
Citation for published version (APA):

Document status and date:
Published: 01/01/2004

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at: openaccess@tue.nl providing details and we will investigate your claim.

Download date: 24. Jun. 2019
AH 2004: 3rd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems

Industry Session

Editors and Co-Chairs:
Mehmet Göker
Geert-Jan Houben

ISSN 0926-4515

All rights reserved
Series editors: prof.dr. P.M.E. De Bra
               prof.dr.ir. J.J. van Wijk

Reports are available at:
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Author&Level=1 and
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Year&Level=1

Computer Science Reports 04/21
Eindhoven, 23rd August 2004
## Table of Content

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADIRA: ADaptive Information Retrieval Application</td>
<td>1</td>
</tr>
<tr>
<td><em>Koen Aben, Richard Beens, Teun Hoogendoorn</em></td>
<td></td>
</tr>
<tr>
<td>Programming adaptivity of course material by use of an action module</td>
<td>10</td>
</tr>
<tr>
<td><em>Egbert Heuvelman, Pjotr Prins, Sander Steenbergen</em></td>
<td></td>
</tr>
<tr>
<td>Ontology-Based 3D Manager System</td>
<td>18</td>
</tr>
<tr>
<td><em>Christophe Nicolle, Christophe Cruz, Renaud Vanlanede</em></td>
<td></td>
</tr>
</tbody>
</table>
ADIRA: ADaptive Information Retrieval Application

Koen Aben, Richard Beens, Teun Hoogendoorn
Advanced Team Systems Center (ATSC),
Industrieweg 69b, 5145 PD Waalwijk, Netherlands
{ka, rb, th}@atsc.nl

Abstract. This article presents the project ADIRA. The project covers cooperating applications that are able to provide an organization with a web information system generated from multiple heterogeneous data sources. With this article, we demonstrate the current status of our work. The first section introduces the project and its application domain. The second section describes the system functionality. The third section explains the overall architecture and its main operations. In the final section we give our conclusions with the guidelines of our future research.

1 Introduction

An accountant has requested our company to develop a particular application. The accountant keeps track of collecting, organizing and controlling financial information. The information can be used to advice clients in a variety of financial subjects. The accountant would like to assist clients with an application that retrieves financial information. This system should give users an efficient and effective access to documents of their interests. These documents are retrieved from a number of organizations, which frequently update their collected works. The presentation should be automated, knowing that the information should be organized cleverly.

This business problem has started our development project on Information Retrieval. At that time, the TU/e already had achieved progressive research on Adaptive Hypermedia [4] and Information Retrieval [13]. In cooperation we developed an architecture called Adaptive Information Retrieval Application (ADIRA) [2]. This architecture can retrieve an adjustable interface for structured data sources. In practice electronic documents are rarely sorted out, whereas the structuring of data involves a lot of manual labor. Another system has consequently been invented, that generates structures according to built-in document analyses. This second system is called Automatic Data/Information Structuring Application (ADISA).
2 The system overview

We have developed an architecture that combines fundamentals for information retrieval, called ADIRA. Our approach has been to construct a system where distributed documents are retrieved as adaptive hypermedia. The adaptive hypermedia provides a user-friendly navigation through the documents. The generated presentations require having useful relationships between available documents available. Our second system, entitled as ADISA, can compute such obtainable data structure. This system structures by configurable algorithms on the document collection, indexed with dictionary assessments. The implemented systems are accessible through a general internet browser, being provided by an everyday web server.

2.1 ADIRA: Adaptive Information Retrieval

The system ADIRA provides a personalized interface for navigating in a hierarchical document structure. The documents are retrieved from distributed, heterogeneous sources. The information is presented in a hierarchy, based on selected subjects, which a user can select from a preference profile. This profile is able to adapt upon user needs, while being updated through user navigation. The quality of available documents is calculated with the collaborative quality-filtering algorithm [12]. This algorithm calculates the quality of an article by shared analyses, which is dependant on the reliability of the user. The architecture provides adaptive hypermedia functionality defined in rules, ranging from information retrieval to e-learning.

ADIRA generates adaptive hypermedia from the retrieved information. The adaptive hypermedia is divisible into two main categories, namely adaptive presentation and adaptive navigation [8]. ADIRA can insert fragments based upon adjustable rules and present links directly related to user navigation. The system employs the coloring and sorting of links as main techniques for adaptation. These concepts are obtained by authored rules, individual algorithms that can specify a task, like a distinct algorithm to sort links on relevance. The initial preferences for this algorithm can be indicated in the user profile. Also, the system observes navigation and lets the user value the read documents. A screenshot of ADIRA is displayed in Fig. 1.

![ADIRA](image)

**Fig. 1. Personalized information retrieval**
2.2 ADISA: Automatic Information / Document Structuring

As mentioned before, the main system needs a metadata structure to generate the hypertext. ADISA can generate this structure in three phases. The first phase concerns the indexing of documents from various sources. The contents of these documents are turned into comparable vectors in the second phase. Finally, in the third phase the documents are structured for the information retrieval. ADISA offers a rich collection of adjustable methods for document calculations and the structuring of information. However, in general the contrasting requests an unacceptable calculation effort; because of the so-called 'curse of dimensionality' [3] and therefore the dimensions of the vectors need to be kept minimized. This section contains a short summation of the implemented methods for indexing and structuring.

The current implementation uses various common indexing methods [1, 7, 11]. We will outline the techniques stemming, tagging, compound splitting, statistical term selection and simplification by dictionaries. The technique stemming transforms various forms of a word towards the same term; the expressions are automatically rewritten by using a dictionary or a conversion algorithm. The technique compound splitting divides combinations into sub words for a more detailed structure. A third technique is tagging that labels sentence parts with a regarding type, verb or subject, for an approximation of the most important words. Statistical term selection, another indexing algorithm, filters with statistics by selecting the dimension values above certain adjustable boundaries. The last technique filters predefined words using dictionaries, as an advanced form of stemming. The implemented potentials to filter are fillers, synonyms and concepts. The filler is a word that does not give any relevant information, but is often frequently occurring, that slows down the algorithm and therefore should be removed to reduce the amount of terms. Synonyms have the same meaning can be translatable towards the same term. Lastly, concepts are composed associations of words translatable to a useful term in a predefined context. After execution, these algorithms have generated an effective scope of the document vectors.

The indexed documents are structured by means of classification and clustering. Classification operates an initial training set to structure, while clustering divides the documents into groups without an influencing training function. ADISA has two methods for classification, labeled as Bayes and TFIDF [10]. The method Bayes classifies using statistics; the values of likelihood are calculated between documents to determine whether a document belongs to a certain class. The likelihood to contain this document is calculated for every class, for all documents to classify. The document is structured afterwards in classes with the highest probability. The second method for classification is TFIDF, a shortage for Term Frequency / Inverse Document Frequency. TFIDF relates the occurring words in a document with the number of documents they appear. The system contains three methods for clustering, respectively K-Means [6], HAC and GHSOM [9]. K-means clusters the documents are grouped in a predefined number of clusters, based on the means in a group. HAC, the shortage for Hierarchical Agglomerative Clustering, unifies vectors or sub clusters containing vectors until all documents are covered into one hierarchical collection. GHSOM, the
algorithm Growing Hierarchical Self-Organizing Map, maps the document vectors as a training collection. To train the vectors, the nodes with a prime similarity are updated recursively.

3 Architecture

This chapter describes the architecture of the two systems. Interfaces allow that these systems can operate independently of each other with shared databases. The applications are positioned between interfaces with the database management system (DBMS) and the user component. The application interfaces are designed to allow the systems being used for various output media. The user component is extensible, but currently ADIRA processes no more than the original browser perspectives and a recently added output for emailing. The interfaces that connect the DBMS ensure that several kinds of databases can be used. This provides possibilities to choose among databases on cost and performance. These kinds of databases are nowadays restricted to MySQL, Postgress and Lotus Notes.

![Fig. 2. Overview of the two systems](image)

3.1 ADIRA

The architecture is designed according to AHAM [5], a reference model for adaptive hypermedia. Three layers divide the application into particular main components (Fig. 3). These are the Runtime layer, the Storage layer and the Within-component layer. The Runtime layer transmits the user interaction. The Storage layer performs the storage and processing of data for the system adaptation. The articles are retrieved
through the Within-component layer. The database interface delivers the data for the layers, mostly the stored documents and models with interpretation rules.

The system uses these models to establish the adaptive hypermedia for both interpretation and construction. The entities are specified in an adaptation-, domain- and user model. These models, based upon AHAM like the main components, together define the techniques for adaptive hypermedia. The rules in the adaptation model specify how the adaptive engine should combine the documents and the user. The domain model is the abstract representation of the information. The user model represents the specified user needs of concepts in the domain model. Finally, the system can deliver structured data with advanced relations to user needs.

The Request Handler processes incoming requests to the Storage layer for interpretation and receives back output in XML. The component XML2HTML transfers this output to HTML on behalf of the application. The Data Repository delivers the needed access to the DBMS for these models and the distributed documents. The AH Handler provides access to the three specific models. The documents can be retrieved by interaction with the Within-component layer. After a page request, the Rule Handler performs rules to observe the needs and construction of the presentation specifications. The output is posted to the Request Handler as specified in concepts. The interpretation of concepts is defined in a Concept Handler for a variety of databases. The components of class Wrap2XML transfer this output into XML.

![Fig. 3. The ADIRA architecture](image)

### 3.2 ADISA

A structured set is constructed in the sequential order indexing, pre-processing and structuring. The ADISA architecture holds three components with corresponding names to structure the sets of documents (Fig. 4). The design of these components is described and illustrated in the following subsections.
3.2.1 The Index layer

The index layer generates the initial properties of the documents received from distributed sources. The Document Handler organizes the entire indexing. Initially, the locations of the available documents are received, granted through the ADISA interface. The handler can wrap a diverse set of input data retrieved by their locations. The document data is filtered through the class TextFilter and afterwards forwarded to the indexer. The indexer stores the received properties of an article in the DBMS.

The Document Handler retrieves information from heterogeneous sources as shown in Fig. 5. A wrapper prepares available documents for internal usage, by reading formats from databases and files. Database wrappers deliver their contents from databases and file wrappers return retrieved data from external files. The actual reading of files occurs in a FileReader, vacant for a number of document formats such as PDF, RTF, and DOC. The indexing layer can be extended for further file formats.
3.2.2 The pre-processing layer

The pre-processing phase ensures the construction of vectors from indexed documents, as explained in section 2.2. The Document Vector Handler coordinates a vector construction in a number of handlings (see Fig. 6). This main component determines the word terms initially, required for the vectors and subsequently calculates the values for these terms. The handlings are dividable into actions for preparation and execution.

There are four classes that implement the preparation actions. The Stemmer class can be equipped with a number of stemming methods, currently a dictionary algorithm and two versions of the Porter algorithm. The Compound class contains methods for compound splitting. Compounds are divided making use of a stored dictionary and the contents from previously investigated documents. The class Tagger contains one tagging method to derive statistics from predefined tags. The class TFIDF has methods to calculate statistics for the structuring phase.

The construction of a document vector occurs in two execution actions. The user can determine the terms to use as a vector dimension, which the class TermSelector can select under a number of restrictions. The Document Vector Calculator (DVC) computes the document vectors subsequently. The dimensions are derived from the vectors and the selected terms. The documents are traced for every dimension to fill in the vector values.

![Diagram](Fig. 6. The pre-processing layer)

3.2.3 The structuring layer

The structuring phase produces a structure due to the automatic construction and the manual changes to existing structures. The Structuring Handler generates the original structure with the help of three classes. The classes Clusterer and Classifier contain the algorithms to execute the structuring. The class Training constructs the labels for the registration of document vectors. For the period of classification all documents
have to be registered, because for training each document vector needs to be labeled. The class Structure Handler ensures that adaptations can be made on an accessible structure to derive a new version. The adjustable structure, displayed by the Visualizer class, can be influenced through the Adapter class.

![Diagram](image)

**Fig. 7. The Structuring layer**

4 Conclusions and future work

This research provides methods for the construction of personalized web systems, built with documents from heterogeneous data sources. The resulting architecture can generate an advanced structure from a chaotic pool of documents and embed retrieved information as an adaptive hypermedia system. The approach offers a rich collection of adjustable methods to structure the information. The diversity of structuring opportunities enables to select an optimum in speed or quality.

The approach shows benefits for both sides of our business solution. An adaptive hypermedia system permits a fine usage of data for information retrieval. The implemented methods to instruct the adaptive hypermedia system ensure models that are very adaptable. Second, dictionary algorithms have proved to be an excellent basis for the generation of a document structure. The project contains many existing algorithms that bring together much functionality, offering a powerful combination.

This article reports the project status prior to full deployment, being much more than a pilot application that just proves our concepts. The application can perform innovative capabilities in an industrial setting, such as the accountant corporation of our client. Although the current implementation gives a sufficient performance, the accountant would like to obtain more basic authoring conventions. We still need to provide support for authoring the document structure and rules for adaptive hypermedia.

In the nearby future we aim to investigate extensions such as dynamic linking and advanced user modeling. The system adaptation can improve when links are inserted directly related to a category of users or documents. Still, being no more important than performance, the additions for executing these links should also guarantee a rea-
sonable response time. Finally, the extended possibilities to indicate user needs ensure categorizing user adaptation depending upon a wide-ranging context.

References


Programming adaptivity of course material by use of an action module.

Egbert Heuvelman, Pjotr Prins, Sander Steenbergen

Turpin Vision, Eindhoven, Netherlands
{e.heuvelman, s.steenbergen}@tv.nl, pjotr.prins@wur.nl
http://www.tv.nl

Abstract. Demand from educational institutions for adaptivity of educational content has resulted in the development of an adaptive player as part of the Content-e tool set. The tool provides authors the possibility to examine student knowledge, alter presentation based on those results, and redirect the student to other content available within the player. Meanwhile the player allows students to filter on the presented content itself. The challenge was set to extend the existing authoring- and Learning Content Management System (LCMS) that facilitates authors to program these adaptations. In this case a solution was chosen that lets authors program an action module with an innovative XML editor. An action module can be placed anywhere within the framework of existing content. The action modules can be tested interactively within the authoring environment.

1 Background

Development of courses is traditionally based on a combination of lectures, written material and practical exercises. Introducing self-study courses has always been a challenge. With the advance of electronic media on student desktops the research for usable intelligent educational systems has become more opportune. Also recent standardization of the data transfer of course material between educational systems like Learning Content Management Systems (LCMS) and Learning Management Systems (LMS), and improved knowledge on effective didactical use of digital educational material sets the challenge to provide tools allowing a group of authors to create adaptive educational content.

Turpin Vision develops the software for content players, enriches textual content and supplies authoring tools for training- and educational organizations. In 1998 the Content-e authoring environment was designed and developed. This browser based tool has various publication formats like Word, PDF and interactive formats for cd-rom and the Internet. Since the original inception Content-e development focus shifted from collaborative content creation to the reuse of content and the adding of metadata for SCORM [3] publications. At this point in time the market asks for adaptive functionalities within a player. This article describes the use of such a player and the accompanying instruments.
2 Adaptation

In conventional tools the student navigates by menus or text search to pages. Within those pages hyperlinks refer to other pages within the same menu structure. Apart from search and navigation the presentation of information is essentially the same as in conventional books.

Recently some new adaptive concepts were implemented by software that focus more closely on didactical concepts.

**Adaptive Filters:** A relative simple example is the option for students to filter the content that is shown in the player according to preferred properties as assigned by the authors. For example, a student can filter all except the learning objectives and exercises of the content student and this way he can focus on self-assessment.

**Adaptive Content:** More advanced adaptive functionality is based on results of exercises that are placed directly in the content and performed by the student. Depending on the results of a set of questions the presentation of menus and content can be altered by adding style sheets to the chosen content. Also redirects to other content can be executed.

3 Authoring environment

To introduce adaptive players with the desired content one needs a specialized and sophisticated content management system. This system facilitates various roles within organizations during content creation- and maintenance processes. The editor, for example, creates templates for the content and assigns them to authors. The authors write and place content within those templates. This way of working is noted as traditional editing [1]. The application managers take care of the overall functionality of the system. Within the Content-e authoringsystem and LCMS those and other functionalities are build in. In this chapter the action module is discussed as being the programmable unit of the authoring environment to create adaptivity. First the content storage architecture of Content-e is discussed.

Content is stored per paragraph in separate records in a central database. A document consists of a tree of paragraphs. A Book consists of a tree of documents. Books can be put on desks. A desk is the place where a student follows the course. Thus content is organized by a hierarchical tree of paragraphs and menu objects (desks, books, documents) as presented in Fig. 1. Menu objects are thus navigation tools or organizing principles, while paragraphs contain the content.
Paragraphs, documents, books and desks can be reused in other branches of the tree. Also these objects can contain their own set of properties. Values are assigned to a property for many purposes. Values can represent certain metadata types like defined in SCORM and IMS [2] standards. In this article properties serve as style sheet magnets for the action module.

A paragraph can contain HTML or XML data. HTML data typically contains textual data. XML data typically represents interactive actions like exercises. Both types can contain images, hyperlinks to paragraphs in other documents, and non-HTML content like Flash animations and MS Office documents. Both HTML and XML paragraphs can be combined within one document and for both types Content-e provides a suitable editor.

HTML data can be edited raw or with the built-in WYSIWYG editor. In practice few authors deal with raw HTML these days.

A set of HTML paragraphs within a document is rendered within the browser as a single HTML document. XML paragraphs are rendered within a frame of the document by the corresponding style sheet. XML paragraphs are edited by the XML editor of Content-e according to the rules of the corresponding XML scheme. This XML editor does not expose the author to the factual XML syntax — it really allows an author to deal with information in a sensible way hiding the intricacies of XML. A set of questions is visible as a tree of questions where certain properties (like answers, and the way of answering) are properties and can be modified as such.

The use of the module by a student does not show the XML — you would not know it is the underlying grammar — other than in the functional display of, for example, a set of questions that need to be answered.
Within this architecture three aspects are used to create an adaptive player within the authoring environment:

- The XML exercises
- The XML action module
- Properties assigned to paragraphs and menu objects.

The first two items are discussed below, while properties are discussed within the action module.

**Exercises**: Within Content-e different types of interactive exercises are possible. An exercise is stored as XML and rendered by the corresponding cascading style sheet (CSS). An exercise itself can contain hyperlinks to any part of text. Thus by adding a hyperlink in a response, a user of the adaptive player, the student, can navigate to certain parts of the content. The limitation of this approach is that on basis of a single exercise the adaptive conclusions have to be drawn while often a set of questions would be more appropriate.

**Action module**: To tackle some of the limitations of an adaptive single exercise the action module is applied. This module is an XML paragraph that normally is added after a set of exercises where the author wants to assign functionality. In the player an XML paragraph only shows a confirm button while underwater it contains a set of functionalities that are assigned by the author with the XML editor. Those functionalities are:

- **Assigning exercises to the action module.** Each assigned exercise will get a score that will be added to the total score of the action module.
- **Creating ranges of scores.** To tackle the limitations of just “Wrong” and “Correct” the action module allows authors to create more score ranges. For example the set of three: “poor”, “moderate”, “good” as visualized in Fig. 2.
- **Assigning actions to each range of score.** Those actions are performed by the player after a match with a range of score.

<table>
<thead>
<tr>
<th>scorerange 1 (poor)</th>
<th>scorerange 2 (moderate)</th>
<th>scorerange 3 (good)</th>
</tr>
</thead>
<tbody>
<tr>
<td>score:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

**Fig. 2. Example of score ranges**

The number of actions within a range of scores is various. Among those are:

- **Feedback alert**: A user is informed about the result and has to confirm the alert to continue his work within the player, or to allow other actions to be performed.
- **Feedback in action module paragraph**: Without any obligations a user is informed about the result. In this response hyperlinks to other paragraphs or documents can be added. The user is free to follow these links. This is the same type of action that can be performed by a single question without the action module.
- **Automatically redirected** to another paragraph or document: This action can best be placed after a feedback alert with appropriate text to avoid confusion for the student.
- **Assignment of style sheets to properties**: content that fits the property and a condition like “level=poor” will be rendered according the assigned style sheet
(Fig. 3) Those style sheets can show or hide content, show text in grey, or use a larger font. Thanks to the hierarchical structure of paragraphs and menu objects deeper level content is rendered according the higher level assigned style sheets. Within one range of scores more properties can be programmed by assigning style sheets to make more advanced adaptivity.

- **Calling a hyperlink with a variable:** The action module can call external programs and variables can be passed to the calling program. This can be used to inform a LMS server about the score of progress of the student, or to call a web service for added functionality to the course.

\[
\begin{align*}
\text{Condition for score range 1:} \\
& \text{If subject = A then stylesheet = italic} \\
& \text{If level = basic then stylesheet = visible} \\
& \text{If level = Advanced then stylesheet = invisible}
\end{align*}
\]

before execution of the action module:

\[
\begin{align*}
& \text{Paragraph 1 Properties: Subject = A} \\
& \text{Paragraph 1.1 Properties: Level = basic} \\
& \text{Paragraph 1.1' Properties: Level = advanced} \\
& \text{Paragraph 2 Properties: Subject = B}
\end{align*}
\]

after execution of the action module:

\[
\begin{align*}
& \text{Paragraph 1 Properties: Subject = A} \\
& \text{Paragraph 1.1 Properties: Level = basic} \\
& \text{Paragraph 1.1' Properties: Level = advanced} \\
& \text{Paragraph 2 Properties: Subject = B}
\end{align*}
\]

**Fig. 3.** Applying combinations of style sheets to the hierarchical structure of paragraphs. The content is shown from the student’s perspective.

Combinations of actions are allowed if not advised. Each action is executed after the other. This gives an author freedom to program his content extensively like discussed in the next chapter.
4 Examples of adaptivity programmed by the action module

Within the customer base of Content-e the next adaptivity needs of the players are met by the action module:

**Accessibility modus.** On the first page of a publication the question whether or not one need to have content rendered according to a certain level of the W3C Accessibility Guidelines [4]. Two types of actions can be performed. Firstly a text is rendered in a larger than standard font. Secondly certain type of content is replaced by a disabled-friendly version. For instance a deaf person will get the subtitled version of a video – if available.

**Level adjustment:** Within a course a student get a test that determines his level about a certain subject. If his knowledge is poor the content with the property-value combination “level=poor” is shown. If his knowledge is good the content with the property-value combination “level=good” is shown. Content without those properties is always shown. For example a student will get additional learning support. Another application can be additional content such as prior knowledge concepts.

**Workflow:** Within a course a student get a test that determines his knowledge about a certain subject. If the student fails this test he is redirected to basic course material about the subject to improve his knowledge. If the student passes this test the remaining part of the course is made visible and he can continue the course. By combining more action modules with this principle one gets a workflow.

Apart from the above examples a wider range of programmed adaptivity is possible.

5 Implementation

For both the adaptive player and the corresponding authoring environment there were technical and human interface considerations when designing and developing the products.

For the student the adaptive player does not look different from a normal player. Documents can be read the conventional way and within this content questions are found. A student answers the questions and confirms its choice. At this point it is the author who virtually leads the student through the course by supplying a meaningful response. For example, if the author chooses a feedback alert in combination with a redirect, this feedback should:

- Present the score of the test;
- Motivate why this score results in a redirection to another part of the course;
- Show what the student is expected to do in the other part of the course.

This way a student understands what is happening and does not get lost within the course (Fig. 4).

15
The authoring environment closely models the user environment. Text or exercises can be added as well as action modules. The XML editor is opened and an intuitive fill-in-form can be programmed to perform the desired actions (Fig. 5). A strong part of the implementation is that the author can test the result instantly within the same authoring environment. This way an author can directly feel how the applied adaptive logic works for the audience. It stimulates experimenting with content and adaptivity.

Authors internal or external to educational institutions often have limited time and scope for mastering authoring tools. Also the frequency of use of those tools can be low and therefore one should be able to easily catch up with work after some time and
the tools should not get in the way to achieve this two concepts are applied: The use of an intuitive and easy to use action module user interface and the addition of interactive animations showing the authoring process.

6 Discussion

At time of writing the action module allows for the selection of exercises, the assignment of score ranges and most actions are operational and applied by authors within recent publications.

These authors using the authoring environment prove to need little time to apply the action module adding to their existing content. The concept of the action module appears to be easy to learn. The possibility of testing the programmed functionality of the action module within the authoring environment is experienced as being author friendly.

Meanwhile authors reported that for the re-assembling of adaptive course material they require an overview of the applied adaptivity within their course. Also there was a demand for more examples of adaptive courses created with Content-e – which would help them finding the best concept for their own course.

Regarding new development the following research challenges have been set:

• How can added value like didactical principles such as personalisation, adaptation, activation and contextualisation be realised?
• How can more complicated adaptive structures be designed and documented?
• How can adaptive content be made reusable?
• What other actions can be added to the action module?
• How are standards like SCORM met? Note, however, that such standards tend to lack behind new developments and will end up being a subset of the possibilities as described in this article.

References

Abstract. The "Industrial Foundation Classes" is an ISO norm used to define all components of a building in a civil engineering project. IFC files are textual files whose size can reach 100 megabytes. This paper describes a practical method to automatically create hypermedia networks from a semantic and physical analysis of IFC files. The concrete application of this work is a web-based platform called ACTIVE3D BUILD SERVER. This platform lets geographically dispersed project participants—from architects to electricians—directly use and exchange project documents using a 3D Manager system. This system, which evolves during the life cycle of a civil engineering project, is dynamically built according to user models (a combination of semantic trade and access rights).

Introduction

The exponential development of Internet tends towards two fields which seem opposite. On the one hand, the visual aspect, where the text which initially composed the pages of the first Web sites was replaced by images and animations. One notes in this field the opening of software such as Flash of Macro-Media. In other hand, the informative aspect has been developed considerably. The information given by the sites becomes intelligent, adaptive, according to the behaviors of the Net surfers. The new projections as regards interconnection of data bases with HTML pages allowed the creation of new dynamic sites. To date, a Web site must be animated, therefore attractive and intelligent, active and interactive. Nevertheless, there are many limits. In the field of the visual aspect, the 3D representation is in full growth on Internet. Nevertheless it is often limited to small animations, because the resources necessary to use the 3D on the network are too significant. As for the informative aspect, it is still too often limited to the interfacing of a data base with HTML code, using a poor hypertext network to associate part of textual information. The construction of complex systems inter-connecting several heterogeneous data bases is still developed only at the stage of research. Moreover, the association of multimedia information is often organized in a predefined scenario (such as in the Flash Software) and is not dynamically generated according to a semantic view defined in user models. To answer these problems, we developed a new technology called ACTIVE3D. Its
principal objective is to create a dynamically generated hypermedia [1, 2] network between the visual representation and various heterogeneous information sources. Our Approach is based on the “Industrial Foundation Classes” [3, 4] which is an ISO norm to define all components of a building in a civil engineering project. IFC files are textual files whose size can reach 100 megabytes. IFC files are cyclic graphs combined with hierarchical graphs. Due to their size and their structure, their handling and sharing is a complex task. To resolve this problem we have developed a method based on a specific ontology to analyze and to decompose the IFC structure. From this ontology [9], we have developed context trees. These trees are user models that allow to dynamically building hypermedia networks according to the business domain of each group of users. The main advantage of this hypermedia network is to associate various business objects derived from IFC files with semantic trade, documents and 3D representation. Moreover, our system is adaptive and based on the user and the context in which it is used. The users of our platform do not have the same level of knowledge nor the same objectives and access rights. The system adapts thus navigation, the presentation of knowledge for each user according to his profile filled by the user with the inscription.

This paper describes a practical method to automatically create hypermedia networks from a semantic and physical analysis of IFC files. The concrete application of this work is a web-based platform called ACTIVe3D BUILD SERVER. This platform lets geographically dispersed project participants—from architects to electricians—directly use and exchange project documents using a 3D Manager system. This system, which evolves during the life cycle of a civil engineering project, is dynamically built according to user models (a combination of semantic trade and access rights).

IFC Files Description

The “Industrial Foundation Classes” (IFC) is an ISO norm to define all components of a building in a civil engineering project. An example of IFC file structure is given in script 1. This file described a building with more than 111000 business objects (one lines per object). To understand the complexity of the IFC, this section presents the IFC model level and the IFC instances level.

ISO-10303-21:
HEADER;
FILE_DESCRIPTION (["AutocAD generated IFC file.", '2.1']);
FILE_SCHEMA ('IFC2X_FINAL');
ENDSEC;
DATA;
#1 = IFCORGANIZATION ('GS', 'Graphisoft', 'Graphisoft', $, $);
#3 = IFCPERSON ($, 'Undefined', $, $, $, $, $);
#4 = IFCORGANIZATION ($, 'OrganizationName', $, $, $);
#5 = IFCPERSONANDORGANIZATION (#3, #4, $);
#7 = IFCUNIT ('.', LENGTHUNIT, 'METRE');

#111029 = IFCRELCONTAINEDINSPATIALSTRUCTURE ('25wKeWex9Qp5PukCllc', #6, 'BuildingStoryContainer', 'BuildingStoryContainer for Building Elelements', (#111007), #110989);
#111030 = IFCRELAGGREGATES ('216d05yj3QqeDehe60F', #6, 'BuildingContainer', 'BuildingContainer for Building Elelements', (#111007), #110989);
IFC Model

IFC files are made of objects and connections between these objects. The attributes in the objects, describe the "business semantic" of the objects. The connections between objects are represented by "relationship elements". The IFC model is an object model modelled with the EXPRESS language [5]. This model describes approximately 600 classes. There are three types of IFC classes: object classes, relationship classes and resource classes.

1. The object classes consist in a triplet (GID, OS UF), where GID defines the identifier of the IFC object, OS defines the ownership features of this object and FU are the functional units. These functional units define the context of use of the classes (i.e. the geometrical representation, its localization, its composition, etc). In the script 1, the #5 element of the type IfcPersonAndOrganisation reference the #3 and # 4 elements.

2. The resources classes constitute the set of attributes used in the description of the functional units. These resources are organized in a hierarchical graph.

3. The relationship classes represents the various relations (relation of capacity, relation of aggregate, etc.) between the object classes and has functional units. The corresponding elements are prefixed by Ifc-Rel. The IfcRelAggregates element from Script 1 having the identifier #111030 constitutes a relation of aggregate between the element #30 and the following element list (# 34, # 16236, # 29699, # 56800, # 62077, # 67336, # 72633, # 91702, # 110989). The element #110989 is also referred by the element #111029 which is a link called IfcRelContainedInSpatialStructure. This means that if an element can be referred by several elements then two elements can mutually refer them by the intermediary of one or more relations. This mutual reference forms a cyclic graph.

Script 1. Example of IFC file

IFC files are made of objects and connections between these objects. The attributes in the objects, describe the “business semantic” of the objects. The connections between objects are represented by "relationship elements". The IFC model is an object model modelled with the EXPRESS language [5]. This model describes approximately 600 classes. There are three types of IFC classes: object classes, relationship classes and resource classes.

1. The object classes consist in a triplet (GID, OS UF), where GID defines the identifier of the IFC object, OS defines the ownership features of this object and FU are the functional units. These functional units define the context of use of the classes (i.e. the geometrical representation, its localization, its composition, etc). In the script 1, the #5 element of the type IfcPersonAndOrganisation reference the #3 and # 4 elements.

2. The resources classes constitute the set of attributes used in the description of the functional units. These resources are organized in a hierarchical graph.

3. The relationship classes represents the various relations (relation of capacity, relation of aggregate, etc.) between the object classes and has functional units. The corresponding elements are prefixed by Ifc-Rel. The IfcRelAggregates element from Script 1 having the identifier #111030 constitutes a relation of aggregate between the element #30 and the following element list (# 34, # 16236, # 29699, # 56800, # 62077, # 67336, # 72633, # 91702, # 110989). The element #110989 is also referred by the element #111029 which is a link called IfcRelContainedInSpatialStructure. This means that if an element can be referred by several elements then two elements can mutually refer them by the intermediary of one or more relations. This mutual reference forms a cyclic graph.
IFC Instances

The study of the IFC instances reveals the complexity of the overlap between instances of relationship classes and instances of object classes. At this level, there exist two types of link between objects. We called them the indirect link and the direct links. The indirect links are defined by the instances of the relationship classes.

The direct links are defined by the instance of resource classes. The indirect links are characterized in the figure 1 by ▲. The object instances of the architecture field become semantic elements. In the figure 1, these elements are graphically represented by ●. The resource instances are represented by △.

Figure 1 shows the indirect links between the semantic elements using a relationship element. Figure 2 shows the direct bonds between semantic elements, they are noted in red. There are two types of direct links. The first type defines the resources of the element. These resources are structured using a tree structure. The second type defines a direct link between two semantic elements. The IFC model defines only one type of links between two semantic elements. This is the placement link between the semantic elements for design of a building in a 2D/3D scene. This relation is carried out by the IfcLocalPlacement attribute of the semantic element. It defines the reference mark of the current object compared to the reference mark of the father object of the direct relation. The set direct link formed by the IfcLocalPlacement attribute forms a tree structure of the 2D/3D scene. The main difficulty is to handle at the same time the cyclic the semantic elements and the hierarchical structure of the 3D elements.

Active3D Methodology

The handling and management mechanisms for IFC files (such as fusion of files into only one, extraction of partial files dedicated to only one context, the
visualization and the storage) will have to deal with a multitude of semantic values for the same object, according to the context of use. To achieve these goals, we defined a hierarchical structure of context called the contextual view. This solution reduces the graph complexity translating a multi-contexts cyclic graph into a set of mono-context trees. This process is done using business rules. An example of a business rule is "a door is in an opening element in a wall". Resulting from this process, many contextual trees are built starting from the IFC files, such as the contextual tree of capacity defining the object composition (a building contains two floors, a floors contains beams, walls, and so on). The figure 3 presents the 3D Manager systems which build a specific user interface made of tree of capacity, 3D scenes and object card file. The navigation between this element is made by hypermedia links which associate a set of semantic elements to a business object (here a slab). The main tree is the geometrical contextual tree which contains the topological relations between the various objects. From these trees, the resulting 3D scene corresponds to a particular business view [6].

![Figure 3. A snapshot of the 3D Manager System](image)

The figure 4 shows the 3D representation of plumbing view of an IFC file. From this 3D view, the users can reach directly the various functional units composing the IFC file. Through this interface, trade information can be manipulated. These business views are textual information, from which specific documents can be generated or associated (technical reports, management information, etc.). In the 3D scene generation process, all the geometry defined in IFC trees is converted into triangular
surface model [7]. During this conversion, the 3D objects are associates with the GID. The GID is the general identifier used to identify each business object of an IFC file. In the script 1, the GID of the IFCRELAGGREGATES object is #111032. This GID is used to link the 3D visualization with the information stored in the databases. All insertion of new data in any base is reference by a GID correspondent to an IFC object. All trees generated in the platform are XML trees [8]. We have developed a specific database schema dealing with the semantic and the 3D aspects of the IFC. The trees and the component elements are stored in a relational database and manipulated using the SQL. From this database and the GID, all types of information can be attached to the 3D visualization of business Object.

Figure 4. A 3D scenes corresponding to the plumbing context

From the definition of the contextual trees, we have defined the corresponding XML grammars. Thus, each functional unit and each context are manipulated as XML documents. These grammars, specific to a civil engineering profession, are used to format IFC data exchanged during the life cycle of a civil engineering project. The contextual trees are transformed using XSL style sheets. These transformation processes are web services. IFC Services provides on the ACTIVe3D server are XSL processes associated with a context. The use of XSL is extended to generate other documents such as technical reports and so on. In the same way, the graphic contextual trees are transformed into X3D documents. Thus, the 3D scene is personalized according to the service called. Moreover, the graphic elements preserve connections with the information system containing management information. These connections allow the 3D scene to carry out queries on the information system concerning the graph elements.

A complete description of this method and the presentation of the resulting IFC schema from our relational database are beyond the scope of this paper.
Conclusion

This paper describes a practical method to automatically create adaptive networks from a semantic and physical analysis of IFC files. The industrial result of this work in a Web collaborative platform called ACTIVe3D Build Server. In this platform, the Hypermedia network is based on a 3D visualization which lets participants move around in the building being designed and obtain information about the objects that compose it. 3D Objects, trade objects and other documents are connected in the network by semantic links defined in context trees.

Currently, 126 people use the application daily and manage sixteen projects. The ACTIVe3D was rewarded for the technological innovation gold medal to the international show BATIMAT in Paris, November 2003.

Now, we are study the update of the IFC database from multiple IFC sources. This multiple update of IFC files generates structural and semantic conflicts (for example, a heating pipe crossing a door). The resolution of these conflicts passes by a semantic and geometric analysis of the associated contextual trees.

Bibliography

http://ateos.it.teithe.gr/~cs1mgad/docs/aspects_of_hypermedia.htm
http://wwwis.win.tue.nl/zh/
[3] International Alliance for Interoperability, 
http://www.iai-international.org/
http://cig.bre.co.uk/iai/uk/documentation/IfcR2x_Final/
http://www.w3.org/XML/