Successful data mining using context aware hypermedia

af
Leif Bredgaard Andersen

25-05-2005

Leif Bredgaard Andersen, studerende
Bent Guldbjerg Christensen, vejleder
## Table of contents

1 INTRODUCTION .............................................................................................. 4  
1.1 THE SCIENTIFIC METHOD........................................................................ 5  

2 RESEARCH ........................................................................................................ 6  
2.1 RELATED WORK .......................................................................................... 6  
2.1.1 AutoLink by Google................................................................................. 6  
2.1.2 Apache Lucene........................................................................................ 7  
2.1.3 The Stick-e Document.............................................................................. 8  
2.1.4 Ontolica by Navigo Systems A/S.............................................................. 8  
2.1.5 IDOL Server by Autonomy ...................................................................... 9  
2.2 HOW IT WAS DONE BEFORE ..................................................................... 10  
2.3 CONCLUSIONS ON RESEARCH.................................................................. 11  
2.3.1 What is the context of digital data and how to model this?..................... 11  
2.3.2 Which type of links can be used to build a context-based link-structure and how can this be modelled? .................................................... 12  
2.3.3 How can context-based link-structures be presented to the end-user?... 12  

3 DESIGN ............................................................................................................ 13  
3.1 APPLICATION STRUCTURE........................................................................ 13  
3.1.1 Data Analyzer ....................................................................................... 13  
3.1.2 Link Analyzer ........................................................................................ 14  
3.1.3 Presentation Manager........................................................................... 14  
3.2 DEXTER REFERENCE MODEL.................................................................. 15  
3.2.1 Short description of Dexter Reference Model ........................................ 15  
3.2.2 Compliance with Dexter Reference Model............................................. 16  
3.3 THE DIGITAL CONTEXT............................................................................. 16  
3.3.1 Taxonomy ............................................................................................. 16  
3.3.2 Clustering ............................................................................................. 17  
3.4 LINK STRUCTURES AND CONTEXT MODELS......................................... 17  
3.4.1 Context Models ..................................................................................... 17  
3.4.2 Link Structures ...................................................................................... 22  
3.5 USER PRESENTATION ................................................................................... 25  

4 PROTOTYPE ................................................................................................... 26  
4.1 INTRODUCTION ........................................................................................... 26  
4.2 INTERNAL DESIGN ..................................................................................... 26  
4.3 APPLICATION DESIGN ............................................................................... 29  
4.3.1 Content View......................................................................................... 30  
4.3.2 Histogram............................................................................................. 31  
4.3.3 The Menus............................................................................................. 33  
4.4 DESCRIPTION OF THE SOURCE CODE.................................................. 33  
4.4.1 UContentViewMan.pas.......................................................................... 33  
4.4.2 UDataManager.pas............................................................................... 35  
4.4.3 UHistogramViewMan.pas ..................................................................... 36  
4.4.4 UXMLDecoder.pas................................................................................ 37  

5 CONCLUSION ................................................................................................. 38  
5.1 WHAT IS THE CONTEXT OF DIGITAL DATA AND HOW TO MODEL THIS? ....... 38  
5.2 WHICH TYPE OF LINKS CAN BE USED TO BUILD A CONTEXT BASED LINK STRUCTURE AND HOW CAN THIS BE MODELLED? ....................... 39
5.3 How can context based link structures be presented to the end-user?
........................................................................................................................... 39
5.4 Overall Conclusion ....................................................................................... 40

APPENDIX A. XML SCHEMAS ........................................................................ 41
  Context Model ................................................................................................. 41
  Link Structure ............................................................................................... 43

APPENDIX B. XML DOCUMENT EXAMPLES ............................................... 44
  Context Model ................................................................................................. 44
    Person ............................................................................................................. 44
    Car ................................................................................................................. 44
  Link Structure ............................................................................................... 44
    Example ........................................................................................................ 44

APPENDIX C. SOURCE CODE .......................................................................... 45
  PresentationManager.dpr ............................................................................... 45
  UAbout.pas ..................................................................................................... 45
  UContentViewMan.pas ..................................................................................... 45
  UDataManager.pas ........................................................................................ 47
  UHistogramViewMan.pas ............................................................................... 52
  UMain.pas ....................................................................................................... 54
  UXMLDecoder.pas .......................................................................................... 56

APPENDIX D. BIBLIOGRAPHY ...................................................................... 59
1 Introduction

As the Internet keeps growing, more and more data becomes available. However, even though some quite effective search engines exist (Google etc.) it is still not an easy task to find the data you are looking for as the search engines are not aware of the digital context – they can find words in the text, but not the digital context relevant for these words and thus it cannot find web-sites sharing the same digital context. The Internet is just one example of a system having this problem. Examples of other information systems could be marketing systems, libraries and forensic analysis, just to mention a few. However, this problem does not only count for electronically stored data, but also for physical items such as books and criminal evidence. The physical items are often, though, represented electronically using a RFID tag or similar. This makes it quite a lot easier to search through physical data, if the meta-data stored about them has a proper level of detail.

The objective of this paper is to describe an open hypermedia system, which can help providing a digital context-based view on large amounts of data. The main focus will be on how the meta-data is linked. I will design the hypermedia system using the Model-View-Controller pattern [GoF95]. The main advantage gained from using this design pattern is the three-tier division of the system and because it becomes much easier to replace any of the three layers, as the interfaces between these are well specified.

Basically the hypermedia system consist of 4 parts:

- The database – where we store link- and meta-data (Model).
- The data analyser – produces meta-data from the input data usable by the link analyser (Controller).
- The link analyzer – analyzes the meta-data from the data analyzer and produces links between these (Controller).
- The user presentation – produces a graphical view of the link data produced by the link analyser (View).

From a hypermedia point of view the link analyser is the most interesting part, as linking data, based on given rules, is what hypermedia is all about. Every time new data is added to the system it should analyse all data again to find whether new
relations can be created, i.e. it is not a terminating process. The 3 other parts will be handled as well but not as thorough as the link analyser.

Based on the above I will try to answer the following three questions through this paper:

1. What is the context of digital data and how to model this?
2. Which type of links can be used to build a context-based link-structure and how can this be modelled?
3. How can context-based link-structures be presented to the end-user?

1.1 The Scientific Method

To gain proper knowledge for answering the three questions stated above, I will look at other systems to study what they have done to develop a context aware hypermedia system (related work 2.1). I will also look at how it can be done in a manual fashion, and finally come it with the highlights from these studies – what is good and what is bad.

Based on this related work I will come up with a design for the system (design 3). This design will primarily cover the data models to model the link structures and the context models.

To give an example on how the models from 3 can be presented to the user I have developed a prototype, which is described in 4. I will describe the prototype itself and comment on source code of special interest.

Finally I will conclude in chapters 2, 3 and 4 by answering the three questions from the introduction and make an overall conclusion on these answers.
2 Research
To gain knowledge on how other context aware systems work I will, in the following, study other systems and find the essentials on how they work.

For the purpose of this system the primary focus will be on how they define context, how the systems find the context of the given data, how they are presented to the user and how to navigate through them.

Also, for this system it is important to be able to link data based on context similarities. I.e. for a library the context for a given book could be naval aviators. The purpose of the system would be to find other books with a similar context – e.g. books about combat aircrafts. One should keep in mind that one item may have more than one context and it should be possible to set/find the main context of this given item.

2.1 Related Work
During my research I have found following related systems I want to look further at:

- AutoLink by Google
- Apache Lucene
- The stick-e document by P.J. Brown
- Ontolica by Navigo Systems A/S
- IDOL Server by Autonomy

In the following I will describe the systems in details and come up with some conclusions on what would work and what would not with regards to the system I am describing through this paper.

2.1.1 AutoLink by Google
The AutoLink by Google [AutoLink] is a service that scans a given webpage for ISBN numbers and postal addresses. If an ISBN number is found a link is made to Amazon.com, and if a postal address is found, a link is made to a mapping service. For now, this mapping service only works in the United States though.

The AutoLink service itself is not interesting but rather how it manages to find the postal addresses and the ISBN numbers. However, there is not much information available on how AutoLink really works but I would guess they use some sort of pattern matching to describe the ISBN and the addresses – this could be done through a regular expression. From an AutoLink point of view the ISBN number and the
postal addresses represents the digital context. It is not a digital context in the sense that it gives you information on the topic of the given page but in the sense that AutoLink recognizes it as ISBN or postal address.

The only links, which really can be used to model this, is hyper links from the HTTP protocol. The links are generated on load time – that is when the user fetches the page.

The links are also modelled the same way as links are on a regular webpage, so colours etc. depends on the actual webpage. There are good and bad things about this way of displaying the links. The good thing is that the users see things, as he/she is used to, the bad thing is that the view may change from page to page. The only way you can describe the links is based on the users knowledge on what postal addresses and ISBN numbers look like. The way AutoLink have implemented links looks a lot like Generic Links, known from Microcosm [Generic Links] in the sense that you have many items around the world pointing to the same item, either by reference of an ISBN number or by reference of a postal address.

2.1.2 Apache Lucene

Apache Lucene [Lucene] is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

Lucene is not a hypermedia system, but only a search engine. However, it can be used as a base for this system as it has high performance and highly scaleable with regards to search queries. Another problem could be that it is written in Java, but depending on its implementation it could be a fairly trivial task to convert it to another programming language.

The digital context of Lucene is really only the words, which the user searches for. It cannot determine a context on its own unless we define an index as a digital context. You can generate indices of the data you want to search in. These indices are stored in a database and are used for future searches. These indices can be generated more or less intelligently depending on which analyzer module is used with Lucene. You can e.g. take an analyzer that eliminates commonly used words such as ‘and’, ‘or’, words that does not really add any value to the search.

As referred to above you cannot really talk about a digital context with regards to Lucene, as it cannot autonomously classify a given document by its content.

---

1 The digital context of a document is really an abstract of the content, or simply the topic of the document.
How the search results are displayed to the user depends on the implementation. Lucene is not an application itself but a framework to integrate into other applications.

2.1.3 The Stick-e Document
The Stick-e Document [Stick-e] is a framework for context-aware applications. It describes a new document type, stick-e document, which is aimed at context-aware applications. The stick-e document is build by a number of smaller components called stick-e notes. Each note consists of two parts:

- The content
- The context.

Stick-e document is designed to work at a PDA, or another handheld device you bring along. Basically it works as post-it notes – i.e. you make a note and attach it to a given position. When you return to that position you see the note again. For stick-e notes the content is the message you write on your post-it and the context is the position on which you place the note. The context of the note depends on the positioning equipment available – e.g. a GPS coordinate.

The context of the stick-e notes is related to a given physical position – e.g. by a GPS coordinate, Bluetooth etc. The context depends on the physical item it is connected to and thus the model must allow different types of context to be presented. The context cannot be derived automatically.

As with the context also the links must support different type of contexts.

This is presented to the user as post-it note popping up when he/she is getting close to a given position. This can be a physical place or simply another person with a PDA. Unfortunately the stick-e note can only present a text message to the user and not other kinds of data as would be needed for this system.

2.1.4 Ontolica by Navigo Systems A/S
Ontolica [Ontolica] is an electronic library developed by Navigo Systems A/S. It is a system developed to maintain an organization’s documents and make them easy to find by indexing them onto shelves like in a library. The system is build on top of a Microsoft Portal server.
The system works by doing taxonomy\(^2\) on a given document. This taxonomy is based on an information model, which defines what characterize a document of a given class. It can process various types of documents and even determine which language they are in. Based on this taxonomy it can place the document on the correct shelve(s) – a document may fit in on several shelves. This information model can be build in either of two ways – one is to learn the model based on example documents and another is to build the model manually based on knowledge about the data.

Ontolica can do a lot of the tasks required by my system, but still it lacks on some details. The input part seems to work really great as it can take many types of documents and determine the language. The process of doing the taxonomy lacks a bit though, from my point of view, as it cannot automatically determine the context without user intervention. It should be able to determine the context automatically based on the input. The next issue is how to present the data to the user. The average user of this system would like to be able to have data presented in a number of different ways depending on the detail level of the view.

The context of Ontolica is based on an information model and taxonomy. The taxonomy is applied to the documents and based on the results and the information model the context is derived. The context of a document cannot be derived automatically – i.e. you cannot enter a new type of document to the system, and expect the system to place the document correct, unless you train the system to know this type of document.

To build the link structure a database is build containing the documents and which shelf to place the document on – a given document can be placed on several shelves so the system must support that. As mentioned earlier the system is based on a Microsoft Portal server and thus the link structures are based on the ones provided by this server.

### 2.1.5 IDOL Server by Autonomy

The IDOL (Intelligent Data Operating Layer) Server [IDOL], developed by Autonomy, is a server capable of deriving the context of a document automatically based entirely on the content of the document. It can do automatic taxonomy and clustering\(^3\) on a large set of documents. Using clustering you can take a large set of

---

\(^2\) Taxonomy is method for classifying data based on the content. I.e. you find the relevant words in a piece of data that identify the digital context of that piece of data.

\(^3\) Clustering is a technique for grouping data based on the content (digital context).
data and identify the clusters within this data – that is the “hot spots” of the data or simply the main topics of the data. Later on you can apply taxonomy on a cluster to gain further knowledge of the cluster or simply of the entire lot by applying taxonomy on all the data.

The taxonomy is used to find the context of a given document. One can argue that clusters are also contexts, but in that case they are on a higher level as they find documents sharing similar topics.

How these links are structured and modelled is not really described for IDOL, but in some cases hyper-links are used – most likely hyper-links, as we know them from the HTTP protocol.

As for presenting the link structures to the user Autonomy has done a great job – the data can be presented to the user in many ways. A really nice way to get a view on a large amount of data is the spectral view. In this way, clusters of data are visualized using brighter colours and this makes it easier to see on which areas to focus. The bigger and brighter a cluster is the more relevant information it contains. Also, Autonomy provides an API for the IDOL server so you can apply your own GUI based on IDOL as the data analyzer.

2.2 How it was done before

Before we had systems to do taxonomy and clustering automatically for us one had to sit down and perform the entire process manually. This took quite a while for large amounts of data and there was a great risk that errors may occur as humans performed the analyses.

Long process time and human errors were not the only problems performing these tasks manually. Clustering and taxonomy were really difficult to perform effectively as you only had one document and it could easily fit into many shelves. This problem could be overcome by making index cards for the different taxonomies and clusters.

The next problem faced after the long process time for taxonomy and clusters would be to get an overview of all the data. You would have all the words from the taxonomy and you would have the clusters, but what would you do with this information? I guess clusters would be the better solution, as you would be able to categorize a bunch of documents by their topic(s). The words from taxonomy could be used to make some sort of index to make it easier to find documents containing given words – the only problem about this was that you would not know which
context was used unless, of course, you had written it down somewhere or looked back into the document.

What this really would give you was a manual search engine performing the same job as e.g. Google does – i.e. you would have a nice index and some clusters but you would not know what all your data was really about and this is where the context aware hypermedia system comes in to help deriving the digital context and get a greater perspective on things.

2.3 Conclusions On Research
During my research I have studied how others have tried to solve these problems and how it could have been done in old-fashion manners. I have looked at what is good and what is not in regards to the questions stated in the preface.

In the following I will look at these questions and compare them to the applications I have studied during the research to get a general idea on what others have done.

2.3.1 What is the context of digital data and how to model this?
Looking at how the digital context is defined you will see it can be done in different ways. The context varies from physical positions, given by e.g. GPS coordinates, to a collection of words. I.e. the context model must support different types of contexts or other means of distinguishing one type of context from another.

Even though we have to support several types of context not all contexts are suitable for automatic discovery. E.g. the physical positions from The Stick-e Document is not suitable for automatic discovery, unless, of course, the system is aware of the users interests, but for the scenario described in The Stick-e Document it is not possible as it is purely based on physical locations.

For other types of data it is much more straightforward to derive the digital context. Systems such as Ontolica and IDOL, relies on statistical analysis of the documents to find the context. This counts for both taxonomy and clustering – the difference is what data comes from the process. From the taxonomy you will gain a list of words representing the content of the document whereas the clustering will produce a list of documents, which have similar content.
2.3.2 Which type of links can be used to build a context-based link-structure and how can this be modelled?

For most of the systems studied in 2.1 it is not really clear how they have chosen to link the context and the actual document. However, some of them are quite obvious like AutoLink. In AutoLink the link is simply modelled by a regular link, as we know them from the Internet.

The Stick-e Document system has a different approach. It link content and context by using an electronic post-it note. Many notes can share the same context. The documentation does not describe what actually triggers a given note. Of course it is getting close to a physical position but it is not described how a specific note is found and how close you should be to the location to trigger a given note.

For the other systems it is not really clear how the link-structure is modelled but Ontolica is based on Microsoft Portal Server.

2.3.3 How can context-based link-structures be presented to the end-user?

There are numerous ways to present the link-structures to the user. How to do it entirely depends on what kind of information you want to show the user – whether you want to show how data are linked, how they belongs together in clusters or simply if you want to give the user some sort of statistics on the data to help clarifying their relevance.

The ones for Stick-e Document and AutoLink are very simple, but then again, you do not have a great level of details opposite IDOL and Ontolica, which both requires much more depth in order for the user to be able to get an overview of the data.

AutoLink simply displays the original text converted into a hyperlink pointing either at a book service or a map service, depending on whether the link points to a postal address or to an ISBN number.

IDOL and Ontolica, however, have many more opportunities regarding display of link-structures. Especially IDOL has some really nice features such as a spectral view on clusters. Like for audio, the spectral view gives a really good picture on how the energy of the data is located or rather, for IDOL, how clusters develop over time.
3 Design
In the following I will, based on the conclusions from research, describe a design for the system. The main focus will be on data models for link structures and the digital context. I will not focus on how to find the digital context, as it is out of the purpose for this paper, instead I will focus on the models for identification and linking of the digital context. Also user presentation will only be covered briefly in the design but more thorough during the prototype.

3.1 Application Structure
This system will be divided into three modules:

- Data Analyzer
- Link Analyzer
- Presentation Manager

These units may be separate applications but not necessarily. Typically, the Data-analyzer and the Link Analyzer will reside on a server, whereas the Presentation-manager is a client-side application.

In the following I will describe the three modules and their purpose in more details.

3.1.1 Data Analyzer
The Data Analyzer analyzes all the source data to extract the digital context into context-models. You really go from unstructured to structured data.

Extracting the digital context of a document is not an easy task and it takes several big steps to complete. One can choose either to develop software to do this, or one can choose some third party software. If you can find any third party software suiting your needs it would be preferable to use this, as it is a quite complex task to develop a context aware analyzer. In The Digital Context (section 3.2), I will get into more details on how the digital context can be extracted from data. Both the taxonomy and the clustering apply to the Data-analyzer. Taxonomy is used for finding the important words of a document, the digital context, and clustering for grouping them together in logical units.

The context-models are defined by XSD [XSD] (XML Schema Definition) and the result from the Data-analyzer will result in a number of XML documents representing
the digital context. What the XSD look like, I will describe closer in the context-models (section 3.4.1).

### 3.1.2 Link Analyzer

The Link Analyzer analyzes the properties of each item in the context model to find similarities either by means of singular words or, from a larger perspective, as a cluster. The result of this is stored in link-models defined by an XLink format.

There are many ways the context-models can be linked by XLink depending on what data you want to look at and how you want to link these. For this reason it will be convenient to have a plugin mechanism to make it easy to change the way your data are linked, or simply to add new linking mechanisms. I will describe the XLink closer in the link-structures section further down.

What the Link-analyzer really does is to look at all the data from the Data-analyzer, and try to link these based on the type of information they contain. This type of linking is called clustering. However, the complexity of the cluster depends on what kind of data you are looking for – e.g. you may want too look for webpages concerning US Cars from 1967 or you want to look for books written by a certain author. The last example is the simpler one to compute, as you only want to look for the author whereas the first example requires that you find pages containing these words, but also pages with a similar content. This is basically some sort of clustering. Clustering is described in The Digital Context (section 3.3) along with taxonomy.

The reason why XLink has been chosen to link the context models is that it is also defined by XML, and gives great opportunities of linking XML documents. We can use the simple one, as I have chosen to do, to simply make references to other XML documents or you could choose to use the XPointer [XPointer] to point to specific points in a XML document.

### 3.1.3 Presentation Manager

The Presentation-manager handles displaying the results from the Link-analyzer to the user. How the data is presented to the user depends on the type of data at hand. The presentation could be a graph on how a number of webpages were visited, a list of related books or a spectral view of a cluster to show which items apply the better to the topic – just to mention a few examples.
One of the great tasks for the presentation manager is to select which data to show to the user based on a given view. I will get further into details on that aspect in the User Presentation part.

### 3.2 Dexter Reference Model

If we look at the application structure (section 3.1) we will see a clear connection to the Dexter model, with some slight differences though, but it is basically the same. In the following I will describe how my system complies with the Dexter Reference Model [Dexter] (DRM) and how it differs as well.

#### 3.2.1 Short description of Dexter Reference Model

The Dexter Reference Model (figure 1) consists of three layers.

- **The Run-time Layer**
  Handles the presentation of data to the user.

- **Presentation Specifications**
  Contains descriptions on how the different types of data should be presented to the user.

- **The Storage Layer**
  Contains all the data and links found by the Data Analyzer.

- **Anchors**
  Is a reference into a given position of a file.

- **Within-component Layer**
  This layer is responsible for the content selection of individual components through anchors.
3.2.2 Compliance with Dexter Reference Model

This system complies with DRM in many ways but there are slight differences.

The run-time layer, presentation specifications and the within-component layer are all integrated into the client application, as it all has to do with presentation of the data to the user. This means the client application can grow quite large as more and more data types are supported. To make it easier to support new data types, and limit the size of the application by removing unneeded data types, it would be an advantage to make a plugin framework. This way you simply develop a new plugin for a new data type and insert it into the client application.

The Data- and Link Analyzers maintain the storage layer and the anchors. They are responsible for creating new links and meta-data for the data feed into the system. Different types of data exist depending on whether it is the Data- or the Link Analyzer working. The Data Analyzer works on the nodes containing the data. It generates context anchors, which then are analyzed by the Link Analyzer generating clusters of context anchors.

3.3 The Digital Context

In the following I will look at the digital context, or more precisely how to find the digital context of a given piece of digital data. I will not go to deep into this, as it is out of the purpose of this paper, but I will give an introduction to the methods of extracting the digital context. Basically there are two methods used in the systems I have researched (with respect to digital context) – taxonomy and clustering.

Taxonomy is used for classifying data based on the contents by finding its digital context whereas clustering is for organizing the documents based on their classification but also on document specific properties such as author, e-mail address etc.

3.3.1 Taxonomy

Taxonomy is the process of making a classification of the given document. This can be done in many ways, but a widely used one is to find all the important words in the document based on statistics.

It works as follows:

1. All normal words, such as “and”, “or”, “the” etc. are removed.
2. A filter is applied to the rest of the document to find which words, or set of words are used more often and thus are identifying the digital context of the document.

3. All the words identifying the digital context are extracted and stored into a database word by word and with a reference to their document.

The essential thing of the taxonomy is that it works on a single document at a time. I.e. you will get the context from a document, but you will have to link the documents yourself.

3.3.2 Clustering
The main difference between taxonomy and clustering is that the taxonomy only produces a list of words describing the digital context for a given item. The clustering mechanism is quite different. It takes a whole bunch of data and put them in groups based on the digital context.

Clustering is a really great technique for identifying documents belonging together and thus limits the amount of data to further process. When you have identified one or more clusters to look at, you can then apply taxonomy on these clusters to find the digital context of each element in the cluster.

3.4 Link Structures and Context Models
In the following I will describe and discuss the link-structures and context-models needed for the Link- and Data-analyzers respectively.

The link-structures and context-models are both described using XML. The context-models are through XML schema and the link-structures are through XLink.

3.4.1 Context Models
Obviously any kind of data will not have their digital context defined in the same manner. E.g. a book and an e-mail have different properties but they may still share properties such as author and subject. In the following I will try to come up with a general context-model applicable to any kind of data.

Since we know that data will share some common properties a reasonable approach will be to model the context-model as an object oriented hierarchy of nodes.
We can start off by defining the top-level node. We have to be able to identify each node individually and probably apply some sort of specific property to this node. This gives us following node (Figure 2):

<table>
<thead>
<tr>
<th>Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID:</td>
</tr>
<tr>
<td>Property:</td>
</tr>
</tbody>
</table>

*Figure 2 – The Node*

The ID is a unique identifier of the given node – we must be able to uniquely identify each node from its ID and never may two nodes exist with the same ID.

Property is a variable attribute list consisting of pairs (key, value). In a property we store values, which does not fit into the model or are client specific.

Clearly, by means of Property, all we need to model the entire system is this node as we can fit any kind of information into Property. However, this is not convenient as it becomes difficult to form a general view on the data – especially from an application’s point-of-view as it makes the job finding the correct type harder and we are not able to distinguish two fields from their type. In order to avoid this problem we will have to look further at the data to feed into the system in order to identify specific nodes.

3.4.1.1 Identifying Node Types

To identify what type of nodes we have is not an easy task to perform. First we have to identify all types of input we can have for the system. Needless to say, this amount of data can grow huge, but we can generalize a lot as much of the data look alike even though it is quite different. E.g. a news message and an e-mail are pretty much the same.

If we look at data from a higher perspective, we can easily identify some identities we should have in the context-model. However, to keep this simple, and only show the concept, I will focus on a few item types only. Some of these, however, will be quite central and can be used for many other types of data.

When we think of published information in general, a very obvious entity pops up – the author. When thinking of the author you easily get a recipient on your mind as well. This leads us to an entity type called Person. Person needs a set of attributes in order to identify the person. Of course the Person entity inherits from the Node identified earlier, and thus we have the ID as a unique identifier, but there are also
other properties such as name, phone number, social security number, e-mail address, postal address and what else information we may find, which can help identifying this given Person. It seems like a pretty straightforward task to identify entities for the context model, but depending on your data source this can be quite a task. You have to look through all kinds of data in order to identify which attributes you want to store for this particular piece of data.

For simplicity I have chosen a few examples of entities and identified the most significant properties from these including the Person identified earlier (Figure 2). The entities I have chosen to use are:

- Person
- E-mail
- Chat
- Car

The three first items are all electronically defined by the system, but the Car is a physical item linked into the system by a RFID\(^4\) tag. The reason for bringing the Car along is to show how physical items can be linked into the system and thus analyzed as well as pure electronic data. For this reason I will argue that a PhysicalItem descendant from Node is needed as for all physical objects counts we need a way to identify them in the real world – a nice way to link a node to a real world item would be by using a RFID tag. See [Physical Hypermedia] for an example on how RFID can be used to integrate physical items into Open Hypermedia.

I have chosen to use RFID tags to identify physical items in this seems but, needless to say, there are also other means of representing the physical world in an Open Hypermedia system. Examples of other means could be barcodes and 2D tags [2D barcodes]

### 3.4.1.2 The Context Model

Once we have identified the node types we can use this to derive the context model. The context model in this is defined by a XSD document. In the following I will describe the top-level design of XSD along with the motive for this design.

You can look at the context model as an abstract of a given piece of data defining that piece of data’s digital context. A document should exist for each data element

\(^4\) Radio Frequency Identification (RFID) is a method for remotely storing and receiving using devices called RFID tags/transponders attached to a physical item. A RFID tag can e.g. be a small sticker.
analyzed and most have a unique identifier. The documents can be stored in either a file system or a SQL database. We will, however, use one XSD to define all possible documents.

Figure 3 shows a picture of how I have chosen to define the context model. Node is the top level containing the attributes in common for all nodes in the context model. All the digital data I have defined on one level (E-mail, Person and Chat), but as mentioned earlier I also want to represent physical items in the model. To model that I have added an extra level for the physical items called Physical Item. Besides the data inherited from Node it also contains a RFID to identify the physical item. See 3.4.1.2.1 below for detailed information about the XSD for the context model.

3.4.1.2.1 XSD

In the following I will describe essential parts of the XSD for the context model. The entire XSD can be located Appendix A.

3.4.1.2.1.1 Node

The Node type represents the top-level of the context model hierarchy. It has two elements and one attribute.

The attribute is the id – a unique identifier for the given node. The elements define a Property and a ContextWord.
The Property value defines a list of pairs (key, value) defining custom properties for a given item. ContextWord is a list of words describing the digital context of the item.

### 3.4.1.2.1.2 PhysicalItem

```xml
<xsd:complexType name="physicalItemType">
  <xsd:complexContent>
    <xsd:extension base="nodeType">
      <xsd:sequence>
        <xsd:element name="RFID" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

The PhysicalItem type represents the physical item in the digital context. It is an ancestor from the Node (3.4.1.2.1.1).

Besides the elements from Node the PhysicalItem also has a RFID element to represent the RFID tag for a physical item. Needless to say, there are also other means of representing the physical world in an Open Hypermedia system. Examples of other means could be barcodes and 2D tags [2D barcodes]

### 3.4.1.2.1.3 Person

```xml
<xsd:complexType name="personType">
  <xsd:complexContent>
    <xsd:extension base="nodeType">
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string"/>
        <xsd:element name="SocialSecurityNo" type="xsd:string"/>
        <xsd:element name="PostalAddress" type="xsd:string"/>
        <xsd:element name="Country" type="xsd:string"/>
        <xsd:element name="E-mail" type="xsd:string"/>
        <xsd:element name="Sex" type="sexType"/>
        <xsd:element name="Age" type="xsd:positiveInteger"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

The Person is a central object in the context model as a person is involved in most other items. Person owns a car, participate in a chat, sends e-mails etc. I have tried to come up with some of the most common properties for a person, such as name, address, sex etc.

### 3.4.1.2.2 XML

In the following I will give a couple of examples on how a context model look in XML.

Further examples can be found in Appendix B – Context Model.

### 3.4.1.2.2.1 Person

```xml
<?xml version="1.0"?>
```
This describes a typical person. The digital context for this person is that he is a writer. He could also be a recipient, a victim, a criminal, anything really.

3.4.1.2.2.2 Car

This describes a typical car with all available options. This was a victim’s car.

3.4.2 Link Structures

The link structure is the one gluing the context model together to something meaningful to the user. To perform this task, the chosen link structure must comply with different demands. We can set the following requirements for the link structure:

- N-ary links to link many items by using only one link.
- Set the type of link – e.g. to distinguish between clusters and context words.
- Easy to expand if new link types or structures occurs.
- Well proven and well known to the world.

A link structure that complies with these requirements is the XLink format. The XLink format is developed by W3C just as XML and is a format for linking into XML documents.

---

5 See http://www.w3.org
Figure 4 shows the link model. It is a pretty simple model consisting of only one item. One should not be fooled by the simplicity of the model though. An XLink item can become quite complex, if we are linking many items.

3.4.2.1 The Link Structure

In the following I will describe essential parts of the link structure and show a couple of samples of how it can be used to link data. I have kept the link structure quite simple in order to make it more efficient to work with, even for large amounts of data.

3.4.2.1.1 XSD

In the following I will describe the XSD behind the XLink structure I used.

3.4.2.1.1.1 LinkStructure

The LinkStructure is the base type of the link structure. It has three attributes and three element fields.

The first attribute is an id field, which a unique identifier ensuring a unique identification of the given LinkStructure throughout the entire system. LinkStructure is defined as an extended XLink type, and also has a title property where we can set the title we want for the given LinkStructure.

The three elements are loc, arc and res identified by Locator (3.4.2.1.1.2), Arc (3.4.2.1.1.3) and Resource (3.4.2.1.1.4) respectively. These are defined in accordance with the XLink standard from W3C, where these are also defined.
Locator defines a link to an item from the content model or to another link structure.

A Locator contain 4 attributes:

- xlink:type – has a fixed value “locator” to identify it is the XLink type “locator”
- xlink:href – defines the URI to the item we want to link
- xlink:role – here we can define the role of the Locator by using a URI
- xlink:label – defines a label used by Arc to link Locators together

3.4.2.1.1.3 Arc

Arc defines a link between two locators.

An Arc contain four attributes:

- xlink:type – has a fixed value “arc” to identify it is the XLink type “arc”
- xlink:from – defines the label of the item at which the link start
- xlink:to – defines the label of the item at which the link stop
- xlink:title – defines the title of the link

3.4.2.1.1.4 Resource

Resource defines a list of attributes. These would normally be a list of context words common for the items in the given link structure.

A Resource contain 4 attributes:

- xlink:type – has a fixed value “resource” to identify it is the XLink type “resource”
- xlink:role – here the can define the role of the Resource by using a URI
- xlink:title – defines the content of the Resource – typically a context word from the context model
- xlink:label – defines a label used by Arc to link the Resource together with other items

### 3.4.2.1.2 XML example

```xml
<?xml version="1.0"?>
<contextLink xmlns:xlink="http://www.w3.org/1999/xlink" id="2" xlink:type="extended" xlink:title="Test">
  <loc xlink:type="locator" xlink:href="file://2.xml" xlink:role="role://person" xlink:label="B" />
  <loc xlink:type="locator" xlink:href="file://4.xml" xlink:role="role://e-mail" xlink:label="D" />
  <arc xlink:type="arc" xlink:from="A" xlink:to="D" xlink:title="has sent" />
  <arc xlink:type="arc" xlink:from="D" xlink:to="B" xlink:title="is delivered to" />
  <arc xlink:type="arc" xlink:from="A" xlink:to="C" xlink:title="owns" />
  <arc xlink:type="arc" xlink:from="A" xlink:to="B" xlink:title="friend" />
  <arc xlink:type="arc" xlink:from="B" xlink:to="A" xlink:title="friend" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="Chevy 69" xlink:label="F" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="gasoline" xlink:label="G" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="V8" xlink:label="H" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="bomb" xlink:label="I" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="parliament" xlink:label="J" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="Chili" xlink:label="K" />
</contextLink>
```

This example shows two people being linked to different types of locators. Besides the two people we have a car, a book and an e-mail.

These items are linked together by a number of arcs. One to pay special attention to is the arcs between A and B – that is a binary link is modelled by two arcs.

Finally we have a number of res for defining the digital context of the link structure. In this example none of the res are linked to any other item.

### 3.5 User Presentation

The user presentation is not a simple task as data can be presented to the user in many different ways. How it should be done depends on what kind of data we are looking at and what kind of information the user is looking for. In some cases it would be good to look at the data through a spectrogram, to see the intensity of data and get an easy view on relevance of a large amount of data. In other you want to look at how a given item relates to other items.

To help clarifying how data could be presented to the user I will use a prototype approach to see what works and what does not work. In chapter 4 I will describe the prototype I have come up with and a typical use-case scenario.
4 Prototype

To clarify some of the possibilities for designing a user application for viewing the link structures I have designed a prototype. In this chapter I will give a description of the prototype. I will cover the internal design of the application, screen shots and a couple of code samples for the more interesting code parts. The entire source code will be available in Appendix C and also as source files.

For simplicity I have chosen to hardcode the data presented in the prototype. However, to keep the structure of the program intact, it is done in the XMLDecoder module, as it is the module, which handles decoding of Link Structure documents.

4.1 Introduction

I have chosen to develop the prototype using Delphi 7 for Windows. Delphi provides a standard framework for designing Windows applications so a lot of the code for creating windows, buttons etc. are given. Using Delphi all you basically have to do is to write the business logic to manage your data, and of course if you need some special components that do not come with Delphi by default. Many of these are available from the Internet though and often as freeware with source code.

The purpose of this prototype is to give a few ideas of how data can be presented to the end user. For this reason, I have chosen to base the view on hard coded data. I.e. the application does not read any real XML documents or data to generate its view.

4.2 Internal Design

The prototype application is designed in accordance with Object Oriented (OO) principles. I have divided the source code into logical units encapsulating functionality logical for the given type of module.

In the following I will describe each module by its functionality and the motivation for creating the module. See figure 5 for a view on the internal application structure.
Presentation Manager

The Presentation Manager (PM) is the application itself. This module encapsulates all the visual components of the application's main window except the Data Viewers. From this module you can invoke the other modules. This invocation happens behind the scenes and none of the modules invoked by the PM are visual to the user, only their output.

The Presentation Manager does not directly use the XML Decoder and the Data Viewers - instead the Data- and View Managers invoke these.

Data Manager

The Data Manager’s job is to populate the treeview with items.

The Data Manager analyzes the Link Structures (3.4.2) and generates the treeview on that basis. To make this analysis most effective it should not be necessary to look through the entire lot of link files and thus there should be some sort of naming convention to make it easier for the Data Manager to find which link documents to use in a given scenario.

View Managers

The View Managers jobs are to present the data collected by the Data Manager to the user.
A View Manager is really an encapsulation of the logic for generating the selected view based on a given data item. In the prototype we have two View Managers – Content View Manager and Histogram View Manager.

The Content View Manager displays how items and link documents are connected. Besides from seeing the connections, one can also see the content of a given item. Ideally the user should be able to follow a given item to see which items it is connected to. Of course this functionality also depends on the link analyser to find all items relevant to a given item.

The Histogram View Manager is more a “in the big picture” module. It gives you an overview of a bunch of link documents. The size of a pin for a given document will depend on how relevant it seems to be. What determines the relevance for a document can be pretty tough to determine, but ways of doing it could be based on how many items a document links to, how many context words exists etc. For simplicity I have chosen to base the prototype on number of items in a given link document.

**XML Decoder**

The XML Decoder’s job is to decode the link documents so we only need to make that logic once.

When the XML Decoder is created, it is given a Link Structure document. The document is decoded and all the data can be extracted to display to user. All data from the document can be extracted.

**Data Viewers**

The Data Viewers jobs are to present data to the user.

Obviously, since we have many different types of data, a number of Data Viewers are needed. We need viewers for e.g. E-Mail, books, cars and whatever piece of data we may come up with. To make the application more robust towards new data types it would be an advantage making a viewer framework that automatically can determine which Data Viewer to use based on the given content of the data file. However, for simplicity, I have chosen not to implement the framework in the prototype but it should be considered for a more comprehensive version of the Presentation Manager.
4.3 Application Design

There are many ways of designing the user interface of an application but also there are some guidelines on how to do it. The prototype I have developed is targeted for the Microsoft Windows platform and thus I have followed some of the basic design principles for this. See figure 6 for a view on what the application looks like.

In figure 6 you see the Presentation Manager (PM) in Content View mode. For this prototype I have chosen to implement two different ways of presenting the link structures to the user – Content View and Histogram. Further down I will explain these two modes in details.

The treeview to the left shows a list of link structures found in the database. These link structures are grouped together based on their given type (i.e. Persons, E-mails and Clusters).

Just above the treeview is a dropdown box where the user can choose which kind of view he/she wants. As mentioned above the prototype supports two kind of views (Content and Histogram), which both will be discussed further down.
To the right is a pane where the selected item from the treeview will be visualized. Of course this visualization depends on what type of view has been previously selected.

When a new view mode is chosen the entire tree will be reloaded, as it may not necessarily be the same tree for each view.

### 4.3.1 Content View

![Content View](image)

**Figure 7 - Presentation Manager - Content View**

In Content View mode you see a visualization of the selected link structure. The example in figure 7 is a quite simple one. Needless to say, the tree can grow quite big depending on the load of information and the granularity of the link structure files, or clusters as they may be called. However, we already have a cluster type, which covers a collection of similar items – that is items sharing the same digital context.

To simplify a complex view containing many items it would be beneficial to have some sort of clustering algorithm to group similar items based on their type. This type of cluster, however, should not be confused with the one mentioned above.

Also in the Content View colouring the items would be beneficial to the user, just as for the histograms. Different types of items can have different colours and their
reliability depends on the brightness of their colour – the brighter the colour the more important the item is. This reliability will be based on how many of the context words concerns the same topics. The more words for a topic the more reliable the item is – i.e. a few topics and many words are better than many topics with a few words.

If you place the cursor above the view (i.e. not above any of the nodes) you will see a popup showing you a description of what you are currently looking at. E.g. that could be the topic of a cluster.

If you double click on an item you see the content of that given item. I.e. for a person you will see all the information stored about this person, and for an e-mail you will see the content of the e-mail and the digital context as well.

If you right click on an item you will get a small popup menu where you can select to View Details (equivalent to double clicking an item) or to follow the item. If you choose to follow an item you will see a new view similar to the one you left showing all the items contained by this item.

4.3.2 Histogram

![Histogram](Image)

*Figure 8 - Presentation Manager – Histograms*
In the Histogram view mode you will see a statistical view of the selected item. What the statistics are based upon depends on the settings you have chosen by right clicking on the background (i.e. not on one of the pins). You can select between three different modes of statistics:

- **No Keywords**
  This one bases the statistics on how many keywords are located in a given item. A keyword is typically entered by the user but may also be set as a system wide configuration.

- **No Items**
  This one bases the statistics on how many subitems a given item contains. That is the number of items it links.

- **No Context Words**
  This one bases the statistics on how many context words exist for the items contained.

Additional modes can be added, but I have chosen these as the most relevant ones for the prototype. In general the higher the pin is the more relevant it is in the current view.

But also the colour of the pins means something. The brighter the colour is the more reliable the item is. This reliability is purely based on the context words – both on how many but also on the variety of these. The more context words that can be grouped based a shared topic, the more reliable an item is, just as for the Content View.

If you place the cursor above the view (i.e. not above any of the pins) you will see a popup showing you a description of what you are currently looking at. E.g. that could be the topic of a cluster.

If you double click on an item you see the content of that given item. So for a person you will see all the information stored about this person, and for an e-mail you will see the content of the e-mail and the digital context as well.

If you right click on an item you will get a small popup menu where you can select to View Details (equivalent to double clicking an item) or to follow the item. If you choose to follow an item you will see a new view similar to the one you left showing all the items, which the item you followed links to.
4.3.3 The Menus

The menus in the prototype are pretty simple. You have only two menus:

- File
  From here you can close the application.

- Help
  From here you can open an “About box” showing version info of the application.

Of course there should be more items in the menus, but it was not necessary for the prototype and thus I have not focused on that. However, it would be convenient to able to save pictures of views etc. from the menus.

4.4 Description of the source code

As you can see in appendix C there is plenty of source code for this small project. In this part I will pick out the more interesting parts of the source code for a closer description.

4.4.1 UContentViewMan.pas

UContentViewMan handles all the drawing in the Content View mode.

4.4.1.1 DrawGraph

The DrawGraph method handles all the drawing of the graph in the window. It is a pretty simple method but still it is quite important for the system.

This method takes the path to a XML document as a parameter. This XML must conform to the XSD for Link Structures (Appendix A)

```
procedure TContentViewManager.DrawGraph(pXMLDoc: string);
var
  lXMLDec : TXMLDecoder;
  li : integer;
  lSource, lDest : TGraphNode;
  lLink : PLinkStructure;
  lgl : TGraphLink;
begin
  First we will create an XML Decoder. The XML decoder takes the XML document passed to DrawGraph and decodes it.
  lXMLDec := TXMLDecoder.Create(pXMLDoc);
  try
    ClearView();
    fGraph.Hint := lXMLDec.Descriptor;
    // Fill in data from the XML document
    li := 0;
```
To extract all the items and create nodes and links, we iterate through the list of links found in the XML document.

```pascal
while (li < lXMLDec.LinkCount) do
begin
  // Show links on the graph
  lLink := lXMLDec.Links[li];
  if Assigned(lLink) then
  begin
    // Get the nodes, if they are already created
    lSource := FindNode(lLink.source.HREF);
    lDest := FindNode(lLink.destination.HREF);
    if lSource = nil then
    begin
      lSource := fGraph.InsertNode(FindPosition());
      lSource.Text := lLink.source.HREF;
      if lDest = nil then
      begin
        lDest := fGraph.InsertNode(FindPosition());
        lDest.Text := lLink.destination.HREF;
      end;
    end;
    // Finally link them together
    lgl := fGraph.LinkNodes(lSource, lDest);
    lgl.Text := lLink.linkname;
    lgl.Kind := lkDirected;
  end;
  inc(li);
end;
```

In order not to get several copies of the same node we start searching for it. If FindNode does not find the node it returns a null pointer (nil in Delphi). FindNode looks for the HREF part of the link.

```pascal
// Get the nodes, if they are already created
lSource := FindNode(lLink.source.HREF);
lDest := FindNode(lLink.destination.HREF);
```

If any of the nodes were not found we must create them. To get a position to place the node at we use FindPosition.

```pascal
// Create the non-existing nodes
if lSource = nil then
begin
  lSource := fGraph.InsertNode(FindPosition());
lSource.Text := lLink.source.HREF;
  if lDest = nil then
  begin
    lDest := fGraph.InsertNode(FindPosition());
lDest.Text := lLink.destination.HREF;
  end;
end;
```

Finally we link the nodes and continue to the next link item.

```pascal
// Finally link them together
lgl := fGraph.LinkNodes(lSource, lDest);
lgl.Text := lLink.linkname;
lgl.Kind := lkDirected;
// finally increase the counter
inc(li);
end;
finally
lXMLDec.Free();
end;
end;
```

### 4.4.1.2 FindNode

The FindNode method searches the entire graph for the node given by pNodeName. It stops the search either if it finds the node or if it reaches the end. If it does not find the node a null pointer (nil in Delphi) is returned.

```pascal
function TContentViewManager.FindNode(pNodeName: string): TGraphNode;
var
  lFound : boolean;
  lcount, li : integer;
begin
  lFound := false;
  li := 0;
```
1: count := fGraph.ObjectsCount();
while (not lFound) and (li < lcount) do
begin
  if fGraph.Objects[li] is TGraphNode then
    lFound := (fGraph.Objects[li].Text = pNodeName);
  inc(li);
end;
if lFound then
  Result := fGraph.Objects[li-1] as TGraphNode
else
  Result := nil;
end;

4.4.1.3 FindPosition

The FindPosition method finds a free spot on the graph for a new item. The implementation of this one is fairly simple as it generates random positions from time to time. It does not take into consideration if a node already fills that space.

function TContentViewManager.FindPosition: PRect;
begin
  New(Result);
  Result.Left := Random(fGraph.Width - 50);
  Result.Right := Result.Left + 50;
  Result.Top := Random(fGraph.Height - 50);
  Result.Bottom := Result.Top + 50;
end;

4.4.2 UDataManager.pas

UDataManager handles loading data into the treeview depending on the view selected.

4.4.2.1 LoadTree

LoadTree takes a pViewType as a parameter defining which type of view we want to load data for. As for now the valid types are Content and Histogram. Depending on the type of view either GenerateContent or GenerateHistogram is invoked.

procedure TDataManager.LoadTree(pViewType: TViewType);
begin
  // First clear data
  ClearTreeView();

  // Load new data
  case pViewType of
    vtContent :
      GenerateContent();
    end;
    vtHistogram :
      GenerateHistogram();
    end;
end;
4.4.2.2 ClearTreeView

The ClearTreeView method frees all the memory allocated for the items in the treeview. For each treenode an extra piece of memory is allocated containing what type of node it is and a pointer to its XML file.

```pascal
procedure TDataManager.ClearTreeView;
var
  li : integer;
begin
  // Free memory
  for li := 0 to fTree.Count-1 do
    if Assigned(fTree[li].Data) then
      begin
        Dispose(fTree[li].Data);
        fTree[li].Data := nil;
      end;
  // Clear treeview
  fTree.Clear();
end;
```

4.4.3 UHistogramViewMan.pas

UHistogramViewMan handles all the drawing of the histogram.

4.4.3.1 DrawHistogram

The DrawHistogram method handles drawing of the histogram. The DrawHistogram method itself is quite simple as it only draws the histogram. It invokes the more complicated CalculateHistogram, which will be described later.

```pascal
procedure THistogramViewManager.DrawHistogram;
var
  lXMLDec : TXMLDecoder;
  li : integer;
  lWidth, lHeight : integer;
  lpr : PRect;
  lgn : TGraphNode;
begin
  lXMLDec := TXMLDecoder.Create(pXMLDoc);
  try
    ClearView();
    CalculateHistogram();
    // Draw the histogram
    lWidth := fGraph.Width div fHistogram.Count;
    lHeight := fGraph.Height div 100;
    for li := 0 to fHistogram.Count-1 do
      begin
        new(lpr);
        try
          lpr.Left := lWidth*li;
          lpr.Right := lpr.Left + lWidth;
          lpr.Bottom := fGraph.Height;
          lpr.Top := lpr.Bottom - (Integer(fHistogram[li])*lHeight);
          lgn := fGraph.InsertNode(lpr);
          lgn.Brush.Color := Random(1000);
        finally
          Dispose(lpr);
        end;
      end;
  finally
    end;
  end;
end;
```
4.4.3.2 CalculateHistogram

The CalculateHistogram is the heavy-duty method in the UHistogramViewMan. In this prototype, though, it is quite simple as we generate random data. In a real solution it would be much more complicated as it would have to calculate the statistics for all the items selected. Some of its functionality may be exported to a server, as it may require a lot of computational power and a lot of disk I/O.

```delphi
procedure THistogramViewManager.CalculateHistogram;
var
  li : integer;
begin
  // Clear existing histogram
  fHistogram.Clear;

  // Fill in new data
  // First calc no of items, 5-25
  li := Random(20) + 5;
  // Fill in new items
  while (li > 0) do
    begin
      fHistogram.Add(Pointer(Random(100)));
      dec(li);
    end;
end;
```

4.4.4 UXMLDecoder.pas

UXMLDecoder handles all the decoding of Link Structure XML documents conforming to the Link Structures XSD described in appendix A.

4.4.4.1 DecodeXML

The DecodeXML method is the method that decodes the actual XML document. This one is not very interesting as we are working with hard coded data, but in the real version there would be a lot of XML decoding to build the link structures to display to the user. This one is simpler than the one in the source code.

```delphi
procedure TXMLDecoder.DecodeXML;
var
  ll : PLinkStructure;
begin
  fDescriptor := 'This is a bunch of test data';
  new(ll);
  ll.source.HREF := 'Sadam';
  ll.source.Caption := 'Iraq';
  ll.destination.HREF := 'EMail1';
  ll.destination.Caption := 'Bomb';
  ll.linkname := 'has send';
  fLinks.Add(ll);
end;
```
5 Conclusion

5.1 What is the context of digital data and how to model this?

When I in this paper discuss the context of digital data, I do not discuss the physical environment in which the data resides. Instead I discuss the digital context – the topic of a given piece of digital data. In other systems, discussed in related work (2.1), the context of digital data is defined by physical positions.

When I have defined the digital context, the next problem is how to determine the digital context of a given piece of digital data. There are several ways of determining the digital context, but the two most well known ways are taxonomy and clustering.

The taxonomy is about finding words from data, which helps defining the topic of that specific piece of data. Taxonomy is also called classification. Basically it is about filtering out commonly used words such as and, or, yes, no etc., and leave the describing words that can define the topic.

Clustering is a bit like taxonomy on some parts but is in general terms it is about grouping data based on their topics. These topics can typically be found by taxonomy, or simply by comparing how many similar words exists in the data.

When the digital context has been determined it should also be stored, so we do not have to determine it each time we look at the given data. Needless to say we will meet many different types of data and thus we must have a dynamic storage, which can easily be adjusted to new types of data. For this purpose XML will suit just fine as XML is a very dynamic language with respect to what types of data it can contain and how easy it is to change the content of a given XML Schema. Every XML document should be based on a XML Schema (XSD) so it can be verified if the document is correct.

During the analysis of related work, I found that several products exist capable of both taxonomy and clustering. For that reason I would recommend using one of these for deriving the digital context – the IDOL server in particular.

As for the model of the digital context I have found a context model defined in Appendix A and is discussed in 3.4.1.
5.2 Which type of links can be used to build a context based link structure and how can this be modelled?

The context based link structure is what bind the different context models together to form a greater perspective on the data at hand. The link structure will link the data based on shared context and/or attributes.

When deciding which type of links to use, there are different issues to take into consideration. For one, we have to deal with a dynamic load of data – i.e. we do not know from time to time how many items should be linked. For now we only need to make link between files. However, later on, one could imagine it would be required to able to make links to specific items inside the context model.

For these reasons, I have chosen to base the link structures on XLink. XLink is developed by W3C and is also based on XML. By using XLink it is rather easy to enhance and change the link structures over time, and if the XLink readers are made correctly, changing none or just a little source code can even do it. It would be easy to link into documents using XPointer.

Based on XLink, I have defined a link structure model. This link structure model can be found in appendix A and is discussed in 3.4.2.

5.3 How can context based link structures be presented to the end-user?

When we want to present the link structures to the end user there are numerous ways to do this, but no exact science saying which one would be the correct way to do it. To help finding ways to present the link structures to the user I choose to develop a small prototype to try out a couple of these (4).

To present the link structures to choose from I have used a treeview, and grouped the nodes by their data type. Treeviews are widely used in Microsoft Windows and gives a good view on which items are available.

To select what type of view to see I have choosen a drop-down box just above the treeview, where the user can select the type of view he/she wants. Using a tab sheet where the user could select the type of view by tabbing between pages could also have solved this. I chose the drop-down box because I found it was more intuitive to the user and because you may not necessarily be able to do that same kind of analysis on all sorts of data.

To present the link structures I have implemented a content view and a histogram.
In the Content View you see all items contained by a single link structure. It would be a nice thing though if you could select several link structures and see how they relate to each other. You may also follow an item and see how that single items link to other items, or you can view the content of that given item.

In the Histogram you get a statistical view on the data based on what kind of information you want to base the statistics on. You can figure the importance of a item in two ways – either by how tall of the pin is or by the brightness of the colour. The taller the pin is the more relevant it is in respect to the kind of statistics chosen, and the brighter the colour is the more reliable the item is with respect to its context.

Determining the reliability of an item is not an easy task to do, even though it might seem so. The reliability is based on how many context words an item contain and how well these words fit together. An item may have many words, but if they have no correlation the item is not very reliable as it would not really say anything about the real topic of the item – the same count for the opposite where we have a few words but with stronger correlation. Of course, the number of context words should be compared with the number words in the item.

### 5.4 Overall conclusion

Based on the conclusions in 5.1-5.3, the overall conclusion will be that is definitely possible to develop a system for doing successful data mining using context aware hypermedia based on the analysis I have stated during this paper.

It also shows that it may not be feasible to develop your own analyser for taxonomy and clustering as good ones, such as IDOL, are already in the market. Price may be an issue on these though, but would still be more feasible compared to developing your own.

For the context models and link structures XML and XLink seems to be fulfil the task representing the context- and link models. One may consider enhancing the XML a bit show what specifically bind two items together.

The user presentation, however, needs some work to find good ways of displaying link structures to the user. I have come with a couple of examples, but more ways of presenting the data should be developed. However, the views I have come up with seems to work pretty good even though they are not fully developed.
Appendix A. XML Schemas

Context Model

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xsd:element name="context" type="ContextModel" />
  <xsd:complexType name="ContextModel">
    <xsd:choice>
      <xsd:element name="person" type="personType" />
      <xsd:element name="e-mail" type="e-mailType" />
      <xsd:element name="chat" type="chatType" />
      <xsd:element name="car" type="carType" />
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="nodeType">
    <xsd:sequence>
      <xsd:element name="Property" type="propertyType" minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="ContextWord" type="xsd:string" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID" use="required" />
  </xsd:complexType>
  <xsd:complexType name="physicalItemType">
    <xsd:complexContent>
      <xsd:extension base="nodeType">
        <xsd:sequence>
          <xsd:element name="RFID" type="xsd:string" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="personType">
    <xsd:complexContent>
      <xsd:extension base="nodeType">
        <xsd:sequence>
          <xsd:element name="Name" type="xsd:string" />
          <xsd:element name="SocialSecurityNo" type="xsd:string" />
          <xsd:element name="PostalAddress" type="xsd:string" />
          <xsd:element name="Country" type="xsd:string" />
          <xsd:element name="E-mail" type="xsd:string" />
          <xsd:element name="Sex" type="sexType" />
          <xsd:element name="Age" type="xsd:positiveInteger" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="e-mailType">
    <xsd:complexContent>
      <xsd:extension base="nodeType">
        <xsd:sequence>
          <xsd:element name="Subject" type="xsd:string" />
          <xsd:element name="Sender" type="xsd:string" />
          <xsd:element name="Recipient" type="xsd:string" minOccurs="0" maxOccurs="unbounded" />
          <xsd:element name="Attachment" type="xsd:string" minOccurs="0" maxOccurs="unbounded" />
          <xsd:element name="Protocol" type="mailProtocolType" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="chatType">
    <xsd:complexContent>
      <xsd:extension base="nodeType">
        <xsd:sequence>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
<xsd:element name="ChatRoom" type="xsd:string" />
<xsd:element name="ChatType" type="xsd:string" />
<xsd:element name="URI" type="xsd:string" />
<xsd:element name="Participant" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="carType">
<xsd:complexContent>
<xsd:extension base="physicalItemType">
<xsd:sequence>
<xsd:element name="Brand" type="xsd:string" />
<xsd:element name="Model" type="xsd:string" />
<xsd:element name="LicensePlate" type="xsd:string" />
<xsd:element name="SerialNo" type="xsd:string" />
<xsd:element name="NoSeats" type="xsd:integer" />
<xsd:element name="EngineType" type="engineType"/>
<xsd:element name="EngineSizeCubic" type="xsd:positiveInteger" />
<xsd:element name="NoWheels" type="xsd:unsignedByte"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="propertyType">
<xsd:sequence>
<xsd:element name="Key" type="xsd:integer" />
<xsd:element name="Value" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="engineType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="Petrol"/>
<xsd:enumeration value="Diesel"/>
<xsd:enumeration value="Gas"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="sexType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="Male"/>
<xsd:enumeration value="Female"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="mailProtocolType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="POP3"/>
<xsd:enumeration value="IMAP"/>
<xsd:enumeration value="WEBMAIL"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>
Link Structure

<?xml version="1.0"?>


<xsd:element name="contextLink" type="xsd:string" />
<xsd:simpleType name="Identifier">
<xsd:restriction base="xsd:ID">
<xsd:minLength value="1" />
</xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="Locator">
<xsd:attribute ref="xlink:type" fixed="locator"/>
<xsd:attribute ref="xlink:href" />
<xsd:attribute ref="xlink:role" />
<xsd:attribute ref="xlink:label" />
</xsd:complexType>

<xsd:complexType name="Arc">
<xsd:attribute ref="xlink:type" fixed="arc"/>
<xsd:attribute ref="xlink:from" />
<xsd:attribute ref="xlink:to" />
<xsd:attribute ref="xlink:title" />
</xsd:complexType>

<xsd:complexType name="Resource">
<xsd:attribute ref="xlink:type" fixed="resource"/>
<xsd:attribute ref="xlink:role" />
<xsd:attribute ref="xlink:title" />
<xsd:attribute ref="xlink:label" />
</xsd:complexType>

<xsd:complexType name="LinkStructure">
<xsd:sequence>
<xsd:element name="loc" type="Locator" minOccurs="1" maxOccurs="unbounded"/>
<xsd:element name="arc" type="Arc" minOccurs="0" maxOccurs="unbounded"/>
<xsd:element name="res" type="Resource" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
<xsd:attribute name="id" type="Identifier" />
<xsd:attribute ref="xlink:type" fixed="extended" />
<xsd:attribute ref="xlink:title" />
</xsd:complexType>
</xsd:schema>
Appendix B. XML Document Examples

**Context Model**

**Person**
```xml
<?xml version="1.0"?>
<person xsi:schemaLocation="ContextModel.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="123">
  <ContextWord>writer</ContextWord>
  <Name>Peter Hansen</Name>
  <SocialSecurityNo>2412351</SocialSecurityNo>
  <PostalAddress>Bytoften 1, 9999 Fantasi Town</PostalAddress>
  <Country>Denmark</Country>
  <E-mail>ph@someserver.dk</E-mail>
</person>
```

**Car**
```xml
<?xml version="1.0"?>
<car xsi:schemaLocation="ContextModel.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="2424">
  <ContextWord>victim's</ContextWord>
  <RFID>3r5jkg5sd</RFID>
  <Brand>BMW</Brand>
  <Model>E46 M3</Model>
  <LicensePlate>KD 3533 DK33</LicensePlate>
  <SerialNo>333 EGEDG 2324 D</SerialNo>
  <NoSeats>5</NoSeats>
  <EngineType>Petrol</EngineType>
  <EngineSizeCubic>3.200</EngineSizeCubic>
  <NoWheels>4</NoWheels>
</car>
```

**Link Structure**

**Example**
```xml
<?xml version="1.0"?>
<contextLink xmlns:xlink="http://www.w3.org/1999/xlink" id="2" xlink:type="extended" xlink:title="Test">
  <loc xlink:type="locator" xlink:href="file://2.xml" xlink:role="role://person" xlink:label="B" />
  <loc xlink:type="locator" xlink:href="file://4.xml" xlink:role="role://e-mail" xlink:label="D" />
  <arc xlink:type="arc" xlink:from="A" xlink:to="D" xlink:title="has sent" />
  <arc xlink:type="arc" xlink:from="D" xlink:to="B" xlink:title="is delivered to" />
  <arc xlink:type="arc" xlink:from="B" xlink:to="C" xlink:title="owns" />
  <arc xlink:type="arc" xlink:from="E" xlink:to="C" xlink:title="reads" />
  <arc xlink:type="arc" xlink:from="A" xlink:to="B" xlink:title="friend" />
  <arc xlink:type="arc" xlink:from="B" xlink:to="A" xlink:title="friend" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="Chevy 69" xlink:label="F" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="gasoline" xlink:label="G" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="V8" xlink:label="H" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="bomb" xlink:label="I" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="parliament" xlink:label="J" />
  <res xlink:type="resource" xlink:role="role://contextword" xlink:title="Chili" xlink:label="K" />
</contextLink>
```
Appendix C. Source Code

PresentationManager.dpr

program PresentationManager;

uses
  Forms,
  UMain in 'UMain.pas' {frmMain},
  UDataManager in 'UDataManager.pas',
  UContentViewMan in 'UContentViewMan.pas',
  UXMLDecoder in 'UXMLDecoder.pas',
  UHistogramViewMan in 'UHistogramViewMan.pas',
  UAbout in 'UAbout.pas' {AboutBox};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TfrmMain, frmMain);
  Application.CreateForm(TAboutBox, AboutBox);
  Application.Run;
end.

UAbout.pas

unit UAbout;

interface

uses Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls;

type
  TAboutBox = class(TForm)
    Panel1: TPanel;
    ProgramIcon: TImage;
    ProductName: TLabel;
    Version: TLabel;
    Copyright: TLabel;
    Comments: TLabel;
    OKButton: TButton;
  end;

var
  AboutBox: TAboutBox;

implementation

{$R *.dfm}

end.

UContentViewMan.pas

unit UContentViewMan;

interface

uses Windows, SimpleGraph, UXMLDecoder, Graphics, Dialogs;

type
  TContentViewManager = class
    private
      fGraph : TSimpleGraph;
      function FindNode(pNodeName: string): TGraphNode;
      function FindPosition: PRect;
    public
      end;

Leif Bredgaard Andersen  45/60  25/05/2005
constructor Create(pGraph : TSimpleGraph);
destructor Destroy; override;
procedure DrawGraph(pXMLDoc : string);
procedure ClearView;
procedure OpenItem(pObject: TGraphObject);
procedure FollowItem(pObject : TGraphObject);
end;

implementation

(TViewManager)

//*****************************************************************************/
constructor TContentViewManager.Create(pGraph: TSimpleGraph);
begin
  inherited Create();
  fGraph := pGraph;
end;

//*****************************************************************************/
destructor TContentViewManager.Destroy;
begin
  fGraph.Clear();
  fGraph := nil;
  inherited;
end;

//*****************************************************************************/
// Draws a graph from the given XML file.
// Pre : pXMLDoc points to a valid XLink document describing a cluster of data.
// procedure TContentViewManager.DrawGraph(pXMLDoc: string);
var
  lXMLDec : TXMLDecoder;
  li : integer;
  lSource, lDest : TGraphNode;
  lLink : PLinkStructure;
  lgl : TGraphLink;
begin
  lXMLDec := TXMLDecoder.Create(pXMLDoc);
  try
    ClearView();
    fGraph.Hint := lXMLDec.Descriptor;
    // Fill in data from the XML document
    li := 0;
    while (li < lXMLDec.LinkCount) do
      begin
        lLink := lXMLDec.Links[li];
        if Assigned(lLink) then
          begin
            // Get the nodes, if they are already created
            lSource := FindNode(lLink.source.HREF);
            lDest := FindNode(lLink.destination.HREF);
            // Creat the non-existing nodes
            if lSource = nil then
              begin
                lSource := fGraph.InsertNode(FindPosition());
                lSource.Text := lLink.source.HREF;
              end;
            if lDest = nil then
              begin
                lDest := fGraph.InsertNode(FindPosition());
                lDest.Text := lLink.destination.HREF;
              end;
            // Finally link them together
          end;
        li := li + 1;
      end;
  except
    on E: exception do
      begin
        // Handle errors
      end;
  end;
end;
lgl := fGraph.LinkNodes(lSource, lDest);
lgl.Text := lLink.linkname;
lgl.Kind := lkDirected;
end;
// finally increase the counter
inc(li);
finally
lXMLDec.Free();
end;
end;

procedure TContentViewManager.ClearView;
begin
fGraph.Clear();
end;

procedure TContentViewManager.OpenItem(pObject: TGraphObject);
begin
ShowMessage('Here would come information about the selected item');
end;

function TContentViewManager.FindNode(pNodeName: string): TGraphNode;
var
lFound : boolean;
lcount, li : integer;
begin
lFound := false;
li := 0;
lcount := fGraph.ObjectsCount();
while (not lFound) and (li < lcount) do
begin
if fGraph.Objects[li] is TGraphNode then
begin
lFound := (fGraph.Objects[li].Text = pNodeName);
inc(li);
end;
if lFound then
Result := fGraph.Objects[li-1] as TGraphNode
else
Result := nil;
end;

function TContentViewManager.FindPosition: PRect;
begin
New(Result);
Result.Left := Random(fGraph.Width - 50);
Result.Right := Result.Left + 50;
Result.Top := Random(fGraph.Height - 50);
Result.Bottom := Result.Top + 50;
end;

procedure TContentViewManager.FollowItem(pObject: TGraphObject);
begin
ShowMessage('By following a item you would get a detail view of item, like the one you are looking at');
end;

function TContentViewManager.FindPosition: PRect;
begin
New(Result);
Result.Left := Random(fGraph.Width - 50);
Result.Right := Result.Left + 50;
Result.Top := Random(fGraph.Height - 50);
Result.Bottom := Result.Top + 50;
end;

procedure TContentViewManager.FollowItem(pObject: TGraphObject);
begin
ShowMessage('By following a item you would get a detail view of item, like the one you are looking at');
end;

UDataManager.pas
unit UDataManager;
interface

Leif Bredgaard Andersen 47/60 25/05/2005
uses
  Windows, Classes, ComCtrls, SysUtils;

type
  TViewType = (vtContent, vtHistogram);
  TNodeType = (ntFolder, ntCluster, ntPerson, ntEMail, ntCar, ntChat);

PNodeData = ^TNodeData;
TNodeData = record
  dwNodeType : TNodeType;
  sData : string;
end;

TDataManager = class
private
  fTree : TTreeNodes;
  procedure ClearTreeView;
  procedure GenerateContent;
  procedure GenerateHistogram;
public
  constructor Create(pTree : TTreeNodes);
  destructor Destroy; override;
  procedure LoadTree(pViewType : TViewType);
  procedure LoadSubitems(pNode : TTreeNode);
end;

implementation

{ TDataManager }

//*****************************************************************************/
constructor TDataManager.Create;
begin
  inherited Create();
  fTree := pTree;
end;

//*****************************************************************************/
destructor TDataManager.Destroy;
begin
  ClearTreeView();
  fTree := nil;
  inherited;
end;

//*****************************************************************************/
procedure TDataManager.LoadTree(pViewType: TViewType);
begin
  // First clear data
  ClearTreeView();

  // Load new data
  case pViewType of
    vtContent :
      begin
        GenerateContent();
      end;
    vtHistogram :
      begin
        GenerateHistogram();
      end;
  end;
end;
procedure TDataManager.ClearTreeView;
var
    li : integer;
begin
  // Free memory
  for li := 0 to fTree.Count-1 do
    if Assigned(fTree[li].Data) then
      begin
        Dispose(fTree[li].Data);
        fTree[li].Data := nil;
      end;

  // Clear treeview
  fTree.Clear();
end;

//****************************************************************************
procedure TDataManager.LoadSubitems(pNode: TTreeNode);
begin
  // First check if data are already loaded
  if pNode.HasChildren then
    if pNode.Item[0].Text <> 'DUMMY' then
      Exit;

  // Ok - data were not loaded - lets do it :)
end;
//****************************************************************************
procedure TDataManager.GenerateContent;
var
    ltn, ltn2 : TTreeNode;
begin
  // Add 'Persons' section
  ltn := fTree.Add(nil, 'Persons');
  ltn.ImageIndex := 0;
  ltn.SelectedIndex := 0;
  ltn.Data := new(PNodeData);
  TNodeData(ltn.Data^).sData := 'TestPersons';
  TNodeData(ltn.Data^).dwNodeType := ntFolder;
  // add anton
  ltn2 := fTree.AddChild(ltn, 'Anton');
  ltn2.Data := new(PNodeData);
  ltn2.ImageIndex := 3;
  ltn2.SelectedIndex := 3;
  TNodeData(ltn2.Data^).sData := 'TestPerson';
  TNodeData(ltn2.Data^).dwNodeType := ntPerson;

  // add kimberly
  ltn2 := fTree.AddChild(ltn, 'Kimberly');
  ltn2.Data := new(PNodeData);
  ltn2.ImageIndex := 3;
  ltn2.SelectedIndex := 3;
  TNodeData(ltn2.Data^).sData := 'TestPerson';
  TNodeData(ltn2.Data^).dwNodeType := ntPerson;

  // add sadam
  ltn2 := fTree.AddChild(ltn, 'Sadam');
  ltn2.Data := new(PNodeData);
  ltn2.ImageIndex := 3;
  ltn2.SelectedIndex := 3;
  TNodeData(ltn2.Data^).sData := 'TestPerson';
  TNodeData(ltn2.Data^).dwNodeType := ntPerson;

  // Add 'E-mails' section
  ltn := fTree.Add(nil, 'E-mails');
  ltn.Data := new(PNodeData);
  ltn.ImageIndex := 1;
  ltn.SelectedIndex := 1;
  TNodeData(ltn.Data^).sData := 'TestEMails';
  TNodeData(ltn.Data^).dwNodeType := ntFolder;

  // add mail 1
lt2 := fTree.AddChild(ltn, 'Mail 1');
lt2.Data := new(PNodeData);
lt2.ImageIndex := 4;
lt2.SelectedIndex := 4;
TNodeData(lt2.Data^).sData := 'TestEMail';
TNodeData(lt2.Data^).dwNodeType := ntEMail;
// add mail 2
lt2 := fTree.AddChild(ltn, 'Mail 2');
lt2.Data := new(PNodeData);
lt2.ImageIndex := 4;
lt2.SelectedIndex := 4;
TNodeData(lt2.Data^).sData := 'TestEMail';
TNodeData(lt2.Data^).dwNodeType := ntEMail;
// add mail 3
lt2 := fTree.AddChild(ltn, 'Mail 3');
lt2.Data := new(PNodeData);
lt2.ImageIndex := 4;
lt2.SelectedIndex := 4;
TNodeData(lt2.Data^).sData := 'TestEMail';
TNodeData(lt2.Data^).dwNodeType := ntEMail;
// Add 'Clusters' section
lt := fTree.Add(nil, 'Clusters');
lt.Data := new(PNodeData);
lt.ImageIndex := 2;
lt.SelectedIndex := 2;
TNodeData(lt.Data^).sData := 'TestClusters';
TNodeData(lt.Data^).dwNodeType := ntFolder;
// add cluster 1
lt2 := fTree.AddChild(lt, 'Cluster 1');
lt2.Data := new(PNodeData);
lt2.ImageIndex := 5;
lt2.SelectedIndex := 5;
TNodeData(lt2.Data^).sData := 'TestCluster';
TNodeData(lt2.Data^).dwNodeType := ntCluster;
// add cluster 2
lt2 := fTree.AddChild(lt, 'Cluster 2');
lt2.Data := new(PNodeData);
lt2.ImageIndex := 5;
lt2.SelectedIndex := 5;
TNodeData(lt2.Data^).sData := 'TestCluster';
TNodeData(lt2.Data^).dwNodeType := ntCluster;
// add cluster 3
lt2 := fTree.AddChild(lt, 'Cluster 3');
lt2.Data := new(PNodeData);
lt2.ImageIndex := 5;
lt2.SelectedIndex := 5;
TNodeData(lt2.Data^).sData := 'TestCluster';
TNodeData(lt2.Data^).dwNodeType := ntCluster;
end;

//******************************************************************************
procedure TDataManager.GenerateHistogram;
var
  ltn, lt2 : TTreeNode;
begin
  // Add 'Persons' section
  ltn := fTree.Add(nil, 'Persons');
  ltn.Data := new(PNodeData);
  ltn.ImageIndex := 0;
  ltn.SelectedIndex := 0;
  TNodeData(ltn.Data^).sData := 'TestHistPersons';
  TNodeData(ltn.Data^).dwNodeType := ntFolder;
  // add anton
  lt2 := fTree.AddChild(ltn, 'Anton');
  lt2.Data := new(PNodeData);
  lt2.ImageIndex := 3;
  lt2.SelectedIndex := 3;
  TNodeData(lt2.Data^).sData := 'TestHistPerson';
TNodeData(ltn.Data^).dwNodeType := ntPerson;
// add kimberly
ltn2 := fTree.AddChild(ltn, 'Kimberly');
ltn2.Data := new(PNodeData);
ltn2.ImageIndex := 3;
ltn2.SelectedIndex := 3;
TNodeData(ltn2.Data^).sData := 'TestHistPerson';
TNodeData(ltn2.Data^).dwNodeType := ntPerson;

// add sadam
ltn2 := fTree.AddChild(ltn, 'Sadam');
ltn2.Data := new(PNodeData);
ltn2.ImageIndex := 3;
ltn2.SelectedIndex := 3;
TNodeData(ltn2.Data^).sData := 'TestHistPerson';
TNodeData(ltn2.Data^).dwNodeType := ntPerson;

// Add 'E-mails' section
ltn := fTree.Add(nil, 'E-mails');
ltn.Data := new(PNodeData);
ltn.ImageIndex := 1;
ltn.SelectedIndex := 1;
TNodeData(ltn.Data^).sData := 'TestHistEMails';
TNodeData(ltn.Data^).dwNodeType := ntFolder;
// add mail 1
ltn2 := fTree.AddChild(ltn, 'Mail 1');
ltn2.Data := new(PNodeData);
ltn2.ImageIndex := 4;
ltn2.SelectedIndex := 4;
TNodeData(ltn2.Data^).sData := 'TestHistEMail';
TNodeData(ltn2.Data^).dwNodeType := ntEMail;
// add mail 2
ltn2 := fTree.AddChild(ltn, 'Mail 2');
ltn2.Data := new(PNodeData);
ltn2.ImageIndex := 4;
ltn2.SelectedIndex := 4;
TNodeData(ltn2.Data^).sData := 'TestHistEMail';
TNodeData(ltn2.Data^).dwNodeType := ntEMail;
// add mail 3
ltn2 := fTree.AddChild(ltn, 'Mail 3');
ltn2.Data := new(PNodeData);
ltn2.ImageIndex := 4;
ltn2.SelectedIndex := 4;
TNodeData(ltn2.Data^).sData := 'TestHistEMail';
TNodeData(ltn2.Data^).dwNodeType := ntEMail;

// Add 'Clusters' section
ltn := fTree.Add(nil, 'Clusters');
ltn.Data := new(PNodeData);
ltn.ImageIndex := 2;
ltn.SelectedIndex := 2;
TNodeData(ltn.Data^).sData := 'TestHistClusters';
TNodeData(ltn.Data^).dwNodeType := ntFolder;
// add cluster 1
ltn2 := fTree.AddChild(ltn, 'Cluster 1');
ltn2.Data := new(PNodeData);
ltn2.ImageIndex := 5;
ltn2.SelectedIndex := 5;
TNodeData(ltn2.Data^).sData := 'TestHistCluster';
TNodeData(ltn2.Data^).dwNodeType := ntCluster;
// add cluster 2
ltn2 := fTree.AddChild(ltn, 'Cluster 2');
ltn2.Data := new(PNodeData);
ltn2.ImageIndex := 5;
ltn2.SelectedIndex := 5;
TNodeData(ltn2.Data^).sData := 'TestHistCluster';
TNodeData(ltn2.Data^).dwNodeType := ntCluster;
// add cluster 3
ltn2 := fTree.AddChild(ltn, 'Cluster 3');
ltn2.Data := new(PNodeData);
ltn2.ImageIndex := 5;
ltn2.SelectedIndex := 5;
TNodeData(ltn2.Data^).sData := 'TestHistCluster';
TNodeData(ltn2.Data^).dwNodeType := ntCluster;
end;

//*****************************************************************************/
end.

UHistogramViewMan.pas

unit UHistogramViewMan;
interface
uses
  Windows, Classes, SysUtils, UXMLDecoder, SimpleGraph, Dialogs, Graphics;

type
  THistogramViewManager = class
    private
      fGraph : TSimpleGraph;
      fHistogram : TList;
      procedure CalculateHistogram;
    public
      constructor Create(pGraph : TSimpleGraph);
      destructor Destroy; override;
      procedure DrawHistogram(pXMLDoc : string);
      procedure ClearView;
      procedure OpenItem(pObject: TGraphObject);
      procedure FollowItem(pObject : TGraphObject);
    end;
  
implementation

  { THistogramViewManager } 

//*****************************************************************************/
constructor THistogramViewManager.Create;
begin
  inherited Create();

  fGraph := pGraph;
  fHistogram := TList.Create();
end;
//*****************************************************************************/
destructor THistogramViewManager.Destroy;
begin
  fGraph.Clear();
  fGraph := nil;
  fHistogram.Clear();
  fHistogram.Free();

  inherited;
end;
//*****************************************************************************/
// Draws a histogram from the given XML file.
// Pre: pXMLDoc points to a valid XLink document describing a cluster of data.
procedure THistogramViewManager.DrawHistogram;
var
  lXMLDec : TXMLDecoder;
  li : integer;
  lWidth, lHeight : integer;
  lpr : PRect;
  lgn : TGraphNode;
begin
  lXMLDec := TXMLDecoder.Create(pXMLDoc);
  try

ClearView();
CalculateHistogram();

// Draw the histogram
lWidth := fGraph.Width div fHistogram.Count;
Height := fGraph.Height div 100;
for li := 0 to fHistogram.Count-1 do
begin
new(lpr);
try
lpr.Left := lWidth*li;
lpr.Right := lpr.Left + lWidth;
lpr.Bottom := fGraph.Height;
lpr.Top := fGraph.Height - (Integer(fHistogram[li])*Height);
lgn := fGraph.InsertNode(lpr);
lgn.Brush.Color := Random(100);
finally
Dispose(lpr);
end;
//TestData();
finally
lXMLDec.Free();
end;
end;

//******************************************************************************
procedure THistogramViewManager.ClearView;
begin
fGraph.Clear();
end;

//******************************************************************************
procedure THistogramViewManager.OpenItem(pObject: TGraphObject);
begin
ShowMessage('Here would come information about the selected item');
end;

//******************************************************************************
procedure THistogramViewManager.CalculateHistogram;
var
li : integer;
begin
// Clear existing histogram
fHistogram.Clear();

// Fill in new data
// First calc no of items, 5-25
li := Random(20) + 5;
// Fill in new items
while (li > 0) do
begin
fHistogram.Add(Pointer(Random(100)));
dec(li);
end;
end;

//******************************************************************************
procedure THistogramViewManager.FollowItem(pObject: TGraphObject);
begin
ShowMessage('By following a item you would get a detail view of item, like the one you are looking at');
end;

//******************************************************************************
UMain.pas

unit UMain;
interface

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, Menus, StdCtrls, ComCtrls, ExtCtrls, ImgList, SimpleGraph, XPMan,
UDataManager, UContentViewMan, UHistogramViewMan, UAbout;

type
TfrmMain = class(TForm)
  mmMenu: TMainMenu;
  miFile: TMenuItem;
  miClose: TMenuItem;
  miHelp: TMenuItem;
  miAbout: TMenuItem;
  sbStatus: TStatusBar;
  panTree: TPanel;
  tvView: TTreeView;
  split: TSplitter;
  pcViews: TPageControl;
  cbViews: TComboBox;
  ilView: TImageList;
  tsContentView: TTabSheet;
  sgGraph: TSimpleGraph;
  tsHistogram: TTabSheet;
  pmGraphObj: TPopupMenu;
  miContShowDetails: TMenuItem;
  sgHistogram: TSimpleGraph;
  pmHistogram: TPopupMenu;
  miHisShowDetails: TMenuItem;
  miConFollowitem: TMenuItem;
  miHistFollowitem: TMenuItem;
  fDataMan : TDataManager;
  fContent : TContentViewManager;
  fHistogram : THistogramViewManager;
private
  procedure ReInitializeView;
end;

var
  frmMain: TfrmMain;

implementation

($R *.dfm)

//*****************************************************************************/
procedure TfrmMain.FormCreate(Sender: TObject);
begin
  fDataMan := TDataManager.Create(tvView.Items);
  fContent := TContentViewManager.Create(sgGraph);
  fHistogram := THistogramViewManager.Create(sgHistogram);
  sgGraph.Color := clWindow;
end;
procedure TfrmMain.FormDestroy(Sender: TObject);
begin
  fDataMan.Free();
  fContent.Free();
  fHistogram.Free();
end;

procedure TfrmMain.miCloseClick(Sender: TObject);
begin
  Application.Terminate();
end;

procedure TfrmMain.sgGraphObjectDblClick(Graph: TSimpleGraph; GraphObject: TGraphObject);
begin
  fContent.OpenItem(GraphObject);
end;

procedure TfrmMain.cbViewsChange(Sender: TObject);
begin
  pcViews.ActivePageIndex := cbViews.ItemIndex;
  ReInitializeView();
  sbStatus.SimpleText := 'View changed to ' + cbViews.Text;
end;

procedure TfrmMain.ReInitializeView;
begin
  // Clear actual view
  case cbViews.ItemIndex of
    0 : // vtContent
      begin
        fContent.ClearView();
      end;
    1 : // vtHistogram
      begin
        fHistogram.ClearView();
      end;
  end;

  // Load tree
  fDataMan.LoadTree(TViewType(cbViews.ItemIndex));
  sbStatus.SimpleText := 'Data loaded';
end;

procedure TfrmMain.tvViewExpanding(Sender: TObject; Node: TTreeNode;
  var AllowExpansion: Boolean);
begin
  fDataMan.LoadSubitems(Node);
  AllowExpansion := true;
end;

procedure TfrmMain.tvViewChange(Sender: TObject; Node: TTreeNode);
begin
  case cbViews.ItemIndex of
    0 : // vtContent
      begin
        if Assigned(Node.Data) then
        begin
          fContent.DrawGraph(TNodeData(Node.Data^).sData);
        end;
      end;
    end;
end;
procedure TfrmMain.miContShowDetailsClick(Sender: TObject);
begin
    if sgGraph.SelectedObjects.Count = 1 then
        fContent.OpenItem(sgGraph.SelectedObjects[0]);
end;

procedure TfrmMain.miHisShowDetailsClick(Sender: TObject);
begin
    if sgHistogram.SelectedObjects.Count = 1 then
        fHistogram.OpenItem(sgHistogram.SelectedObjects[0]);
end;

procedure TfrmMain.sgHistogramObjectDblClick(Graph: TSimpleGraph; GraphObject: TGraphObject);
begin
    fHistogram.OpenItem(GraphObject);
end;

procedure TfrmMain.miAboutClick(Sender: TObject);
begin
    AboutBox.ShowModal();
end;

procedure TfrmMain.miConFollowItemClick(Sender: TObject);
begin
    if sgGraph.SelectedObjects.Count = 1 then
        fContent.FollowItem(sgGraph.SelectedObjects[0]);
end;

procedure TfrmMain.miHistFollowItemClick(Sender: TObject);
begin
    if sgHistogram.SelectedObjects.Count = 1 then
        fHistogram.FollowItem(sgHistogram.SelectedObjects[0]);
end;

end.

*UXMLDecoder.pas*

unit UXMLDecoder;

interface

uses
    Windows, Classes;

type
    TLink = record
        HREF : string;
        Caption : string;
    end;

    TLinkStructure = record
        source : TLink;
        destination : TLink;
    end;


TXMLDecoder = class
private
  fXMLDoc : string;
  fDescriptor : string;
  fDigitalContext : TStringList;
  fLinks : TList;
  procedure DecodeXML;
  procedure FreeLinks;
  function GetLinkStructure(index: longword): PLinkStructure;
  function GetLinkCount: longword;
public
  constructor Create(pXMLDoc : string);
  destructor Destroy; override;
  property XMLDoc : string read fXMLDoc;
  property Descriptor : string read fDescriptor;
  property DigitalContext : TStringList read fDigitalContext;
  property LinkCount : longword read GetLinkCount;
  property Links[index : longword] : PLinkStructure read GetLinkStructure;
end;

implementation

{ TXMLDecoder }

//****************************************************************************
// Pre : pXMLDoc points to an existing XML file conforming with LinkStructure.xsd
// Post: Object created and doc decoded.
constructor TXMLDecoder.Create;
begin
  inherited Create();
  // Init internal vars
  fXMLDoc := pXMLDoc;
  fDigitalContext := TStringList.Create();
  fLinks := TList.Create();
  // Decode the doc
  DecodeXML();
end;

//****************************************************************************
destructor TXMLDecoder.Destroy;
begin
  if Assigned(fDigitalContext) then
    fDigitalContext.Free();
  if Assigned(fLinks) then
  begin
    FreeLinks();
    fLinks.Free();
  end;
  inherited;
end;

//****************************************************************************
procedure TXMLDecoder.DecodeXML;
var
  lls : PLinkStructure;
begin
  fDescriptor := 'This is a bunch of test data';
  new(lls);
  lls.source.HREF := 'Sadam';
  lls.destination.HREF := 'EMail1';
end;
lls.destination.Caption := 'Bomb';
lls.linkname := 'has send';
flLinks.Add(lls);
new(lls);
lls.source.HREF := 'Kimberly';
lls.source.Caption := 'Citizen';
lls.destination.HREF := 'EMail1';
lls.destination.Caption := 'Bomb';
lls.linkname := 'has received';
flLinks.Add(lls);
new(lls);
lls.source.HREF := 'Anton';
lls.source.Caption := 'Citizen';
lls.destination.HREF := 'Sadam';
lls.destination.Caption := 'Iraq';
lls.linkname := 'is buddy with';
flLinks.Add(lls);
new(lls);
lls.source.HREF := 'Kimberly';
lls.source.Caption := 'Citizen';
lls.destination.HREF := 'Ford Mustang';
lls.destination.Caption := 'Car';
lls.linkname := 'owns';
flLinks.Add(lls);
new(lls);
lls.source.HREF := 'Sadam';
lls.source.Caption := 'Iraq';
lls.destination.HREF := 'How to cook "spicy" meals';
lls.destination.Caption := 'Book';
lls.linkname := 'loans';
flLinks.Add(lls);
end;
//*****************************************************************************/
begin
if Assigned(fLinks) then
begin
if index < fLinks.Count then
Result := PLinkStructure(fLinks[index])
else
Result := nil;
end
else
Result := nil;
end;
//*****************************************************************************/
function TXMLDecoder.GetLinkCount: longword;
begin
if Assigned(fLinks) then
Result := fLinks.Count
else
Result := 0;
end;
//*****************************************************************************/
procedure TXMLDecoder.FreeLinks;
begin
while fLinks.Count > 0 do
begin
dispose(fLinks[0]);
fLinks.Delete(0);
end;
end;
//*****************************************************************************/
end.
Appendix D. Bibliography

[2D barcodes]
http://www.barcodeman.com/faq/2d.php
http://www.pocketpcmag.com/_archives/nov03/barcodes.asp

[AutoLink]
http://blog.searchenginewatch.com/blog/050216-124431

[Dexter]
The Dexter Hypertext Reference Model
Frank Halasz and Mayer Schwartz
February 1994
http://portal.acm.org/citation.cfm?id=175235.175237

[Generic Links]
Adam Warren
1992
http://www.soton.ac.uk/~connect/misc/mcosm1.html

[GoF95]
Design Patterns: Elements of Reusable Object-Oriented Software
E.Gamma, R. Helm, R. Johnson and J. Vlissides
Addison-Wesley, 1995.
http://www.amazon.com/exec/obidos/ASIN/0201633612/objectarchite-20/103-4570218-1778245

[IDOL]
http://www.autonomy.com/content/Products/IDOL/
http://www.autonomy.com/content/Products/IDOL/f/Classification.html#01

[Lucene]
Doug Cutting et.al.
1997
http://lucene.apache.org/java/docs/

[Ontolica]
http://www.navigosystems.com
http://www.navigosystems.com/dansk/ontolica/produktoversigt.html

[Physical Hypermedia]
“Physical Hypermedia”: Organising Collections of Mixed Physical and Digital Material
Kaj Grønbæk, Jannie F. Kristensen, Peter Ørbæk and Mette Agger Eriksen
August 2003
http://www.daimi.au.dk/workspace/site/content/heading_07/papers/ht03.pdf
[Stick-e]
The stick-e note architecture: extending the interface beyond the user
Jason Pascoe
1997
http://portal.acm.org/citation.cfm?id=238344

[XPointer]
Henry S. Thompson and Dan Connolly
April 2000
http://www.w3.org/XML/Linking

[XSD]
C. M. Sperberg-McQueen and Henry Thompson
April 2000
http://www.w3.org/XML/Schema