depending on to which extent these assumptions are applied to the certain media format, further CRP services for this structure could be:

- restricted to the query inside CRP
- restricted modification of the documents (no add and remove of the links)
- not restricted

The object structure of CRP, the CMS into which a document structure will be imported, was investigated. This provided the basis to formulate requirements for the Importer Tool

which are imposed by the difference between CRP meta-model and the model of hypermedia structure.

Architectural analysis allowed formulating document structure and component structure specifications. In order to separate hypermedia format-dependent and independent operations, two components were specified:

- Importer: responsible for traversing hypermedia structure and import nodes into CRP
- Parser: responsible for analysis of node structure in order to extract linking information

The third component, Allocator, is a service module for Importer and Parser. This component structure makes it possible to implement Parser module for different media and plug-in into the Importer Tool.

During the Design phase these specifications were formalized in order to develop a framework of the Importer Tool. That included

- development of DTD of XLink structure
- specification of functions and interfaces of each component
- development of working algorithms of each component
- specification of the communication between components

The Importer Tool framework covers only the issue of how to preserve the link structure while importing hypermedia content into a CMS. Further work should cover the modification of the link structure inside CRP based on the developed document structure and the use cases, identified in this work, for example editing, moving, deletion of a document, modification of links, publishing of the imported document structure. A user should be able to change the type of a document, modify the content of documents, add or delete links etc. It will require the development of a link analyzer to check link consistency, and the development of the re-importer which can interpret the changes in the document content.

Another subject to research is re-generation, or the export, of the hypermedia content. An export module should assemble the structure on the new location taking into account the properties of links.

Bibliography:

[Son, J.-B., 1998] Son, J.-B. *Understanding hypertext: A discussion for TEFL*. English Teaching, 53 (3),1998, 113-124.

[Conklin, 1987] Conklin, Jeff. *Hypertext: An Introduction and Survey*, IEEE Computer, September 1987

[Slatin, 1990] Slatin, John M. *Reading Hypertext: Order and Coherence in a New Medium*. College English 52(1990): 870-83.

[Nielsen,1990] Nielsen, Jakob. *Converting Existing Text to Hypertext*, Chapter 11, Hypertext and Hypermedia, Academic Press, 1990

[Seyer, 1992] Philip Seyer, *Understanding Hypertext Concepts and Applications* (Blue Ridge Summit, PA: Windcrest Books, 1991)

[SMIL 2.0, 2001] W3C Recommendation for Synchronized Multimedia Integration Language (SMIL 2.0), 2001

[Rutledge, van, 1999] Rutledge, van, *Anticipating SMIL 2.0: The Developing Cooperative*, 1999

[OHP, 1996] Hugh Davis, Andy Lewis, and Antoine Rizk. *OHP: A Draft Proposal for a Standard Open Hypermedia Protocol* (Levels 0 and 1: Revision 1.2 - 13 March 1996).

[Grabinger et al., 1992a] Grabinger, R. S., Dunlap, J. C., and Jonassen, D. H. (1992a). *Hypertext links. Performance and Instruction*, pages 36-49

[Grabinger et al., 1992b] Grabinger, R. S., Dunlap, J. C., and Jonassen, D. H. (1992b). *Sequential links. Performance and Instruction*, pages 46-49.

[Grabinger et al., 1993a] Grabinger, R. S., Dunlap, J. C., and Jonassen, D. H. (1993a). *Relational links. Performance and Instruction*, pages 35-40

[Grabinger et al., 1993b] Grabinger, R. S., Dunlap, J. C., and Jonassen, D. H. (1993b). *Support links. Performance and Instruction*, pages 33-39.

[Landow, 1989] Landow, G. P. (1989). *The rhetoric of hypermedia: Some rules for authors*. Journal of Computing in Higher Education, 1(1):39-64

[XLink 1.0, 2001] W3C Recommendation for XML Linking Language (XLink) Version 1.0, 27 June 2001

[Wienberg A., 2002] Wienberg, A., Ernst, M., Gawecki, A., Kummer, O., Wienberg, F., Schmidt, J.W., *Content Schema Evolution in the CoreMedia Content Application Platform CAP 4* in: C.S. Jensen et al. (Eds.):EDBT 2002, LNCS 2287, Springer-Verlag, Berlin, Heidelberg, p. 712-721

[EMB 1.0 PD] Enterprise Media Beans Specification, v.1.0, Public Draft (JSR 086), 2002

[Halasz 1990] Halasz, F.G., Schwartz M., *The Dexter Hypertext Reference Model*, NIST Hypertext Standardization Workshop, Gaithersburg, MD, January 16-18, 1990

[Gronbak 1996] Gronbak, K., Trigg, R.H, *Toward a Dexter-based Model for Open Hypermedia: Unifying Embedded References and Link Objects*, Proceedings of Hypertext'96, Washington, D.C., March 16-20, 1996