XIRAF: An XML-IR Approach to Digital Forensics

W. Alink

October 21, 2005

Database Group
Faculty Electrical Engineering, Mathematics, and Computer Science
University of Twente

Supervised by:
Dr. R.A.F. Bhoedjang [2]
Dr. P.A. Boncz [1]
Dr. Ir. D. Hiemstra [3]
Dr. Ir. M. van Keulen [3]
Dr. Ir. A.P. de Vries [1]

[1]CWI, Amsterdam, The Netherlands
[2]NFI, Ypenburg, The Netherlands
[3]University of Twente, Enschede, The Netherlands

Abstract

English: This Master's thesis addresses problems in current digital forensic investigations. It proposes the XIRAF system as a novel approach towards the integration of existing forensic analysis tools using XML technology. The concept of integrating these tools can be compared to the concept of concurrent XML hierarchies. The representation of concurrent XML has been widely studied, but concurrent XML hierarchies cause a variety of unsolved problems when such data has to be queried. Querying concurrent XML hierarchies has however many practical applications, including digital forensics, question answering, and multimedia retrieval. This thesis introduces Burkowski axis steps in XPath as a viable solution for the digital forensics application area. The steps can be used in stand-off XML annotation in which the content is separated from the annotations. This approach has many advantages over inline annotation, especially in field of digital forensics. The introduced steps have been implemented in an existing open source XQuery system called MonetDB/XQuery.

Nederlands: Deze scriptie adresseert problemen in digitale forensische onderzoeken. Het XIRAF systeem wordt gepresenteerd als een innovatieve benadering voor het integreren van bestaande forensische analyse applicaties en maakt gebruik van XML als onderliggende taal. Het concept van het integreren van forensische applicaties kan vergeleken worden met het concept van overlappende XML structuren. Het representeren van overlappende XML is uitgebreid bestudeerd in de wetenschappelijke literatuur, maar onopgeloste problemen doen zich voor wanneer overlappende XML bevraagd wordt. Desondanks heeft overlappende XML vele applicatiegebieden; denk aan digitale forensische wetenschap, natuurlijke taal vraag beantwoording en het bevragen van multimedia. Deze scriptie introduceert de Burkowski-stappen in XPath als een plausibele oplossing voor het bevragen van overlappende hierarchien in digitale forensische onderzoeken. De stappen maken het mogelijk om overlappende hierarchien te bevragen in documenten waarbij data gescheiden is van annotaties. De geintroduceerde stappen zijn geimplementeerd in een bestaande open-source XQuery database systeem, genaamd MonetDB/XQuery.

Foreword

First of all I would like to thank my Mom, Dad, and my brothers; Robert and Laurens. Without them I would not have been able to finish this thesis. Thanks also go to my supervisors Peter, Arjen, Raoul, Maurice and Djoerd for providing a stimulating research environment and helping me out with writing down this thesis. And of course thanks to the INS1-crew at the CWI and the guys from Munich for their support, especially to Jan R, Fabian, and Stefan who dealt with all my (incomplete) bug-reports.

Contents

Chapter 1

Introduction

An average hard disk in a modern computer can store more text than a human could read in a lifetime. Luckily, some of us are very organised and know where they have stored their important data and therefore do not have to read the whole disk when searching for a particular item, but how does a forensic investigator know what to read when a confiscated hard disk lies on his desk?

This master's thesis for the Computer Science programme at the University of Twente deals with current problems in digital forensic investigations and tries to overcome those problems by using advanced database and Information Retrieval (IR) techniques. The XIRAF —an XML Information Retrieval Approach to Digital Forensics— system is a result of this master's project. There are three parties involved in the project; the Netherlands Forensic Institute (NFI)[1] is the initiator of this project. The NFI has expertise in the field of digital forensics. The Research Institute for Mathematics and Computer Science (CWI)[2], and in particular the Database cluster, has expertise in the field of IR and data management. The last party involved, the Database cluster at the University of Twente (UT)[3] has also done extensive research in IR.

Forensics is the use of science to gather evidence. Digital forensics is the use of science to gather evidence from digital sources. Digital forensic investigations can be divided into online and offline investigations. In online investigations a live system, or the communication between live systems, is being monitored. In offline investigations data from an offline (dead) system (the device has been switched off) is examined. This thesis focuses on offline investigations. A typical forensic media investigation work-flow is depicted in Figure 1.1; after a piece of digital media has been seized and brought to a digital forensic lab, the first step is to create a working copy of the data. This copy is then analysed with forensic analysis tools and finally the forensic investigator will write a report of the findings which can in turn be used in court or for prosecution, defence or investigation.

1.1 Goals and problems

The analysis of the data in a digital forensic investigation is currently the most labour intensive part of an investigation. One of the problems in digital investigations is that an investigation typically involves a huge amount of data. Currently the amount of data to be processed in an investigation ranges from tens of gigabytes to multiple terabytes and these amounts are continuously increasing.

Searching through such vast amounts of data involves problems, but not only the amount of data is causing problems, it is also the increase in the number of digital forensic investigations [33] that causes problems, and calls for a more automated approach towards digital forensic investigations. An objective could be to try to speed up the process of digital forensic investigations, thereby increasing the throughput. Another objective could be to make it easier for forensic investigators to find the desired evidence and thereby relying less on skilled digital forensic investigators.

[1] http://www.forensischinstituut.nl/
[2] http://www.cwi.nl/
[3] http://db.cs.utwente.nl/

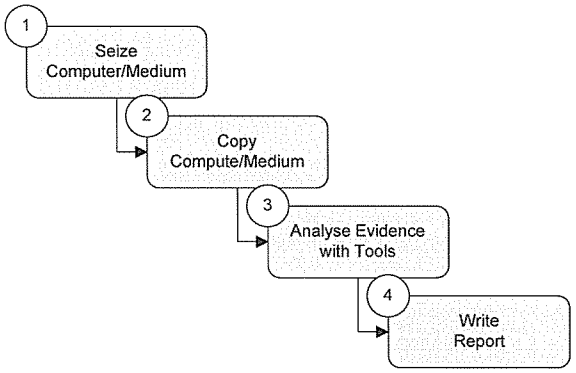


Figure 1.1: Typical phases of a forensic investigation

The last objective could be achieved by supplying forensic investigators with equipment which requires less expertise, so that regular forensic investigators can conduct their own digital forensic investigations, without the need for a *digital* forensic investigator. An approach for the first objective would be to offer forensic investigators more powerful tools to find evidence with, thereby limiting the time per investigation, and increasing the quality of an investigation.

None of these two goals have reached their full potential. Currently digital forensic investigators analyse case data using a set of forensic analysis tools. These are used to extract specific features from the data. As the field of digital forensics is growing more and more *forensic analysis tools* [4] have been created. The problem with current 'tools' is that most of them have their own output format. Because of that they are not able to interact with each other. A digital forensic investigator needs to have the skills to be able to use each of these tools. With a few exceptions, almost all tools have their own query language or search interface. Another problem that can be identified is that combining the results of several tools for querying purposes is difficult.

The goal of this thesis is to set up a framework in which forensic analysis tools can be combined and to offer a single query language to be able to query across the output of multiple 'tools'. Such a system would require less qualified forensic investigators and would also provide forensic investigators with a more expressive query language. Note that this thesis is not about increasing the quality of extracted features. Extracting features is a subject with many fields of expertise (cryptography, steganography, reverse-engineering, data-filtering, data-analysis, etc) and is covered by many scientists. Neither does this thesis describe the capturing of data phase or reporting phase.

1.2 XIRAF

The XIRAF system, that is presented in this thesis, is a prototype system demonstrating how forensic analysis tools could be integrated and queried over using XML database technology. XIRAF consists of a *query interface*, a *feature extraction framework*, and a *tool repository* (see Figure 1.2). The framework applies the forensic analysis tools, which have been gathered in the tool repository, to the case data. The feature extraction framework stores the output of all the individual tools as annotations to the case data. The combination of the annotations and case data will be referred to as stand-off annotation. The query interface allows a forensic investigator to query over the extracted annotations. Separating the annotations from the actual data has been referred to as stand-off annotation. Annotations often denote a specific object found in the data (for example: a file, an e-mail or a name). Crucial is that often these objects are found at the same location in the data (for example: a name is found inside a file, or an e-mail containing a phone number) and that a mechanism is needed to relate these interesting objects to each other.

The idea to achieve our goal is to combine forensic analysis tools by converting their output to XML. The assumption is that tools can be easily managed by *wrapping* them in such a way that the tools produce

[4] forensic software packages, short: 'tools', see Appendix A for a list of common forensic analysis tools

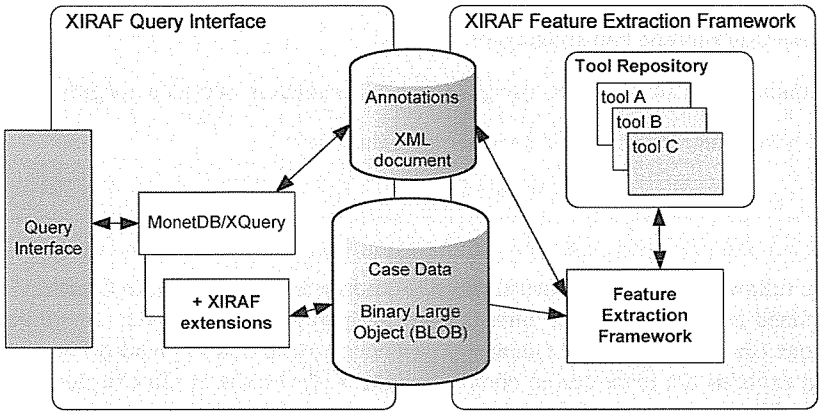


Figure 1.2: XIRAF architecture

XML. The output of these tools can then be used for further analysis and as input for other tools (Figure 1.3). All tools must first be wrapped before they can enter the tool repository.

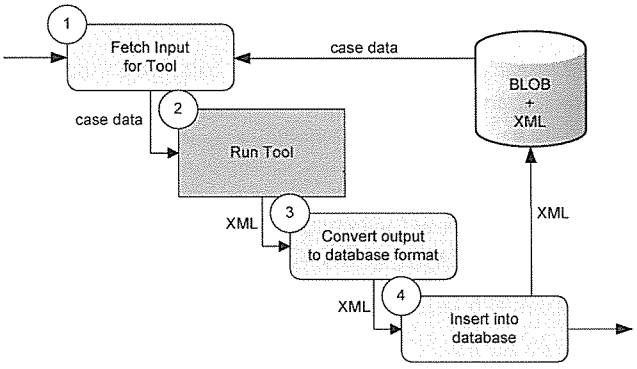


Figure 1.3: Tool Wrapping Example

A database which can store and query over XML documents (an XML database) can be used to store and retrieve the annotations made by the tools. These annotations can be queried using an XML query language. Currently several XML query languages exist, among which: XQuery[4], XPath[12] (a subset of XQuery), and NEXI[39]. The idea is to be able to query the output of the 'tools' using one of these languages. The Database cluster at the CWI together with the UT and the University of Konstanz developed an XQuery implementation[6] called 'Pathfinder' on top of MonetDB[5], a relational database management system. This XQuery implementation can handle large XML documents. Another ongoing research project, involving the UT and the CWI, resulted in TIJAH[21], an implementation of the NEXI language, which is well-suited for various IR tasks.

1.3 Research Questions

Designing XIRAF introduces a couple of research questions:

1. (a) Can output from forensic analysis tools be converted to XML and (b) does the quality of the tool's output degrade due to this conversion?

2. (a) How should stand-off annotation be shaped, and (b) can stand-off annotation be queried using widely accepted XML query languages such as XQuery? If not, (c) which extensions are necessary,

and (d) how can these extension be implemented efficiently?

3. Is it possible to formulate forensic questions using a stand-off annotation query language?
4. Is a unified approach for integrating forensic analysis tools feasible?

1.4 Outline

The thesis is structured as follows; first a digital investigation is described in more detail in Chapter 2. The current problems are outlined and an analysis of current forensic analysis tools is made. As an answer to how forensic annotations can be represented, Chapter 3 introduces the notion of stand-off annotation. Chapter 4 introduces and explains the main design choices for the XIRAF system. In Chapter 5 details on the implementation of XIRAF and stand-off annotation are given, and this is followed by results from experiments with the feature extraction framework and with the extensions to XQuery (Chapter 6). Finally in Chapter 7 conclusions are drawn and recommendations are given. The appendices contain background information about some topics discussed in this thesis. They will be referred to when needed.

> "Ancient cultures in Africa revered the giraffe, as some modern cultures do today, and commonly depicted it in prehistoric rock and cave paintings. Unknown outside of Africa, early written records described the giraffe as "magnificent in appearance, bizarre in form, unique in gait, colossal in height and inoffensive in character." The Giraffe moves about the semi-arid regions in groups. Its height allows it to keep in contact with other giraffes over large distances as well as spotting predators from afar. It is not uncommon to see other animals following a giraffe using it as an early predator warning system. The Giraffe is vulnerable when drinking. It is a quiet species although the males fight viciously for dominance over the group."
>
> *South African Wildlife - www.sa-venues.com*

Chapter 2

Background in Digital Forensics

Police escort: Tell me, inspector - all those robberies. How did you ever manage it?
Inspector Jacques Clouseau: Well, you know... it wasn't easy.

The Pink Panther, 1963

Digital forensics is an interesting field for computer science and especially for the database community. In digital forensic investigations not only the quantity of data is interesting, but it is also the specific questions about the data that makes the field of digital forensics an suitable research topic for the database community. This chapter will first explain the field of digital forensics, its sub-domains, and then describe a typical digital forensic investigation. The forensic analysis tools used during investigations will be discussed in Section 2.2. Others have already tried to integrate forensic analysis tools and these approaches are briefly discussed in Section 2.3. Finally the contours of XIRAF will be presented in Section 2.4.

2.1 Digital Forensics

Forensic science is aimed at the development and application of mainly technical and scientific methods and technologies to gather evidence in order to find truth in justice. Forensic science also focuses on the reflection of these activities from a scientific perspective. Forensic science embodies a broad spectrum of methods and technologies, which are applied in numerous fields of expertise[8].

Digital forensic science has been defined as the use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorised actions shown to be disruptive to planned operations[28].

Digital forensic science is one of the many fields of expertise in forensic science. Digital forensic science can again be divided in many subfields. One of these subfields is the analysis of confiscated static (offline) data. This subfield's goal is to retrieve evidence from digital sources, where the digital data has already been retrieved. Digital evidence can be many things: traces in log-files, digital letters, e-mails, etc. Finding sufficient evidence is actually quite difficult; when a hard disk contains illegal images, this does not directly imply that these images were put there by the suspect. The digital evidence stems from digital media. Digital media can be hard disks, cd's, memory sticks, mobile phone memory, and sometimes even volatile memory.

The common problems in the field of digital forensics are:

- The size of the data: data from digital sources tends to be large in volume.
- The volatility of the data: data from digital sources is sometimes difficult to preserve or capture.

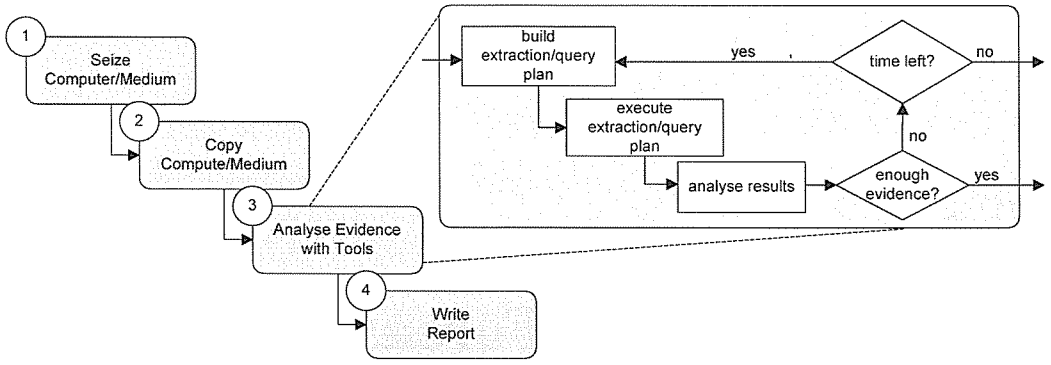


Figure 2.1: Typical digital forensic investigation depicted in steps

- Heterogeneity of the data: data from digital sources is often just a series of 0's and 1's which are hard to interpret.
- The integrity of the data: parts of the data may have been deleted or overwritten.

2.1.1 A digital forensic investigation

A digital forensic investigation starts when digital media and one or more questions about the data on the media are given to a digital forensic investigator. A digital forensic investigation usually involves the steps shown in Figure 2.1.

The investigation starts by copying the data to local storage. One case may consist of multiple data carriers (several hard disks, cd's, memory sticks, etc). The copying of the data makes sure that the original data will be preserved, and that the evidence is not disturbed. In the second step the forensic investigator selects *forensic analysis tools* (programs that search for pieces of evidence). Sometimes specific *tools* might be most useful when searching for a specific type of evidence. In other cases more generic tools (for example text-indexing- and file-system-analysis-tools) might be used. Afterwards the data (or a selection of the data) will be fed to these 'tools'. Depending on the size of the data and the complexity of the 'tool', this step can last anywhere from minutes to several hours/days. Good forensic analysis tools should be scalable and robust; they might take a while, but they do complete their work.

The output of the 'tools', this can be a query-able index, a short summary/answer, or a browsable index, will be evaluated by the forensic investigator. If the forensic investigator thinks he/she has gathered enough evidence, the case can be closed by writing down the findings in a report. If the tools did not produce satisfactory results, there may be too few or too many results. In case of too many results, the forensic investigator has to apply other tools or queries to downsize the results. In case of too few results, the forensic investigator has to choose a different set of tools and apply them to the data. This can not be done infinitely, because a digital forensic investigation often has strict time limitations.

2.1.2 Typical queries

The queries asked in digital forensic investigations are quite diverse. Often the exact query is not known, and finding the right question is an iterative process of analysing the data and refining the query. It happens that only a few keywords, like names or events, are given as the question, and the answer should be anything about those keywords. Often this involves multiple steps, in which the question is refined using the clues which were found.

Frequently asked questions are "did person x visit site y on the Internet?", "has this computer been hacked into?", and "was person x on-line at time y?", but these questions most of the time can not be answered by just asking a computer for the answer. The questions have to be split into more computer-query-language-like questions:

- Is a log-entry for event x present?

- Is a cookie for site y present?
- Do any of the files in list z appear on the disk?
- Which e-mails have word p in the subject?
- Which items were created or modified on date q?
- For which items in the Internet history are there no corresponding files in the "temporary Internet files" folder?

Other queries, to get a better grasp of the usage of the computer, could be:

- Which instant messenger('chat') applications are installed?
- Which image editing applications are installed?
- When was the computer last rebooted, re-configured?
- Was the hard disk almost full?

2.2 Forensic Analysis Tools

Forensic analysis tools are used to find the answer to questions asked in forensic investigations. Forensic analysis tools extract features from a given set of data. They either

1. directly present the extracted features to the user,
2. create an index on the extracted features and offer a query mechanism to the user,
3. or offer an interface through which the extracted features can be browsed.

Tools which present all the extracted features directly to the user are often tools of which the output is used as input for other tools, or allow for import in a database. The reason for this usage is that there are hardly any tools that scan the complete case data and thereby extracting only a single feature or at least a list with a readable amount of features. This behaviour would be to costly. Tools which create an index on extracted features are often a (simplified) DBMS. These types of tools often have a simple query language like a stripped down SQL or a boolean query language. The index created on the data is often tool specific (only usable for one tool) and kept hidden from the user. It is often not possible to export the index to use in another database environment. The last type of tool, presenting a browsing interface to the investigator, focuses on the representation of the data. For example a tree-view of a file-structure, a sortable (and sometimes queryable) list of e-mails, a time-line of log-events. Browsing can be very handy when an investigator does not exactly know what to look for. Creating a convenient browsing view to aid the investigator in finding specific objects in large quantities of data is difficult. Browsing is often designed specifically for a tool.

Forensic analysis tools can be either specific (for example only finding e-mails created with "Mozilla E-mail v1.7") or generic (for example being able to find emails by looking for a 'Subject' field and an e-mail-address in raw data). The first category will be very selective, but accurate. The latter one will not be selective, but might have a large false-positive rate (and might be less accurate). If evidence is partly deleted, most of the specific tools will fail to identify the evidence, while generic tools might include the evidence (although sometimes incomplete).

Some of the forensic analysis tools which fall in category (2) or (3) have an internal scripting language or a programming interface with which the features can be exported in a user defined format. In this way these tools can be used as being of category (1).

A well-known area of digital forensics is the analysis of file-system information. Digital forensic investigators share information on how to analyse file-systems and discuss new techniques in online journals such as the Sleuthkit Informer[10], and the International Journal on Digital Evidence (IJDE)[1]. The forensic investigation as process is discussed in workshops such as the Digital Forensic Research WorkShop (DFRWS)[2].

[1] http://www.ijde.org/
[2] http://www.dfrws.org/

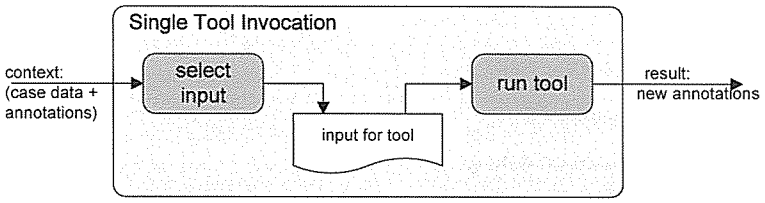


Figure 2.2: Invocation of a typical tool: first a selection is made of the input data, the tool will be run on this selection and afterwards a result will be output

Example of a tool: the 'E-mail Examiner' by Paraben Forensics [3]. This forensic analysis tool extracts e-mail messages from e-mail-archive-files of various kinds. The input data can be in `.pst` (Microsoft Outlook) format, but also many other known mail-archive formats are supported. The result is a list of e-mails. The tool has a user interface in which the investigator can browse, sort and read e-mails. It can also export the e-mails to a file.

Many forensic analysis tools have been developed. Some in-house at the NFI, but other companies and institutes have been developing tools as well. These tools do not always have to be designed specifically for forensic purposes, but might as well be used for Information Retrieval in general. A short overview of currently available tools is given in Appendix A. Invoking a tool typically involves the steps depicted in Figure 2.2.

Sometimes features are extracted from or calculated over the case data itself (for instance the MD5 checksum of a file), but it can also occur that external sources are queried to provide the metadata, for instance looking up files in a database to check whether they are known to be irrelevant to the case (for example system files or program files). The National Software Reference Library (NSRL)[4] provides a set of cd's containing hashes of known-to-be-good-files, and these hashes can be used in investigations.

Some other tools listed in the Appendix are actually a set of tools integrated into a single environment (Encase[5], Sleuth Kit[6]). These products might offer a search, browse, case administration, and scripting environment to the set of tools as a single package. But despite of available scripting environments, it is often difficult to integrate other forensic analysis tools into these products.

2.3 Integration of Forensic Tools

The process of refining the initial queries into computer-query-language-like queries is a process performed by the forensic investigator (see Figure 2.3). Currently the forensic investigator is supposed to know how to translate the queries into the tool-specific query languages. During a typical digital forensic investigation multiple tools are used. In Figure 2.4 a simple investigation is depicted. The forensic investigator is looking for a specific e-mail which was addressed to 'John'. He/she first uses a file-system-analysis-tool to extract the individual files from the case data. Next, an e-mail-archive-analysis-tool is used to extract the e-mails from the e-mail archives found in the list of files.

Although this seems quite trivial, the forensic investigator might not find all e-mails he/she was looking for. For example, some of the e-mail-archives could have been deleted and not been detected by the file-system-analysis-tool, or some of the archives might have been in a zip-file or were encrypted. In these cases the tools used were too specific. On the other hand the e-mail might have been found by the tool, but it got the 'to'- and the 'from'-field mixed up and the investigator could therefore not find the specific e-mail. In this case the tool was too generic (or the query was too specific).

Not every tool flawlessly extracts the desired data; often a tool is designed to extract information of a specific type. For example: some tools only extract data from NTFS-partitions while others examine

[3] http://www.paraben-forensics.com/
[4] http://www.nsrl.nist.gov/
[5] http:/www.guidancesoftware.com/
[6] http://www.sleuthkit.org/

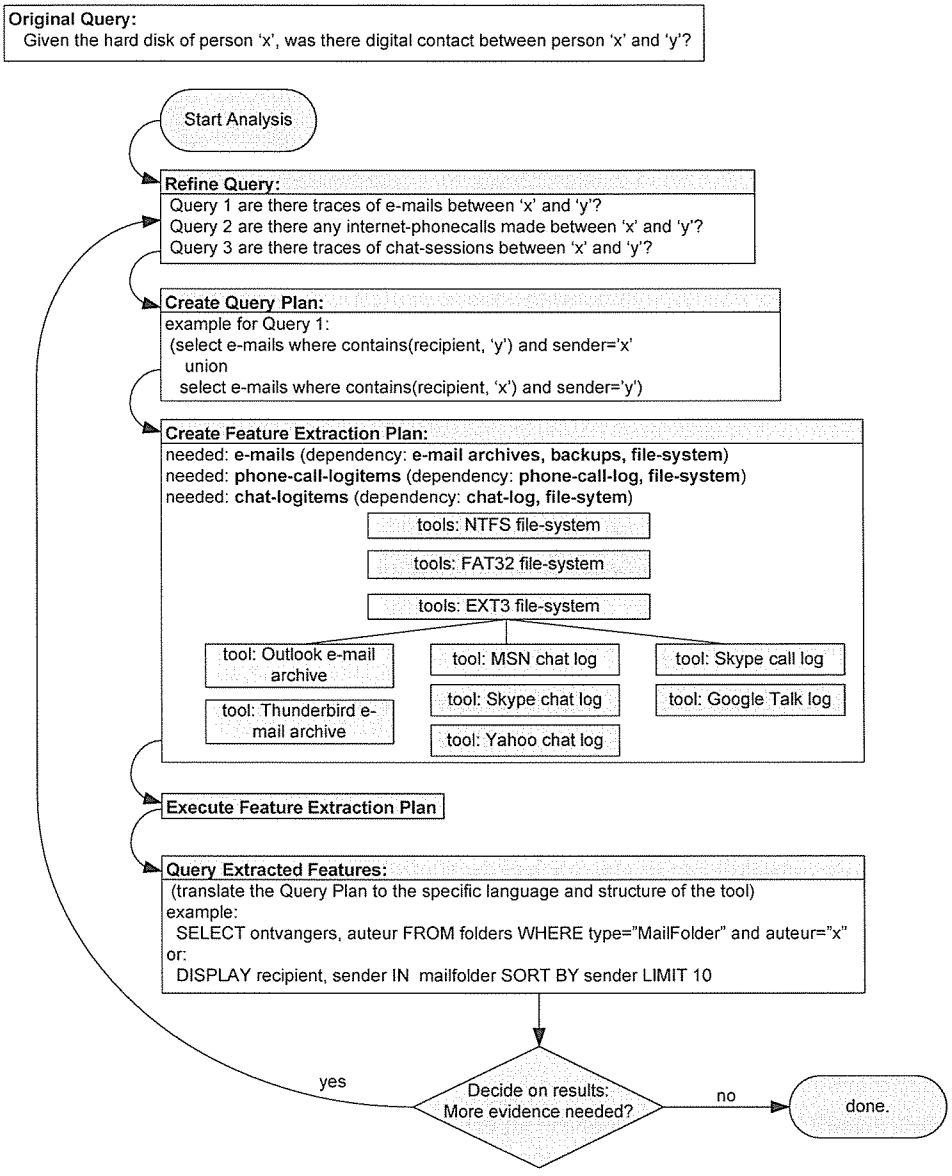


Figure 2.3: Example query refinement plan

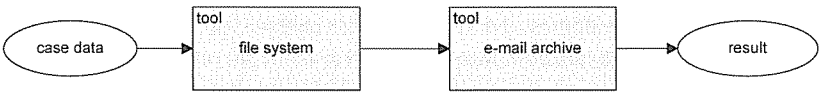


Figure 2.4: Tools used in an investigation (simplistic)

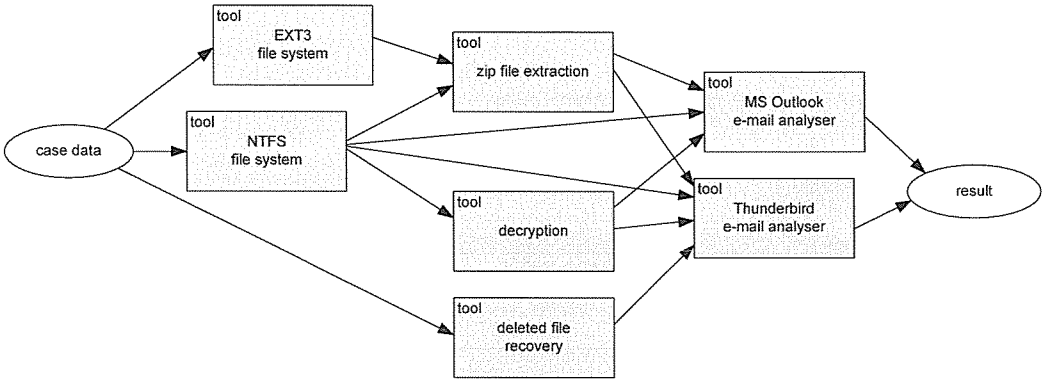


Figure 2.5: Tools used in an investigation (more advanced)

FAT32-partitions, others tool could interpret a Mozilla e-mail archive while again another tool was designed to extract a Microsoft Outlook e-mail archive. This leads to the fact that sometimes more than one tool has to be tried to get the expected results. A more advanced investigation scenario, where more tools are used, is depicted in Figure 2.5. In a digital forensic investigation a series of problems arise:

1. Some tools need to be called recursively (for example a zip-file in a zip-file in a ... etc.) or in a specific order (an encrypted file in a zip-file or a zip-file in an encrypted file).
2. With each step in the investigation the investigation becomes more complex. The forensic investigator needs to make choices, and the number of choices grows rapidly.
3. A forensic investigator needs to know all the tool's interfaces and often has to relate the answers of different tools himself.
4. A forensic investigator constantly has to keep track of the context (which items in the data have been analysed with a certain tool, which have not, which items are irrelevant, etc...)
5. A complete investigation (running all the tools on all the case data) will find most evidence, but the output of the less accurate tools cannot be related to the output of other tools, and will therefore be difficult to query. For example; if a tool identifies an e-mail by looking for a "Subject" field and an e-mail address in unprocessed data, the forensic investigator might not be able to relate this e-mail to the file in which it was stored.
6. A complete investigation (running all the tools on all the case data) is often not possible because of time and space constraints of the investigation. Presumably intermediate selections in tools to use, and/or data to search either by hand or automatic are needed to trim the size of the case data.

To solve the problems mentioned above a unified approach is needed. One way to solve the problems 1, 2, 4(, 5) is to make the extraction hierarchy (the step-by-step approach of calling tools) explicit. Beebe and Clark[3] propose to create a list of all possible objects to be identified in the case data and make a complete extraction-dependency (which tools to use after a specific object has been found) hierarchy. They introduce the SEE Data Analytical Objectives-Based Task Hierarchy, in which a 14 step procedure for analysing a hard disk is proposed. Such a list has also been proposed by others, but creating such lists is merely creating a snapshot of today's technologies. For example: the analysis of 'internet-phone-calls', a relatively new phenomenon, is not taken into account into the list. Nor is the automatic analysis of images included. It might be reasonably safe to assume that computer applications, but also technology such as in the fields of audio-, image-, video-analysis, will continue to evolve. Therefore it might also be save to say that creating *the* list that Beebe and Clark try to generate is most likely not feasible. Digital forensic experts should continue to create new tools extracting more, or new features. But the forensic investigator, the person trying to find evidence and not caring for the technology behind the system, should be able to use the new extraction features without actually knowing about them (or maybe just a little).

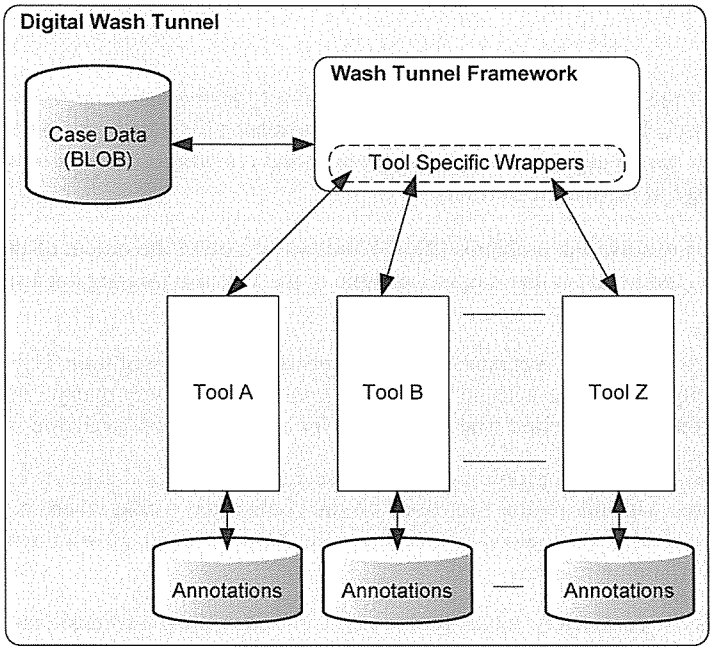


Figure 2.6: Architecture of current digital wash tunnels: each tool stores and manages its own annotations.

Problem 6 is difficult to solve for complete investigations, but a relevance assessment such as described in [32] could be used to identify irrelevant case data and thereby limiting the size of the case data.

Wash Tunnels

When multiple forensic analysis tools are managed by a single environment and when feature extraction is somehow automated, this is commonly referred to as a *digital wash tunnel* (Dutch: *digitale wasstraat*). Digital wash tunnels are a new phenomenon. Their entrance is mainly due to the increasing complexity of digital investigations. Digital wash tunnels try to hide the individual interfaces of forensic analysis tools by offering the forensic investigator a single integrated querying environment.

Looking at the digital wash tunnels from the tool-integration perspective; none of the digital wash tunnel attempts allow for simple addition of new tools to the repository of tools. A vast amount of changes need to be made to a tool to integrate it into the wash tunnel. The interface to effectively add new tools to the repository is either hidden because of commercial reasons, or is being obstructed due to necessary adaptations to the core database schema or code of the digital wash tunnel.

There are also problems with current wash tunnels when looking at them from a query perspective: some of the digital wash tunnels attempts that have been made are merely more than the above description; tools have been put under a single roof, but the tools kept their own non-integratable indexes, query languages and/or browsing features. A typical wash tunnel architecture is depicted in Figure 2.6. The wash tunnel framework invokes the tools, but does not capture the output. Instead, all tools maintain their own database with annotations. Querying the annotations needs to be done by querying the individual databases.

Examples of digital wash tunnels are ZyIndex (by Zylab[7]), Forensic Toolkit (by AccessData[8]), Sleuthkit (by B. Carrier[9]), and Encase (Guidance Software[10]). The Coroner Toolkit (TCT), on which the Sleuthkit is based, is arguable a digital wash tunnel. It combines several tools, but tries to achieve a single goal: extracting the file-system structure.

[7] http://www.zylab.nl/
[8] http://www.accessdata.com/
[9] http://www.sleuthkit.org/
[10] http://www.guidancesoftware.com/

2.4 XIRAF

From points (2), (3) and (4) of the previous section we see a clear need for integration of forensic analysis tools. This is backed by the article by Noblett et al.[24] and forms the reason for starting the XIRAF project. XIRAF will be a digital wash tunnel capable of integrating tools, and allowing a forensic investigator to query without actual knowledge of the individual tools.

- The XIRAF project tries to solve the problems (2), (3) and (4) by storing the output of the tools in a generic language (XML) and making the output available to the forensic investigator using a single language (XQuery).
- The XIRAF project tries to solve the recursive and order-specific invocation of tools (1) by allowing tools to abstractly specify their input with an *input descriptor* as an XQuery query. With such an input descriptor the system can be alerted when new input for a certain tool is available.
- The XIRAF project tries to solve the problem of relating seemingly unrelated extracted features (5) by annotating all features with the region (position) on the confiscated media where it was found. XIRAF makes it easy to query over regions by extending the XQuery language with new *axis* steps that find enclosed or overlapping regions.
- The XIRAF project has not completely solved problem (6), but does allow a forensic investigator to make a selection of forensic analysis tools to run on the case data.

Chapter 3

Regions and Stand-off Annotation

'n stukkie van 'n bloemkool is ok weer'n bloemkooltie op zich

't giet zoas 't giet, Skik

XML stands for *eXtensible Markup Language*[41]. XML is used for inline annotation of data, i.e. the markup (or metadata) is interleaved with the actual content. *Stand-off annotation* is the external markup of data, and leaves the original data physically untouched by maintaining a separate document containing the annotations. One of the goals of this thesis is to query forensic case data as XML. XIRAF uses *stand-off XML annotation* to accomplish this goal. The use of stand-off annotation can be justified by finding a solution to overcome the following two key problems when storing XML annotations on case data:

1. The amount of data in current investigations is huge, for example physically interleaving case data with XML elements would be very inefficient in terms of disk I/O. If an element would be inserted in the middle of the case data, roughly half of the document needs to be shifted to make room for the annotation.

2. Forensic analysis tools extract features from the case data (annotate the data) in different ways creating concurrent hierarchies. Combining these hierarchies (for representation or querying) is not straightforward, and might for example produce partially overlapping annotations.

In this chapter a definition of *stand-off annotation* stating how the term is used in XIRAF will be given (Section 3.1), followed by a short overview in Section 3.2 of the term 'stand-off annotation' as it has been used in earlier work. Other approaches similar to stand-off annotation are summarised in Section 3.4. The remainder of the chapter deals with two crucial aspects of stand-off annotation: representation of stand-off annotation (Section 3.3) and querying stand-off annotation (Section 3.5). For the latter we base our approach on earlier work by Burkowski[9].

3.1 Definition

Stand-off annotations, as the name already reveals, are *annotations* separated from the *data*. In the forensic case this data may be a hard disk image, to which we often refer as a *Binary Large Object* or BLOB. The term *annotation* is used as "metadata describing a certain object in a BLOB". In the forensic case our annotations are the features extracted by forensic analysis tools. Such metadata typically *labels* an object (a file, an e-mail, a date, etc.), and optionally provides additional information about the object (type of data, name of the object, time of last access, author, etc.). Metadata will be defined as an XML fragment, leaving the actual structure of the metadata completely free (See: Figure 3.1).

In Figure 3.2 an example is given of how data could be annotated in different hierarchies. Each annotation defining a different object in the BLOB. In our approach, *regions* are used to denote objects in the BLOB. The term *region* has already been briefly introduced: it identifies a contiguous byte segment present

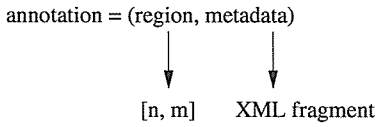


Figure 3.1: Annotation, consisting of a region and some metadata

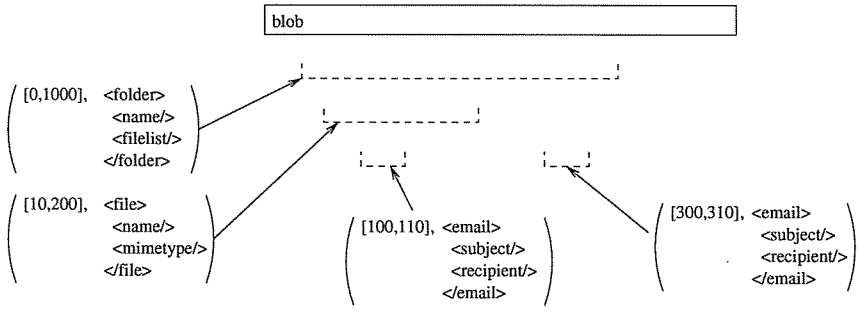


Figure 3.2: Annotation example involving multiple (concurrent) hierarchies

in the BLOB (case data). Regions are used, because objects annotated in different hierarchies do not inherit an implicit relation with each other. Regions make it possible to link objects to each other, possibly found with different tools, in concurrent hierarchies, but at the same location in the BLOB. A region can be represented as $[m, n]$ where $(d_m, d_n \in d, m \leq n)$ and d is the data, and d_i represents byte i in d. Note that a region can not be empty, it should denote 1 or more discrete items (in our case bytes) in the data. Region annotations can for example be represented in XML by adding the attributes `start="`m`"` and `end="`n`"` to the root XML element of the annotation. Stand-off XML annotations will in this way conform to the XML specification. Stand-off annotations can be expressed and gathered in an XML document. In a very naive way, all annotations could be listed after each other, but we could exploit the semi-structuredness of XML further, by placing annotations as children of other XML elements from which they are derived (data lineage). How stand-off annotations are gathered and stored, and how they are queried will be explained in Chapter 4.

3.2 History

Stand-off annotation is hardly ever used when "indicating an XML document describing an external piece of data". This is however how the term is used in this thesis. Some other authors are listed below that have used the term stand-off annotation, or described the phenomenon of stand-off annotation.

Pianta and Bentivogli (2004) wrote a paper with the title "Annotating Discontinuous Structures in XML"[29]. They claim to have introduced the term 'stand-off annotation' and give a pointer to SGML[1]'s CONCUR feature.

Dekhtyar and Iacob (2004) "A Framework for Management of Concurrent XML Markup"[13] They use the terms 'concurrent XML markup' and 'distributed XML documents'. These terms do not define the same principle, but both terms will probably lead to the same implementation: the data will probably not be stored for every document in the distributed XML document. The pointers in this paper refer to GODDAG.

DeRose (2004) talks in "Markup Overlap: A Review and a Horse"[14] about standoff markup, apparently also referred to as "out of line" markup. DeRose defines standoff markup as a technique including methods

[1] SGML is the predecessor of XML

that locate at least some markup quite separate from the content it applies to. A pointer to the TEI join structure[35] and the XLink recommendation by the W3C [15] are given.

Ogilvie (2004)[25] Ogilvie speaks about 'offset annotation' and 'byte offset annotation' in "Retrieval Using Structure for Question Answering". He does not give any pointers to earlier work.

Durusau and O'Donell (2003) define in "Concurrent Markup for XML Documents"[27] *Bottom Up Virtual Hierarchies* (BUVH). This structure is comparable to stand-off annotation. The phrase BUVH appears to be uttered by Sperberg-McQueen.

Liefke and Suciu (2000) worked on compression of XML data in "XMill: an Efficient Compressor for XML Data"[20]. They talk about 'separating markup from content', but Liefke and Suciu do not give a name to this phenomenon, and it is only used for achieving better compression in XML documents. There is no link with concurrent XML hierarchies in this paper.

Navarro and Beaza-Yates (1997) argue in "Proximal Nodes"[23] that separating text from structure is very important. (text is static, multiple hierarchies can be built on top of them. Before the XML era, but do talk about hierarchical structures, and also about adding other indexes like suffix arrays (PAT expressions model) on top of the static text.

Thompson and McKelvie (1997) introduce 'standoff annotation' in "Hyperlink semantics for standoff markup of read-only documents"[38], but define standoff annotation as having a kind of 'base'-document containing text and a basic markup and being able to add more annotation documents to this base document.

3.3 Representation

Stand-off annotation is useful in the following ways:

1. Annotations can be easily added, without altering the case data. Creating new hierarchies on the data will only cause an update of the XML document and not to the data itself. Having to modify the case data is also unattractive for forensic reasons (i.e. presentation of proof).

2. Stand-off annotations are allowed to partially overlap each other (will be explained later in this section). This is not possible when case data is interleaved with XML, due to the inherent tree-structure of an XML document.

3. Stand-off annotation allows other (non-XML) annotations to be created on top of the data besides the XML representation (for example: suffix arrays, word indices), without requiring to keep multiple copies of the data.

Earlier in this chapter, we already stated that stand-off annotations could be captured in a single XML document. A central question in stand-off annotation is how the resulting XML structure should match the structure defined by the regions of annotations. Before reasoning about querying stand-off annotation, first the interesting phenomenon of partial overlapping regions will be introduced: *partial overlap* between two regions (A and B) occurs (see Figure 3.3) when the intersection ($A \cap B$) and both subtractions ($A - B$ and $B - A$) are all non-empty. This is a special situation, because elements in XML cannot partially overlap. If an XML element a starts before element b, than a needs to end after element b, which makes the subtraction ($b - a$) the empty set, or b needs to start after a, which makes the intersection ($a \cap b$) empty.

Being able to express partial overlap can be very useful. Good examples of why overlapping annotations are needed can be found when extracting features from video fragments [2]. In Figure 3.4 an example of overlapping annotations is given. Depicted is a video fragment divided into a couple of scenes and during the fragment music is played. The music fragments can span multiple scenes, but a scene could also span

[2]The occurrence of partial overlap in digital forensic analysis is less common, that is why this chapter will be explained using an example from the area of annotating video data.

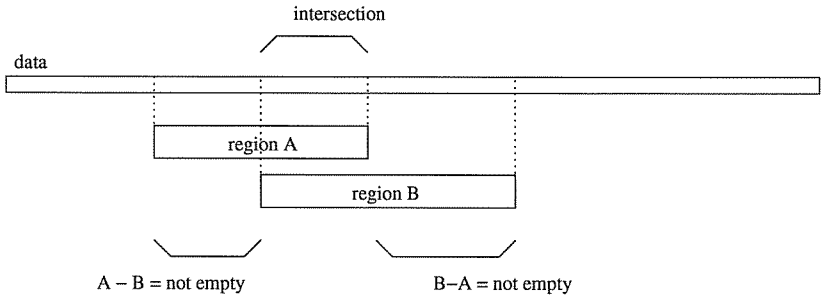


Figure 3.3: Two regions which partially overlap each other ($A - B \neq \emptyset$, $B - A \neq \emptyset$, and $A \cap B \neq \emptyset$)

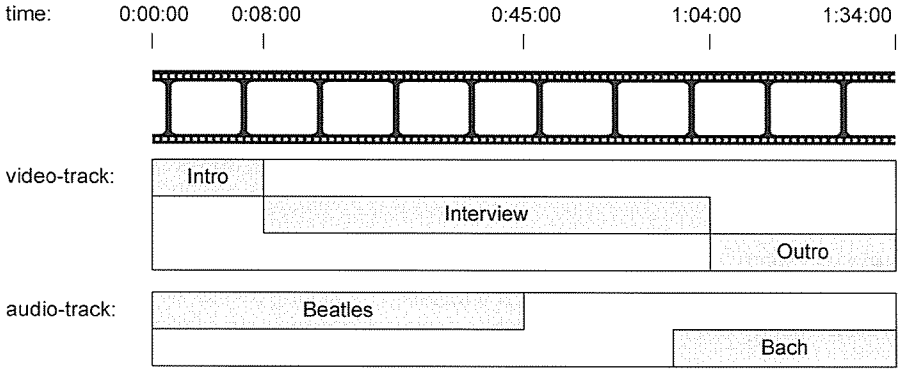


Figure 3.4: Example of annotating video fragments (causing overlapping annotations)

multiple music fragments, potentially causing partial overlap. The annotations can be captured in a stand-off XML document by marking the events with their corresponding `start`- and `end`- times (Figure 3.6).

3.4 Related Work

Some computer scientists have proposed markup languages similar to the stand-off annotation format. Languages like "General Ordered-Descendant Directed Acyclic Graph"(GODDAG)[36], and "Layered Markup & anNotation Language"(LMNL)[3][26][30] are also capable of annotating the video example from Figure 3.4. Only LMNL and GODDAG are mentioned here, as they probably are the most well-known XML dialects, but many more dialects have been proposed[4]. LMNL and GODDAG are quite similar to each other, and also to XML. Figure 3.7 which shows the same example from Figure 3.4 marked up in LMNL. The main difference with an XML annotation is that in LMNL partial overlap can be written down allowing

[3] http://www.lmnl.net/

[4] http://www.pault.com/pault/pxml/xmlalternatives.html

Figure 3.5: Tree view of video fragment example

```
<sample>
 <video>
  <scene id="Intro" start="00:00:00" end="00:08:00"/>
  <scene id="Interview" start="00:08:01" end="01:04:00"/>
  <scene id="Outro" start="01:04:01" end="01:34:00"/>
 </video>
 <audio>
  <music artist="Beatles" start="00:00:00" end="00:45:00"/>
  <music artist="Bach" start="01:00:00" end="01:34:00"/>
 </audio>
</sample>
```

Figure 3.6: Annotations for the video example in XML

```
[sample}
 [video}[audio}
  [scene id="Intro" start="00:00:00" end="00:08:00"}
    [music artist="Beatles" start="00:00:00" end="00:45:00"}
  {scene]
  [scene id="Interview" start="00:08:01" end="01:04:00"}
    {music]
    [music artist="Bach" start="01:00:00" end="01:34:00"}
  {scene]
  [scene id="Outro" start="01:04:01" end="01:34:00"}
    {music]
  {scene]
 {video]{audio]
{sample]
```

Figure 3.7: Annotations for the video example in LMNL. GODDAG uses a similar language.

us in case of the video example to express the events in chronological order. Languages like LMNL and GODDAG are however not widely supported. This is probably due to their short existence and because an implicit order between elements is difficult to define/grasp (there is no equivalent representation in a tree-graph). Another disadvantage of LMNL and GODDAG is that they are not a subset of XML, which would ease the integration and probably make it easier for the languages to get accepted.

Another approach is *concurrent XML*[18], which is basically a kind of mixture between stand-off annotation and inline annotation. In concurrent XML multiple XML documents contain the same data annotated with different hierarchies. In [13] concurrent XML hierarchies are defined as a collection of DTDs[5] sharing the same root element. Concurrent XML has also been defined in *SGML* by the introduction of the *CONCUR* feature[29]. This feature allows concurrent hierarchies to be represented in a single document. The SGML CONCUR feature uses inline annotation for the different hierarchies, but the drawback of SGMLs CONCUR was that it required the DTD of each hierarchy to be complete. The LMNL language has been derived from SGML with CONCUR, and the requirement of the complete DTDs has been dropped in LMNL. The SGML CONCUR feature never made it to a standard or a recommendation, nor was a query language relating the concurrent hierarchies to each other designed. Although, an SGML CONCUR editor has recently been presented by Hilbert, Schonefeld and Witt[22], which might be a sign of life of SGML CONCUR.

DeRose has defined CLIX[14] which adapts the LMNL language into an XML compliant language using TEI-Milestones(placeholders inside an XML document). In [14] an overview of annotation approaches is presented and also the need of being able to express a region in stand-off annotation as a *set of contiguous extents* is stated.

[5] a DTD is a way to describe the structure of an XML document

Stand-off annotation is often used for annotating non-textual objects (images, video, audio) [6] of which the data is difficult to represent as XML. Stand-off annotation has however seldom been used for textual data; Ogilvie[25] uses stand-off annotation to annotate *text* for grammatical and contextual analysis, and Thompson and McKelvie have used stand-off annotation in a similar fashion. Navarro and Baeza-Yates[23] defined a framework for annotation called *proximal nodes*, which is conceptually stand-off annotation.

DeRose identifies the advantage of standoff markup being the power to express nearly any construct imaginable. According to the paper the main disadvantage of standoff markup is the difficulty to maintain such documents. Another disadvantage of using stand-off annotation for markup of texts from a logical representation perspective is the obvious result that the text is not *inlined* with the markup, and this makes the annotations more difficult to interpret (less readable). Human readable/understandable markup becomes a more abstract annotation. The solution to this problem would be just as simple, using an inline-annotation language such as regular XML, GODDAG, or LMNL, to logically represent the document with inlined markup. The other way around, if binary data is stored inline (images, video, audio, hard disk images), a GODDAG or LMNL becomes unreadable, and the data unusable, because all non-textual characters need to be escaped in XML with `&...;` or `#...;`, and indenting the document with whitespace is not allowed as this would corrupt the binary data. Stand-off annotation leaves the data in its original format, and the binary data is ready for use in the appropriate application. Note that in the above case the logical representation is meant, the physical representation could still be a stand-off approach, preventing the previously mentioned I/O bottleneck.

3.5 Querying

Earlier in this chapter the need for being able to relate different annotations to each other has already been expressed. Querying over stand-off XML documents would require a mechanism to relate annotations to each other based on the available region information. Assume that we already have an XML Query Language such as XPath with which we can define queries within annotations. What we look for is an extension to XPath which can relate the region-annotations to each other. Querying over stand-off XML according to region information is useful when asking questions like: "does file x *contain* the word y?" and "is fragment x *part of* fragment y?". There is currently no suitable solution towards querying overlapping annotations[27], neither for XML dialects nor for stand-off XML annotation. One of the problems in using an annotation language, such as LMNL or stand-off annotation, is that it is difficult to ask questions like "which music is being played during the 'Intro' scene?" If the music and scene annotations are in separate hierarchies, than there is no trivial answer to this query in terms of the regular XPath steps *child*, *parent*, *following*, or *preceding*. Queries like these are not only quite common in video-retrieval, but also in digital forensic analysis: "what data did tool x find at the locations which were marked as possible answers by tool y?" or a more practical example: "which phone numbers have been found in e-mails to person x?". Assume that the phone numbers have been extracted by a different tool than the e-mails, which will cause the annotations to be in separate hierarchies.

Our query approach is based on earlier work by Burkowski[9], but before giving a more in-depth analysis of Burkowski's *Retrieval Command Strings*, first an overview of the available approaches will be given. How the ideas by Burkowski are actually incorporated into stand-off annotation will be explained in the next chapter.

3.5.1 XML Approaches

To get a better grasp of the issues around querying stand-off annotation, one should ask a more general question: "In which way can regions be related to each other?" As long as all regions in a document strictly *contain / are contained by* each other and the regions are *ordered*, there will not be a problem defining parent-child and preceding-following relations. It should in this case be possible to generate an XML document from the annotations, in which region-operations like 'contains', 'is contained by', 'follows', and 'precedes' can be formulated using regular XML query languages like XPath and XQuery. The problem sets in when regions partially overlap or are only partially ordered, because it is difficult or actually impossible

[6] TRECVID: http://www-nlpir.nist.gov/projects/trecvid/, MPEG-7: http://www.mp7c.org/, Acoi

to tell whether the overlapping nodes, or nodes of between which no order is defined, either have a parent-child or a following-sibling relationship.

The only articles to the author's knowledge about querying overlapping annotations in (dialects of) XML are by Iacob et al.[13][18][17] introducing EXPath as XPath extended with additional steps to query GODDAG structures, and Ogilvie[25] overloading XPath. Ogilvie queries his stand-off representation by overloading the descendant XPath-step to be able to query the annotation (similar to the *music* in *scene* query) by asking for `//scene//music`. In this way the '`//`'(descendant)-step does not only select all descendant elements in the XML document, but does also select all regions 'contained by' the context elements. Ogilvie does not state whether partially overlapping regions should be included in the answer or not, and neither does he present an option to select only 'regular' descendant elements, nor only 'region' elements. This makes the solution inadequate for direct use; the semantics of the descendant step should be well-defined.

Iacob et al. have made similar attempts as Ogilvie did, but Ogilvie does not specify the exact behaviour of the newly introduced step, while Iacob miniaturely specifies the possible relations between two regions and proposes *EXPath*, an extension to XPath with a total of 11 new axis steps, which can be used within a GODDAG. The problem with the approach by Iacob is that it is only suitable for GODDAG structures and not for stand-off XML annotation, because no implicit order is defined between two stand-off annotations. It would be possible to order the annotations by their start-end-values, but in that case there still would be no order between annotations with similar start-end-values. This un-orderedness causes ambiguity when it comes to defining a parent-child relationship of annotations[27]. The approach by Iacob et al. is therefore also unsuitable for stand-off annotation.

3.5.2 Generic Approaches

The attempts by Ogilvie and Iacob are aimed at querying concurrent XML hierarchies and stand-off annotation XML documents. Approaches not specifically designed for XML, or even for querying, have been proposed by Allen(1983), Navarro and Baeza-Yates(1997) and Burkowski(1992). The famous paper by Allen[1] distinguishes 13 relations between two regions, but he also requires that regions can be ordered, which assumption we dropped, making his relations inadequate.

3.5.3 Burkowski

Burkowski[9] proposes an 'Algebra for Contiguous Text Extents'. He describes a contiguous *text extent* to be a single word, a phrase comprised of consecutive words or some longer contiguous extent to which can be referred to as *text elements* (chapter, headline, text body, author). These text extents are captured in a hierarchy of concordance lists. A concordance list keeps track of all regions in the data for a particular type of text element.

The concordance lists are used as operands in queries (or *Retrieval Command Strings*(RCS) as Burkowski calls them). Burkowski defines two types of filter operations between two concordance lists: *Select* and *Reject*, both always used in conjunction with *Wide* or *Narrow* (see Definitions 1-4). [maybe copy formal definition from Burkowski?]

Definition 1 (Select Narrow) $[m, n]$ contained by $[p, q] \Leftrightarrow p \leq m \wedge n \leq q$.

Definition 2 (Reject Narrow) $[m, n]$ not contained by $[p, q] \Leftrightarrow \neg(p \leq m \wedge n \leq q)$

Definition 3 (Select Wide) $[m, n]$ contains $[p, q] \Leftrightarrow m \leq p \leq n \vee m \leq q \leq n$

Definition 4 (Reject Wide) $[m, n]$ not contains $[p, q] \Leftrightarrow \neg(m \leq p \leq n \vee m \leq q \leq n)$

An example Retrieval Command String would be:

```
<chapter> SW {"Feynman","virtual particle"}
```

This RCS asks for all 'chapter'-elements containing the word "Feynman" or "virtual particle". Note that concordance lists can be concatenated. Forbes does not use partial overlap in his Command Retrieval Strings, and does neither specify any order between text extents or regions (i.e. it is not possible to ask for the word *before* or *after* "Feynman"). Because the RCS operators do not require order between regions, it makes them applicable in many cases. Burkowski furthermore expands the RCS with additional syntax for ranking and selecting and fetching specific text extents.

Although Burkowski does not define an operation for partial overlapping text extents, we will use the names he gives to the filter operations in the remainder of this thesis. In the next chapter these filter operations will be converted into XPath steps and the semantics of the Select Wide and Reject Wide operations will be extended by also including partial overlapping regions in the result.

Chapter 4

XIRAF Design

> Break on through to the other side, oww!, oh yeah!
>
> *The Doors*

In this chapter the design of XIRAF will be presented. XIRAF has been designed to automate the process of a digital forensic analysis (steps 2 till 5 of the forensic investigation process depicted in Figure 2.1), but does not actually contain forensic analysis tools itself. It rather offers an *infrastructure* for integrating forensic analysis tools. A key aspect of XIRAF is that it hides the numerous interfaces of the different tools from the forensic investigator by offering a single query language (XQuery) interface. The system has been designed as a proof of concept (a prototype). XIRAF has been created in such a way that it is very generic; almost any forensic analysis tool can be added to XIRAF. The system is able to capture and relate the output of these tools. The intended users of the system are the forensic investigators; not only digital forensic investigators, but especially non-digital investigators. Besides the investigation process, XIRAF allows for adding new forensic analysis tools to it, and XIRAF will therefore also be used by tool-wrapping programmers. XIRAF has been designed having in mind the problems and goals, as they are mentioned in Section 2.4. XIRAF has several benefits over a non-integrated set of tools:

1. In XIRAF multiple tools will automatically extract features; the forensic investigator is not required to have knowledge about the individual tool-interfaces.
2. XIRAF has a single query interface, and it keeps dozens of different interfaces hidden from the forensic investigator. A negative side-effect is that XIRAF will keep the nice graphical user interfaces(GUIs) hidden. XIRAF is currently not equiped with an intuitive GUI.
3. Standard investigation questions can be pre-run, and specific investigation scenarios might be programmed.
4. XIRAF uses the selected tools exhaustively, lowering the chance that an object will be forgotten to annotate, and this relieves the forensic investigator from his task of having to keep track of current context. E.g. which tools already have been applied to the current extracted objects.
5. XIRAF can reproduce the actual location of the evidence on the media even after a series of tools that have annotated it.
6. In XIRAF tools only need to be wrapped, not completely rewritten to fit the architecture. This is due to choice of XML as output format, which is very flexible.
7. All annotations are captured in a single document, and not spread out over several databases.
8. XIRAF relies on the capabilities of a DBMS and thereby automatically benefits from technological improvements of that DBMS.

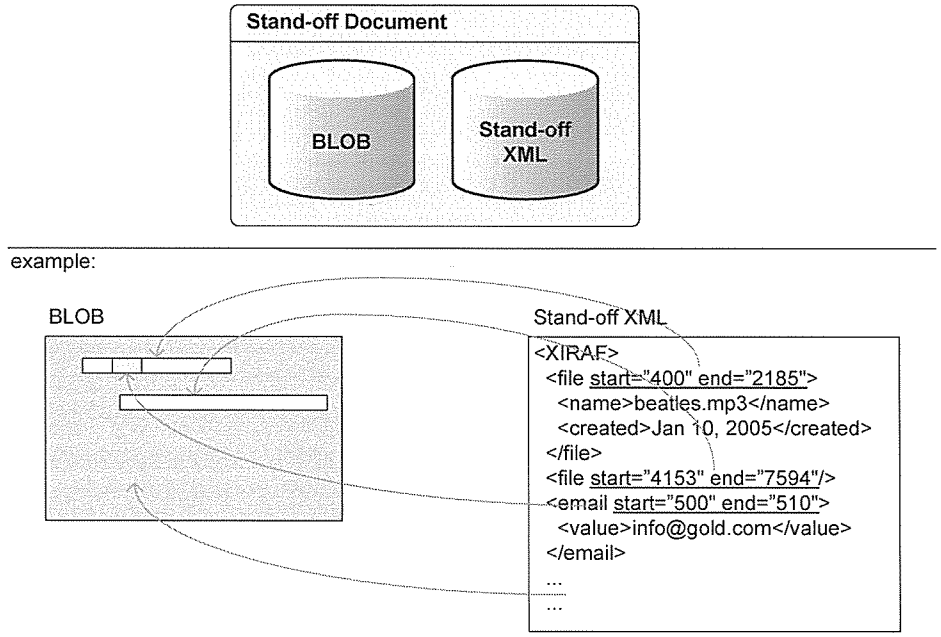


Figure 4.1: A stand-off XML annotation document

Stand-off annotation has been designed as a part of XIRAF, but it is actually applicable in a broader environment (i.e. video information retrieval [34], language processing, and even 'biblical annotation problems'[26]). Stand-off annotation is therefore treated separately from the design of the XIRAF system.

The chapter is structured as follows: first the data model for stand-off annotation will be described (Section 4.1). In Section 4.2 the XIRAF system is shaped. The querying of stand-off annotations is discussed in Section 4.3, and updating stand-off documents in Section 4.4. How XIRAF uses stand-off annotation is answered in Section 4.5. The items that are still left to be designed are listed as future work (Section 4.6).

4.1 Stand-off data model

A stand-off document consists two items: a piece of data called a *Binary Large OBject* and an *XML document*(metadata)(see Figure 4.1). The XML document *annotates* the BLOB. The annotations in the XML document locate objects in the BLOB by defining a region (begin and end byte-offsets within the BLOB) as has been explained in the previous chapter.

The BLOB can be seen as a very long sequence of bytes. The BLOB initially consists of the case data (i.e. the hard disk image). For simplicity reasons the BLOB can be seen as a logical view on the file system of a hard disk (i.e. a concatenation of all files grouped per folder). The XML document initially contains only an XML root element, but during the feature extraction process, the stand-off XML document is gradually filled with annotations describing the BLOB. During the feature extraction process the BLOB might also be modified with newly derived (binary) data. Examples of such data are the contents of ZIP-files or decrypted version of files/messages. This new data logically replaces the old data, but as already stated earlier: physically modifying the BLOB is undesirable. Therefore the data is physically appended to the end of the BLOB. To be able to add annotations to a stand-off XML document and to update the BLOB some update operators are described in Section 4.4.

Users will often only query the annotations and not the actual content in the BLOB, as it is (in the digital forensics case) mostly filled with non-readable binary data. But to give the user the ability to retrieve data from the BLOB, a new XQuery function is hereby introduced:

```
blob(element x)
```

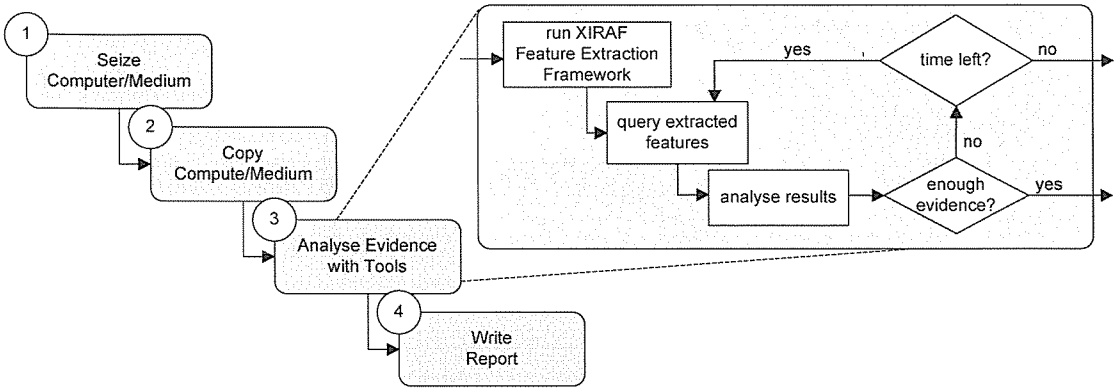


Figure 4.2: An XIRAF Investigation

This function, given a region-element as context, will retrieve the data present in the region. For example

```
<a start="10" end="19"/>/blob()
```

retrieves ten bytes (10 till 19) from the BLOB and returns it as an XML-escaped string. If `/blob()` is used on non-region-nodes the result will be an empty node.

4.2 XIRAF System Design

The XIRAF system consists of a *feature extraction framework*, a *query interface*, and a *tool repository* (see Figure 1.2). The work-flow of a forensic investigation using XIRAF is depicted in Figure 4.2.

A short introduction of the three components is given here before the components are discussed in detail, because the three components are closely related. The *tool repository* manages and stores the *wrappers* for individual forensic analysis tools. For each tool a wrapper consists of an *input descriptor*, telling which objects can be annotated by the tool, and a *tool-executable-wrapper*, hiding the specifics of the tool. The tool-execution-wrapper takes care of invoking the tool when an input object needs to be annotated. The tool-execution-wrapper also takes care of converting the output of the tool to (stand-off) XML. Digital forensic investigations can be initiated in the *feature extraction framework* by creating a new project (a new stand-off XML document). Case data can afterwards be added to the project by inserting new BLOB objects into the stand-off document. Wrapped forensic analysis tools from the tool repository are used to enrich the stand-off XML document. If XIRAF is equipped with a 'good' toolbox (the tool repository is suited for the job), then it can successfully operate on case data. The framework stops processing when there is no single tool anymore that can produce new annotations. When the feature extraction phase is finished, the project can be queried by a forensic investigator using the query interface.

By using the input descriptors the tools are applied to the right objects. All identified objects are stored in a stand-off annotation XML document. The case data itself is kept in the BLOB and references are made from the XML document to the data.

4.2.1 Feature Extraction Framework

The feature extraction framework (Figure 4.3) is the heart of XIRAF; it manages the feature extraction process, by

1. deciding which tool to use next,
2. (a) preparing the data for, (b) invoking, and (c) capturing the output of the selected tool, and
3. adding the new annotations to the database.

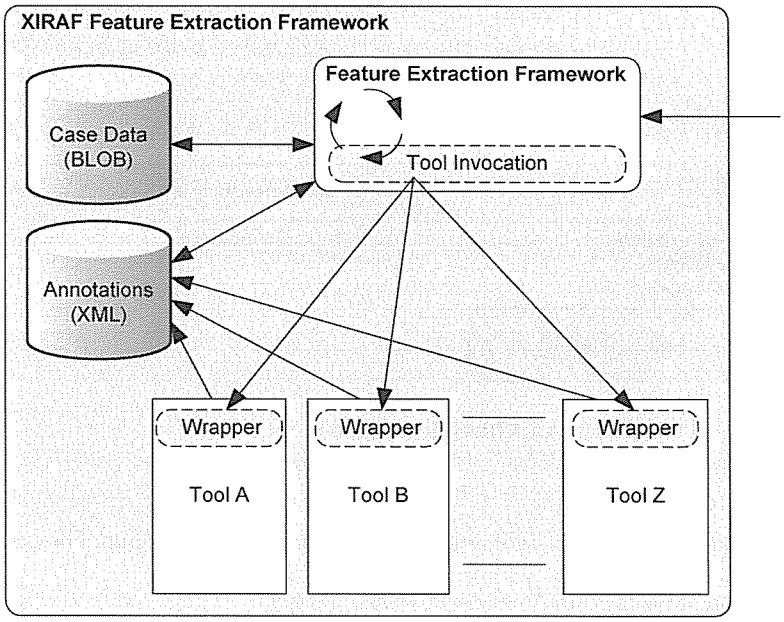


Figure 4.3: XIRAF Feature Extraction Framework

The framework repeats this process until none of the tools can produce additional features. In the second step, the framework prepares the data by executing the input descriptor of the selected tool on the current annotations in the database. When annotatable objects have been found, the next action is to annotate the extracted data using the tool. This is done by invoking the *tool-execution-wrapper* of the wrapped tool. Afterwards the output produced by the tool is captured and added to the database (Figure 4.4).

An input descriptor(see example in Figure 4.5) specifies the type of objects that a tool can annotate, but once a certain tool has annotated a certain object, the tool should not be invoked on the same object again. To prevent recursive annotation, the framework adds a special clause to the input descriptor, limiting the results to only the objects that have not yet been annotated by that tool. This is done by adding a label, containing the name of the tool, to an object after a tool has annotated it, and adding a 'not having label'-clause to the input descriptor of a tool. The additional clause is added automatically as it is the responsibility of the framework and compulsory for each tool. The principle is quite simple, but effective.

Related Work

In other application areas some related work can be found as well, similar to the wash tunnel concept and the XIRAF Feature Extraction Framework. An, on first sight, similar approach is Acoi[40] designed by Windhouwer, Schmidt, Kersten en Nes. Acoi is a feature detection system for indexing World Wide Web content and is especially focussed on indexing multimedia. The *detector* concept presented in Acoi could be compared to tool wrappers in XIRAF and the *detector scripts* can be compared to forensic analysis tools. Some differences between XIRAF and Acoi are depicted in Table 4.1.

4.2.2 Tool Repository

The *tool repository* is the virtual toolbox of XIRAF, in which any number of wrappers for forensic analysis tools can reside. Forensic analysis tools are software programs that search for pieces of evidence (extract features). As identified earlier, almost every forensic tool comes with a different interface. For XIRAF to be able to use those forensic analysis tools, they need to be *wrapped* (i.e. a uniform interface is created). A wrappable forensic analysis tool (a *tool*) can almost be any computer application extracting features, as long as it has some kind of command-line or other programmable interface and an export feature.

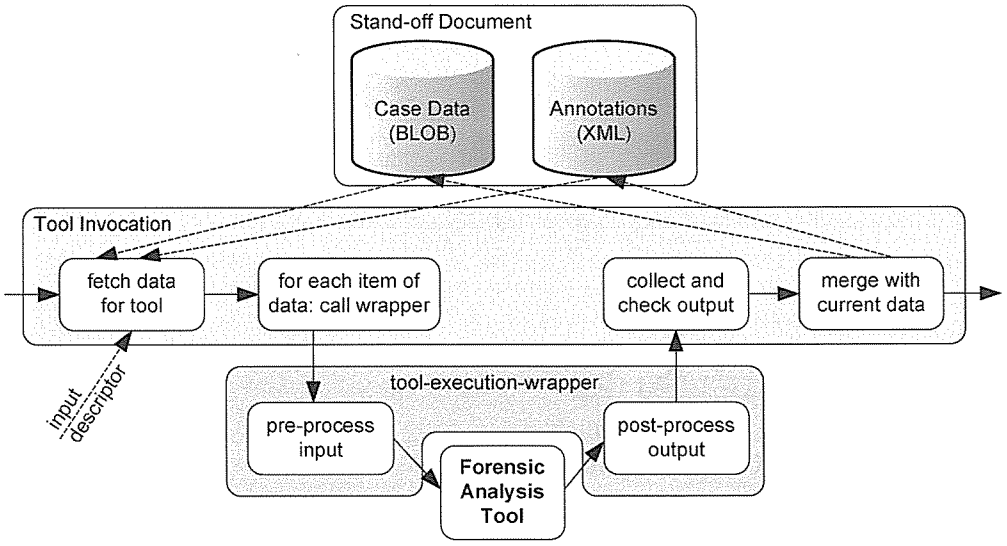


Figure 4.4: XIRAF Tool Invocation

```
$casedata//*[mime="application/zip"]
```

Figure 4.5: An *Input Descriptor* written as (part of) an XQuery requesting all ZIP-files

Acoi	XIRAF
In Acoi the data repository grows continuously	in XIRAF the data is fairly static
In Acoi an object can only have one meaning, if the object is identified as a web-page, it will always remain a web-page and will never become an image	In XIRAF an object can be identified having multiple meanings.
Links are made between objects	Annotation relate through regions

Table 4.1: Differences between Acoi and XIRAF

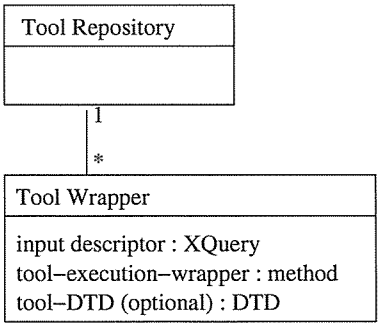


Figure 4.6: XIRAF Tool Wrapper objects in a Tool Repository

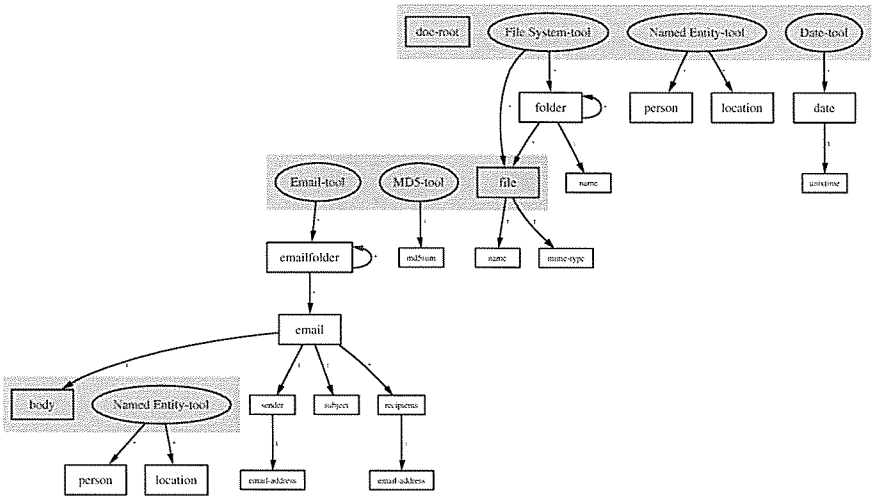


Figure 4.7: example of a tool-dependency graph

The forensic analysis tools can request objects using an *input descriptor*. The input descriptor is an XQuery specifying the annotatable objects (see Figure 4.5). An input descriptor helps XIRAF to find all objects that can be annotated by the tool, without having to try the tool on each piece of data in the BLOB[1].

A tool-execution-wrapper is the wrapper for the tool's executable program(the tool itself). It acts as an intermediar between XIRAF and the tool. The XIRAF framework only has to be able to issue the command 'analyse x'. The wrapper translates this into the tool's specific language and will also capture the output of the tool and format it as stand-off XML.

A wrapper for a tool *should* include a DTD of the output of the tool. When each tool would have a DTD, it should be possible to generate a *tool-dependency-graph*. This can be helpful in many ways. For instance, it can be calculated up front whether a specific set of tools is capable of generating results for a specific query. An example of a tool-dependency-graph is depicted in Figure 4.7. A tool-dependency-graph is beyond the scope of this thesis and will in the remainder of this thesis only be mentioned in the Future Work section.

4.2.3 Query Interface

Annotations can be queried after they have been extracted by the framework. The *query interface* is the interface for forensic investigators to query and assess the annotations.

The XIRAF query interface is currently quite primitive. The investigator can directly issue XQuery statements to a stand-off XML database using the XQuery language. The results will be presented in plain XML.

The query interface (Figure 4.8) allows a user to formulate queries over the features extracted by the tools. The forensic investigator will be aided by *extensions* to the XQuery language, making it possible to relate output from different tools to each other. The XQuery extensions are presented in the next section.

For XIRAF to be useful in real investigations, it will need a Graphical User Interface (GUI) which should be built on top of the current XQuery interface. To allow investigators without knowledge about XQuery to query the database some *query templates* have been created which will aid the forensic investigator with the formulation of queries during forensic investigations. In Appendix C.1 a short list of possible patterns is given and in Appendix C.2 some patterns with more focus on digital investigations are listed.

[1]The process of creating appropriate input descriptors for tools is beyond the scope of this thesis and can probably be analysed best when XIRAF is actually used.

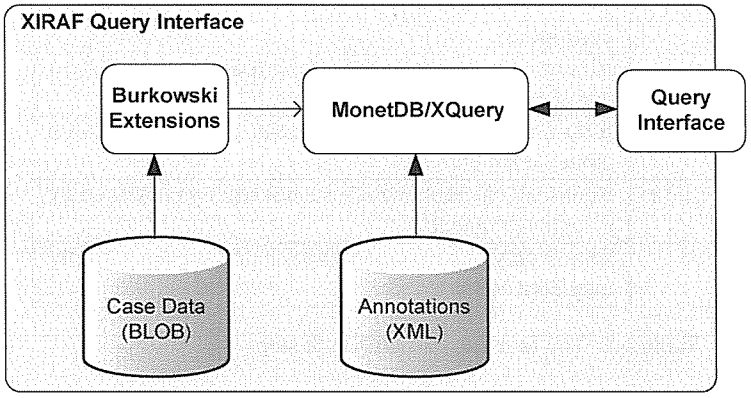


Figure 4.8: XIRAF Query Interface architecture

The tool-dependency-graph, as it has been introduced in the previous section, should make it possible to create a 'smart' user interface; if a tool-dependency-graph could be drawn including tools that have not been used on the case data, it would be possible to know which objects are left to be annotated, and a forensic investigator could be warned if he/she is searching for objects which have not yet been annotated. The implementation of such a warning system is beyond the scope of this thesis and will be left as future work.

Naming Conventions

In a normal XML document the structure can be understood quite well, because either the document has a DTD or the structure is self-explanatory, but when a large number of tools substitute to the creation of a single XML-document, each having its own DTD, it very soon becomes a mess. Therefore the tools should comply to a certain set of rules. For example: the naming of objects in annotations should be consistent throughout the tool repository. An identified problem with wrapping tools is that tools give different names to specific *types* of information; for example features like 'mtime' (date of last modification),'atime' (date of last access), 'ctime' (date of creation) all indicate timestamps. Using different names for timestamps makes it difficult to query for events happening during a specific period. Retrieving all timestamps would require the investigator to list *all* the different names. To overcome this problem, a uniform naming scheme should be used to be able to query the data effectively. For example rules like: all tools *must* annotate any timestamp they find with `<timestamp/>` would simplify querying times and dates considerably. Other named-entities could for example be: `<file/>`, `<folder/>`, `<e-mail-address/>`, `<ip-address/>`, etc. Optionally these entities could have their own type-specific DTD.

4.3 Queryable Stand-off Annotation Design

The contribution of the XIRAF project to the database research field is the querying of stand-off XML annotations. In this section the extensions to the XPath language based on previous work by Burkowski will be introduced. The original ideas of Burkowski, as described in Section 3.5.3, were not meant for XPath, a query language that did not exist at the time (1992) the article was published. The filter operations however inherit many aspects found in regular XPath steps:

- filter operations (steps) accept a given set of extents(context-nodes) and return a set of extents (result-nodes).
- Each extent in a concordance list(node in a document) can only appear once in the result of a filter operation (step).
- extents (nodes) can only be returned from the same set of concordance lists (document)

Burkowski Step:	description:
`/select-narrow::`	Return only those elements which are contained by a context element
`/select-wide::`	Return all elements which at least partially overlap a context element
`/reject-narrow::`	Return all elements which are not strictly contained by a context element
`/reject-wide::`	Return only those elements which do not even partially overlap a context node

Table 4.2: The four Burkowski XPath steps

- in every filter operation (step) a concordance list is selected (nametest is performed) to return extents from.

These reasons give reason to believe that Burkowski's filter operations could be shaped as path-steps into XPath. To recall from the previous chapter the syntax of the filter operations:

```
<title> SN <chapter>
```

which would return only those `title`-extents contained by a `chapter`-extent. If this query would be translated to XPath it would probably look like: first selecting all chapter elements and then selecting those titles within the chapters:

```
//chapter/SN::title
```

in which the arguments have been reversed. The reversal of the arguments can be explained by the fact that XPath-steps always return elements of the last operand while the Burkowski filter operations return the elements of the first operand.

Besides this transformation to XPath, another important and major change in the semantics of Burkowski's original filter operations has been made: in the result of an SW steps *all* overlapping regions are included and, similarly, in a RW *all* overlapping regions are rejected. The reason for this change is that partially overlapping regions were not contained in any of Burkowski's filter operations. It is arguable that `wide` should not be used for querying over partial overlapping regions. The step for querying partial overlap could be given another name. However, an argument in favour of changing the semantics of the `select-wide` and `reject-wide` steps is that in the original approach the expression `a/select-wide::b` could also be formulated using an XPath where-clause and a `select-narrow` step:

```
b[select-narrow::a]
```

and would therefore be redundant. Although this argument does not constitute to the discussion, it is hereby decided that the `wide` filter operation will be overloaded by also including every overlapping extent, which includes partial and contained extents.

The naming of the steps can actually be a little bit more verbose as SN, RN, SW and RW are not really self-explanatory. Therefore the resulting XPath steps extracted from Burkowski's filter operations will become the steps depicted in Table 4.2. Using these new steps the query "all music played in scenes" could be written as the XPath expression:

```
//scene/select-wide::music
```

Note that in regular XPath, taking a descendant-step (`//`) from the document-node of an XML document will yield all the elements in the document. To be able to express queries using Burkowski-steps only, the document-node (the root-node of an XML document) is being defined as the largest possible region. Given this document node a `select-narrow`-step will find all region elements defined in the document:

```
/select-narrow::* == all region elements
```

In a document like in Figure 3.6 it is now possible to ask questions like:

- the music played during the Intro scene:
  `/select-narrow::scene[@id="Intro"]/select-wide::music`
- the scenes within the song by the Beatles:
  `/select-narrow::music[@artist="Beatles"]/select-narrow::scene`
- the scenes in which there is absolutely no music by Bach:
  `/select-narrow::music[@artist="Bach"]/reject-wide::scene`

In Chapter 5 the details of the implementation of the Burkowski join will be revealed.

4.3.1 Comparison to other XPath extensions

Although the introduced Burkowski steps are clearly less complicated than the 11 steps introduced by Iacob et al., the steps are still capable of answering the example questions given by Iacob et al. in Section 2.2 of [18] in which concurrent XML hierarchies are queried:

1. find all damaged characters:
   `/select-narrow::dmg`
2. find all words containing damaged characters:
   `/select-narrow::w[select-wide::dmg]`
3. find all words containing damaged characters ONLY:
   `/select-narrow::dmg/select-narrow::w`
   (not a correct solution, see explanation below)
4. find all damaged characters which have been restored from other manuscripts:
   `/select-narrow::res/select-wide::dmg`
5. find all words containing damaged characters which have been restored from other manuscripts:
   `/select-narrow::w[select-wide::dmg/select-wide::res]`

Query 3 is actually difficult to answer: Iacob et al. do not evaluate the queries with their proposed EXPath steps, but to the author's knowledge the query can neither be answered by the Burkowski steps, nor the EXPath steps. Take the following counter-example (for simplicity expressed in LMNL):

`[doc}aa[dmg}aa[w}bb{dmg][dmg}bb{w]aa{dmg]aa{doc]`

In this example a query like `/select-narrow::dmg/select-narrow::w` will not yield the `w`-element, while `w` only contains data that also has been marked as `dmg`. There is currently no solution towards such queries.

The queries by Ogilvie can also be answered with the Burkowski steps by replacing each instance of the overloaded `descendant`-step with a `select-wide`-step. For example Q15 in [25] would become:

```
/select-wide::sentence[
  ./select-wide::event/select-wide::VBD[
          weigth(0.4 kill 0.3 assassinate
                 0.2 murder 0.1 shoot)]
  AND
  ./select-wide::patient/select-wide::person[
          weight(0.4 'Abraham Lincoln'
                 0.4 'President Lincoln'
                 0.1 'Honest Abe'
                 0.1 'Lincoln')]
  AND
  ./select-wide::agent/select-wide::person]
```

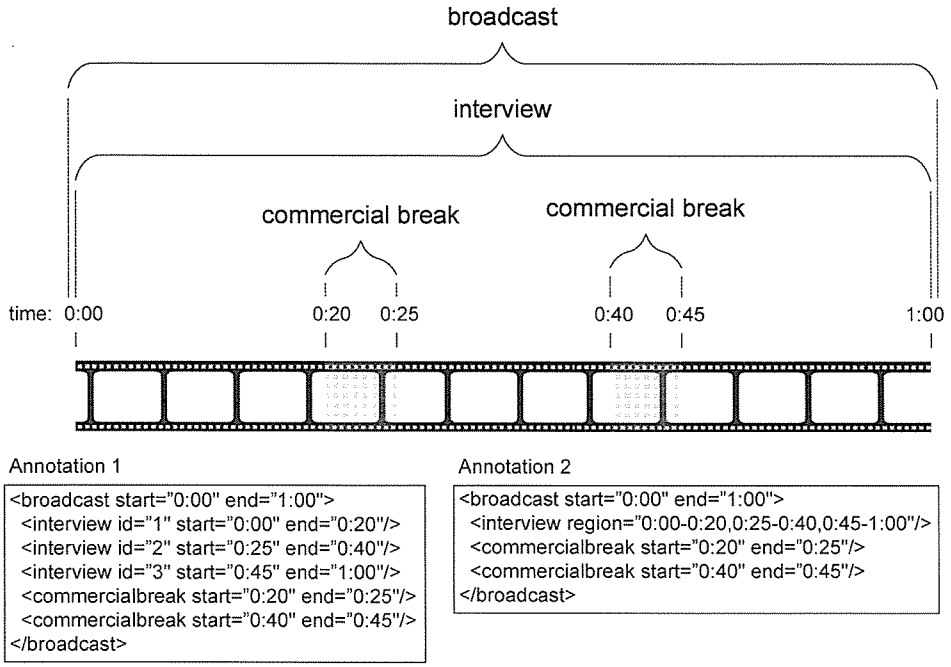


Figure 4.9: A Region as a set of contiguous segments

4.3.2 Representation of regions

Regions have in the previous examples been expressed in terms of `start` and `end` attributes having integer or date/time values (at least something countable). By storing regions as attributes in an XML document they are related to a single element in an XML document. Regions can in this way not be given to a text-node, attribute, processing-instruction or comment. This also means that some XML elements might not have region-attributes, and this automatically implies that these elements cannot be a result of the newly introduced Burkowski XPath steps. The new Burkowski XPath steps can only return XML elements which have region-attributes. The *non-region-nodes* will never be returned as a result of an XPath step. The assumption is that the semantics of the attributes and the steps will not be very difficult to grasp.

The first idea was to use the attributes `start` and `end` for each node to specify a region for an element, but soon after having implemented these regions it became clear that it should be possible for one element to have multiple (`start`, `end`)-tuples. Using `start` and `end` attributes is therefore not a viable solution, because attribute names have to be unique within an XML element.Using a single `region` attribute might be better. For example a region $[10, 200]$ could instead of `start="10" end="100"` be denoted by `region="10-100"`. Denoting a region with multiple segments would be possible using a syntax like `region="10-100,200-210"`.

Figure 4.9 gives an illustration of what happens when a region-type with multiple segments would be introduced. To answer the question whether or not to annotate an element using multiple segments, it can help to ask the question: how many instances of the annotated item can be counted? The answer depends on the implementation and personal preferences. In the example a possible solution would be to say there is only *one* interview and *two* commercial breaks.

The use of multiple segments for a single node should only be used when necessary, because it will be more difficult (will cost more time/indices) to compute query answers. The use of multiple segments is not only useful in digital forensic investigations or annotation of video, but it could also be used in language processing. For example: a 'gezegde' (verb phrase) is not always a contiguous block of text. In forensic investigations this phenomenon happens even more often as files can be 'fragmented', which means the data for the file is stored at several places on the hard disk. The new region representation will not change the semantics of the Burkowski steps and should be easy to introduce, but it is beyond the scope of this

thesis and will be left as a bullet in the Future Work section.

4.4 Updatable Stand-off Annotation Design

As has been expressed in [14], a very vulnerable aspect of stand-off annotation is the synchronization of the stand-off XML document and the BLOB while performing updates. During the feature extracting process annotations are added to the stand-off XML document and new data might be added to the BLOB (for example newly acquired case data) or part of the existing data might be replaced (contents of decompressed ZIP-archives, or decrypted versions of files/message). Updating an *XML* document is an already widely studied subject[37][7], however, updating the XML in combination with a separate BLOB has not been studied at all. UpdateX[37] is an addition to the XQuery language for updating XML documents. The UpdateX language can be used to update the stand-off XML document. UpdateX is currently being implemented for MonetDB/XQuery, the system that will be used as backbone of XIRAF, and UpdateX would therefore be a viable choice to start reasoning with.

To implement a full update language for stand-off annotation all possible types of update operations are needed. However, XIRAF only uses a few types of update operations, and it would be beyond the scope of the thesis to define an update language for stand-off annotation. The basic cases in XIRAF in which a stand-off document is modified are:

case 1 new or initial case data can be added: As order is not defined in the BLOB, the data can in this case just be appended to the BLOB. It might be assumed that insertion of case data is always accompanied with a small XML fragment describing the data. For instance, if a new hard disk image is added, this will most likely be labeled with an identifier, in that after an investigation it can be determined on which physical piece of evidence the relevant annotations were found.

case 2 new annotations can be added: The feature extraction framework constantly adds new annotations. This is the most frequently used operation.

case 3 new data can replace existing data: As previously described, some tools will decrypt or unpack files already present in the case data. The original data will become obsolete (no tools other than decryption or unzipping tools will be able to use encrypted or zipped data).

To avoid synchronization issues, a function should be designed that is capable of updating the BLOB and the XML at the same time, and preferably respecting database properties like Atomicity, Consistency, Isolation and Durability (ACID properties). The minimal update function needed to make XIRAF work is the insertion of a *stand-off-fragment* (a partial stand-off document consisting of a non-empty XML fragment and optionally new BLOB data) at a specific place in the stand-off document.

```
insert(node x, stand-off-fragment y)
```

The `insert()`-function has two parameters: `x` - the position inside the XML document of where the fragment needs to be inserted, and `y` - the new data itself. The function will *insert* the XML part of the stand-off fragment `y` into the stand-off document *and* it will *replace* the region specified by node `x` with the BLOB part of the stand-off-fragment `y`.

In case 1 the `insert()`-function can be used to insert new case data by specifying `x` as a non-region-node. If `x` is a non-region-node, it does not have any BLOB data attached to it, and therefore the replace in the BLOB will become an insert. The location `x` can for example be the root-element of the document[2]. Case 2 is also an insertion, but in this case the stand-off-fragment does not contain new BLOB data, and thereby will not replace any data. The insert-location `x` can in principle be arbitrary, but XIRAF will always require the `x` to be the region-node from which the annotations are derived. Case 3 is always accompanied by a case 2 update, and the only difference between a case 2 and a case 2+3 is that in case 2+3 the BLOB part of the stand-off-fragment will not be empty. The `insert()`-function can be divided in a series of smaller operations:

1. insert an XML fragment (can be done with UpdateX)

[2]The root-element of a document is not the same node as the document-root-node.

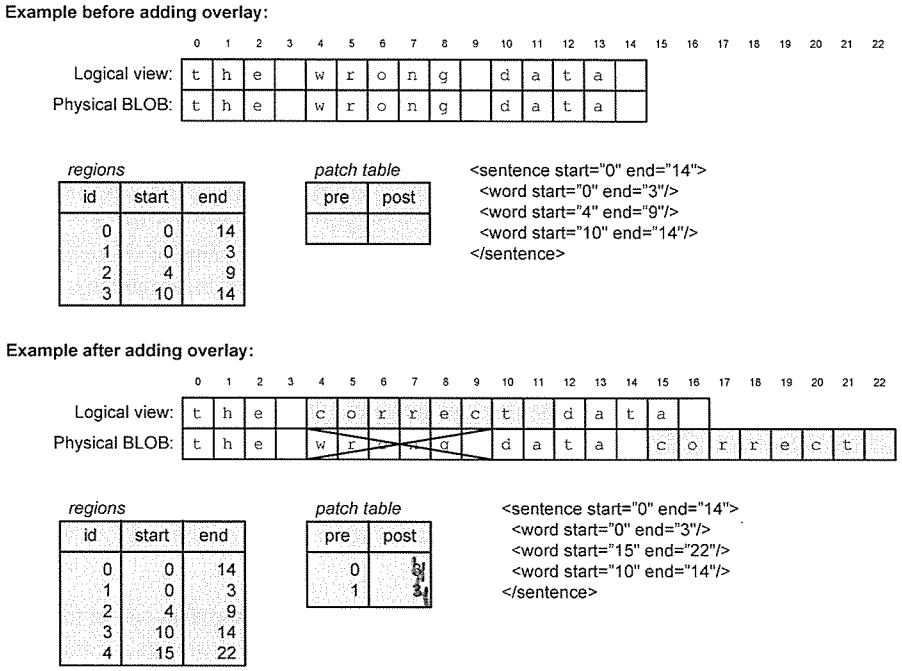


Figure 4.10: Implementation of 'patches' (updates in the BLOB). For illustration purposes an example XML fragment is given with the BLOB. The actual shape of the XML may be defined by the user.

2. recalculate the region-attributes (can be done with UpdateX)
3. insert BLOB data
4. replace BLOB data

Inserting an XML fragment into an XML document and updating attributes with new values can be done with the UpdateX language. It is not possible to manipulate a BLOB using the UpdateX language. One very important aspects is that, although updates to the BLOB should be possible, XIRAF should not allow tools to modify the case data itself as we would like to be able to retrieve the original image after an investigation. To avoid touching the original data in the BLOB, the data could be appended to the end of the BLOB. The original data would thereby not be touched. It would require a logical view separate from the physical view of the BLOB. The logical view should translate the position of the new data in the physical view to the position of the data it replaces. Translating the physical view into a logical view could be done by maintaining a 'patch'-table in which all replaced regions are listed together with their new (physical) region (at the back of the BLOB) (See Figure 4.10). The solution is again quite primitive, but the number of updates is expected to be small and therefore this approach is likely to suffice for XIRAF.

4.5 XIRAF and Stand-off XML

XIRAF uses stand-off annotation to store the features extracted by the forensic analysis tools.

During the process of extracting features (annotating regions), more information about the annotations is known than just the region information. Recall that a tool uses an input descriptor to specify which regions it can annotate. It is therefore known from which parent-annotation the new annotations stem. Thus, a hierarchy of annotations can be created. The definition of stand-off annotation as presented in Chapter 3 is in XIRAF expanded with the notion of a parent-node:

```
annotation = {region, metadata, parent-node}
```

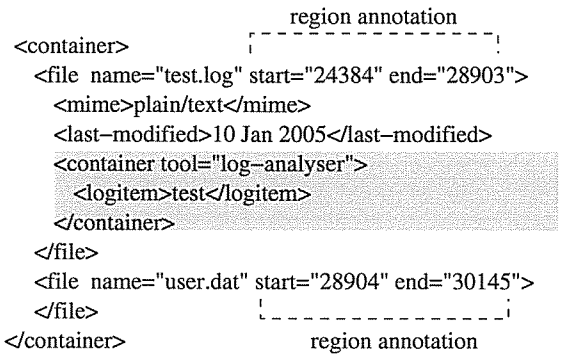


Figure 4.11: Scope of an annotation

When for instance a log-file is annotated by a log-analyser-tool that extracts log-items, those log-item-annotations will have the log-file-annotation as parent and will therefore become children of the log-file annotation. Annotations will as a result be nested within each other, creating a hierarchy. Creating such a hierarchy is commonly referred to as *data lineage*. To separate the annotations in the XML document and to avoid losing information about which tool created the annotation XIRAF introduces *container*-elements to the stand-off document. Container-elements keep annotations from a tool together, and make it possible to combine the output of all tools in one file, while being able to distinguish the annotations from a single tool. Containers are root-elements of an annotation by a tool. An example of the use of containers is given in Figure 4.11.

4.6 Future Work

In the current XIRAF design a few areas for future research are identified:

1. The current XIRAF feature extraction framework uses an exhaustive approach. Future studies might be able to improve the input descriptor idea, or might use time-bounded analysis (do as much useful work in 1 hour as possible).

2. The design of the user interface: How would a forensic investigator like to see the results?

3. generating tool-dependency-graphs and exploiding them. Look into tool-dependency graphs, and try to aid the investigator when querying the database. (Section 4.2.2)

Chapter 5

XIRAF Implementation

"If I had more time, I would have written a shorter letter."

Blaise Pascal

This chapter discusses the implementation details of XIRAF. The implementation phase can be split up in two parts; the extension of MonetDB/XQuery with stand-off annotation and the implementation of the XIRAF system itself. The implementation of the XIRAF system focussed on the feature extraction framework. The parts of the system that got little attention were the actual wrapping of tools and the implementation of an user interface[1]. During the development it has been kept in mind that XIRAF should be robust; it should not crash and it should be able to handle vast quantities data. In Figure 5.1 an overview is given of the architecture of the XIRAF system. The system is split into two parts; the stand-off annotation (querying and updating) and the XIRAF system (framework, tool repository, and query interface).

The chapter is structured as follows: in Section 5.1 the choice for the database system underlying XIRAF is discussed. The implementation of the feature extraction framework and tool repository is presented in Section 5.2. Then the Burkowski steps are described (Section 5.3) and finally an implementation of the stand-off update mechanisms is presented in Section 5.4.

5.1 Platform Choice

The stand-off XML annotation model is to the author's knowledge new to any XML database management system. A database system has been chosen in which the stand-off annotation model could be implemented. At the CWI the TIJAH[21] and MonetDB/XQuery[6] system are being developed, and those two systems were therefore plausible choices to build stand-off annotation on. Both systems are built on MonetDB, a database back end. MonetDB/XQuery is an implementation of the W3C XQuery language recommendation[4], useful for structural database queries. TIJAH is an implementation of the NEXI language[39], which can be used for information retrieval activities such as INEX. The two languages are, except for the fact that they query XML and are derived from XPath, not comparable. MonetDB/XQuery has the advantage over TIJAH of a larger audience, and it has matured very rapidly after its first release in May 2005, while TIJAH is still very experimental software. MonetDB/XQuery follows the XQuery language recommendation as set by the W3C, and applications built on the standard will therefore in the future be more portable to other systems. XIRAF needs a solid database system which can perform structural queries and therefore MonetDB/XQuery has been chosen.

It is planned that in future releases of MonetDB/XQuery the UpdateX language will be added[7]. This improvement would open the door to updating XML documents in a transaction save environment. With this feature, stand-off updates could be more easily and more efficiently implemented in XIRAF. Another

[1] More information on how XIRAF can be used, how tools can be wrapped, and other usage-related info can be found in the user manual presented in Appendix B

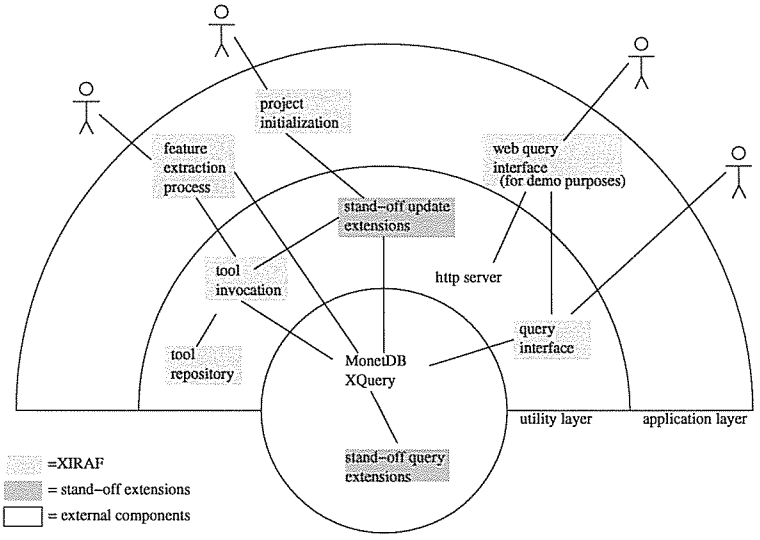


Figure 5.1: Overview of Architecture

reason for using MonetDB/XQuery is that there is no other XML-DBMS available, capable of handling multi-gigabyte documents in interactive time[6].

The IR in XIRAF stand for Information Retrieval, and it should be obvious that the XIRAF system needs IR mechanisms. Actually this problem is not an XIRAF problem, but actually a more fundamental problem of databases[11][31], and several proposals of (semi-)structured information retrieval languages for XML have been proposed: XQuery-fulltext[2], NEXI[39].

Choosing MonetDB/XQuery as the back end for XIRAF might at first seem as a remarkable choice, as MonetDB/XQuery does not have any IR extensions. One of the reasons for this choice is that there are serious plans and efforts already made to integrate TIJAH with MonetDB/XQuery, and in this way XIRAF would gain IR functionality too. However, the current status of MonetDB/XQuery is that all string-handling is very primitive and slow. This deficit has to be accepted for the time being.

5.2 XIRAF

The following components of XIRAF, as designed in Section 4.2 and depicted in Figure 5.1, have been implemented:

1. Feature Extraction Framework :
   (a) project initialization,
   (b) feature extraction process and
   (c) individual tool invocation.
2. Tool Repository
3. Query Interface :
   (a) XQuery interface and
   (b) a Web Query interface designed for demo-purposes.

Initiating a new investigation (a new project) is done by creating a new stand-off document. Each stand-off document has a name and is represented by two files: a BLOB file and an XML file. The XML-file is stored in the database with this name (with the `.xml`-extension added). The root element of the XML-file will contain a `blob`-attribute which contains a pointer to the accompanying BLOB-file of the stand-off document.

5.2.1 Feature Extraction Framework

The feature extraction framework has been implemented in a modular way. Each link in the chain (Figure 4.4) of the feature extraction process has become an individual (small) program, solely designed for its purpose. Wherever possible XQueries were used to transform and query XML documents, but sometimes an XQuery could not be formulated or would be too complex to be executed in reasonable time.

Feature Extraction Process

The feature extraction process is a script to automatically (and exhaustively) run a set of tools. The script is implemented as follows:

1. for each tool in tool repository: - invoke tool
2. fetch the stand-off XML document
3. compare the fetched document against the stored document
4. if the fetched document is equal to the locally stored document
   – quit
   otherwise
   – replace the local document with the fetched document and repeat from step 1

Project Initialization

The project initialization script creates a new project (a stand-off document with a root-element) and adds it to the database. Afterwards, a script is called which adds the initial case data to the project (inserting a BLOB + initial XML). This script is currently needed to import case data, but it will become obsolete in future versions of XIRAF, when adding case data will be just one of the wrapped tools in the repository.

Tool Invocation

Tools are invoked by a script. Given a tool from the repository and a stand-off document this script:

1. fetches the input descriptor of the tool
2. adds a clause to the input descriptor to retrieve only those objects not yet annotated
3. runs input descriptor on the stand-off document in the database
4. for each region in the result: extracts the data from the BLOB document
5. for each data object: invokes the tool-execution-wrapper
6. adds the new annotations to the stand-off document in the database

At the moment the feature extraction framework does only provide BLOB data as input for the tool-wrappers as most tools require this 'raw' data as input. In a future version of XIRAF the tool-wrappers should be provided with the XML part as well.

5.2.2 Tool Repository

The tool repository is currently implemented as a folder containing tool-wrappers. Tool-wrappers are currently implemented as a folder containing (at least) these two files:

id.xq The 'Input Descriptor'-file containing an XQuery specifying the objects the tool is able to annotate.

tool.sh The tool-executable-wrapper.

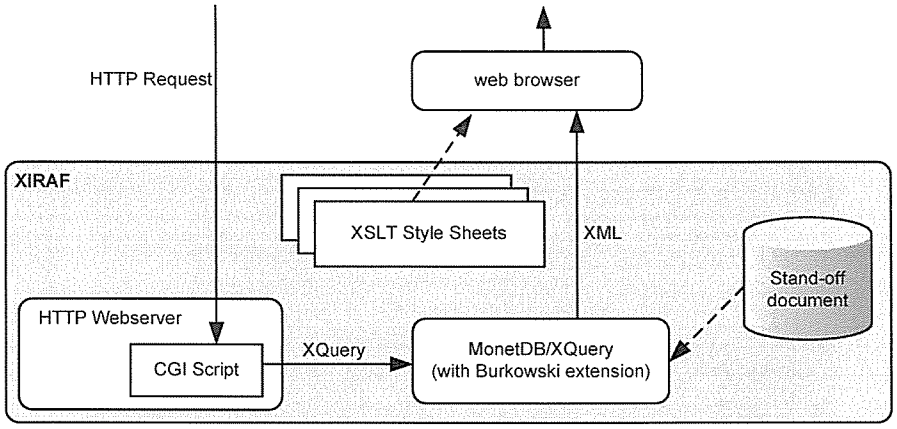


Figure 5.2: Architecture for Web Interface

The tool-execution-wrapper will be invoked by XIRAF with the following parameters:

```
tool.sh XML-input BLOB-input XML-output BLOB-output
```

in which `input` is the location of where the input data can be found and `output` is the location of where the new annotations and BLOB data can be written to. A detailed explanation of how to wrap tools can be found in XIRAF manual (see Appendix B).

5.2.3 Query Interface

XQuery Interface

The current user interface for XIRAF is the XQuery front-end of MonetDB/XQuery with the XIRAF extensions. A forensic investigator can formulate an XQuery (optionally using the extensions) and view the results as XML. There are currently no ranking or other Information Retrieval techniques available. Some XQuery patterns have been created to aid the forensic investigator (see Appendix C.1).

Web Query Interface

To abstract from the XQuery language a web-interface has been designed (see Figure 5.2). This web-interface makes it possible for forensic investigators to almost 'Google' the case data, without having to know XQuery or another query lanuage. The web interface encapsulates XQuery statements by providing simple HTML-forms to the user. The resulting XML from the queries is then formatted using *XSL* stylesheets. In Figure 5.3 a screenshot of the web interface is presented. The XQuery and XSLT languages are quite similar, they both accept and return XML, but there are substantial differences; XSLT is more a find-and-replace language, suitable for small documents. XQuery is more suitable for large documents and is more a database query language. Both have their own (dis)advantages. A more detailed comparison of the two languages is given by Kay[19].

5.3 Querying Stand-off Annotation

In this section the implementation of the Burkowski Steps will be presented. To do so, first a small intro to MonetDB/XQuery will be given to get an idea of how the system internally works. Then the changes to the internal structure of MonetDB/XQuery are elaborated on. Finally some optimalizations are applied to the new steps to let them respond in interactive time for multi-gigabyte documents, just like the other axis-steps in MonetDB/XQuery. The performance evaluation for the steps is presented in the next chapter.

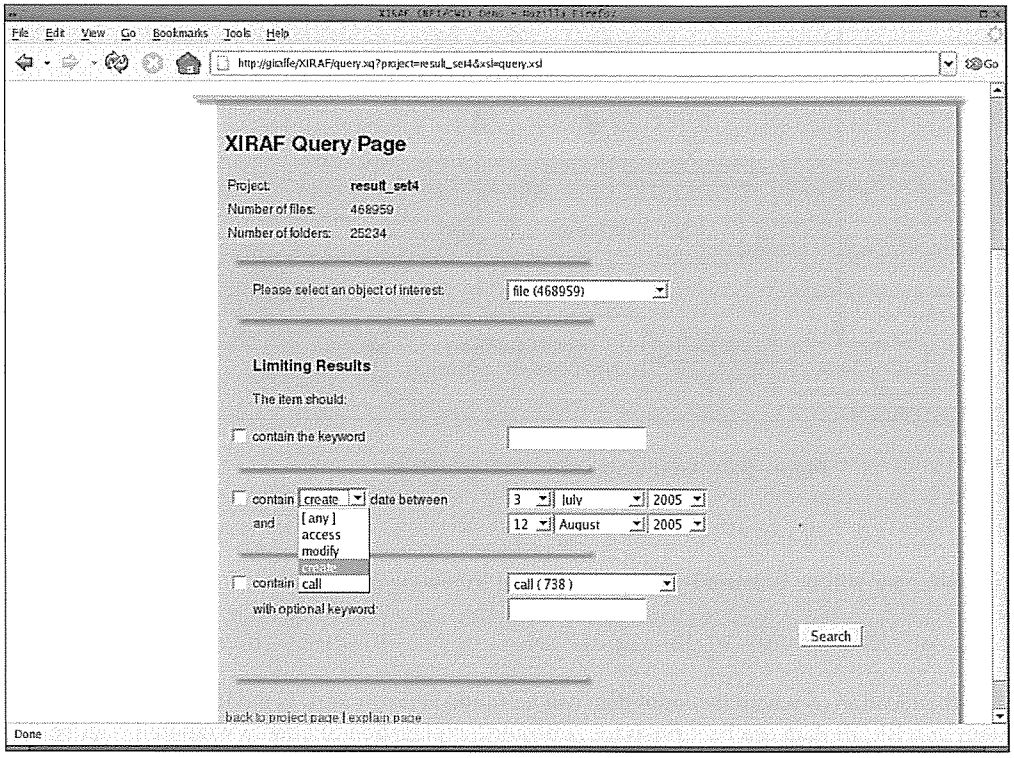


Figure 5.3: Screen shot of the web interface for the XIRAF demo

MonetDB/XQuery

A little introduction into the MonetDB/XQuery internals: MonetDB/XQuery[2] is based on the Pathfinder XQuery compiler[3]. MonetDB/XQuery has a typical front-end/back end design: the Pathfinder front-end is a compiler that translates XQuery statements into the relational Monet Interpreter Language(MIL). The MIL statements are then executed on the MonetDB server. Pathfinder is a typical compiler with a scanner, parser, internal tree representation, optimizer: which recognizes joins, and a code generator. MonetDB is a database back end interpreting the MIL statements. MonetDB uses Binary Association Tables (BATs) as its primary storage structure. BATs are binary relations. MonetDB provides an internal language (MIL) for fast and simple database operations on these BATs such as: `join`, `select`, `project`, `group`, etc.

Besides executing queries, MonetDB/XQuery can store XML documents persistently by *shredding* them into *working set containers*. A working set container is an internal relational representation of an XML-document, consisting of a set of BATs. It uses a `pre/level/size` numbering scheme to represent the XML document. The `pre` is the ordering in which the elements in a XML document appear. The `level` is the depth at which the element appears and the `size` is the number of elements that the element contains. Attributes are stored as a `name-value`-pair accompanied by the `pre`-value of the corresponding element. Because MonetDB/XQuery can only handle binary relations, each column in these relations is stored as a (row-number, value)-tuple, and because the row-number is always a dense column, it becomes a virtual column (called a *void*-column) in MonetDB consuming zero bytes. In case of the `pre/level/size`-table, the row-number is equal to the `pre`-value and therefore this column is not physically stored. A simplified working set container is depicted in Figure 5.4.

[2]http://monetdb-xquery.org/
[3]http://pathfinder-xquery.org/

```
<a start="1" end="20">
    <b start="4" end="8">
        <c f="g"/>
        <d start="12" end="14"/>
    </b>
    text node
    <e start="6" end="10"/>
</a>
```

Document Structure

pre	level	size	type	value
0	-1	6	doc	-
1	0	5	elem	"a"
2	1	2	elem	"b"
3	2	0	elem	"c"
4	2	0	elem	"d"
5	1	0	text	"text node"
6	1	0	elem	"e"

Attributes

pre	name	value
1	"start"	"1"
1	"end"	"20"
2	"start"	"4"
2	"end"	"8"
3	"f"	"g"
4	"start"	"12"
4	"end"	"14"
6	"start"	"6"
6	"end"	"10"

Figure 5.4: (Simplified) internal representation of an XML document in MonetDB/XQuery

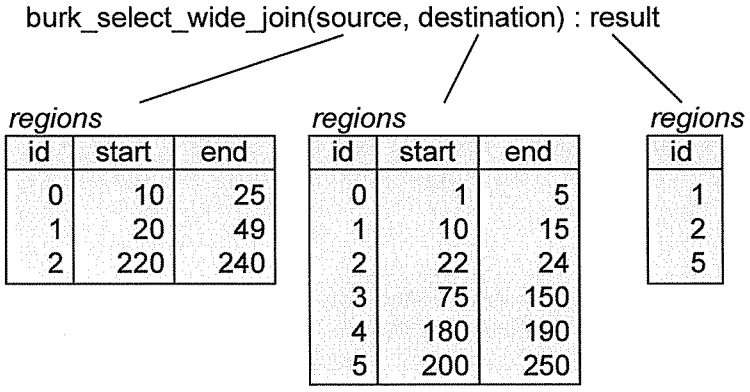


Figure 5.5: MIL Burkowski Join

5.3.1 Burkowski Steps

The steps designed in Chapter 4 have been implemented in MonetDB/XQuery. To be able to do so, a new MIL-primitive, the *Burkowski Join* has been added to MonetDB. Afterwards, the Pathfinder compiler has been modified in that it recognizes the Burkowski-steps in an XQuery expression, and generates the appropriate MIL-code. This MIL code uses the Burkowski Join to calculate the node result set. To make the step more efficient an extra index has been added to the internal representation of an XML document in the database. The exact implementation of these changes will be discussed in this section.

Burkowski Joins (MIL)

The MIL primitive's signature of the Burkowski Join has been depicted in Figure 5.5. It accepts a set of *context* regions (pairs of `start`- and `end`-values) and a set of *candidate* regions (pairs of `region`-, `start`- and `end`-values). Both of these lists should be ordered on the `start`-value. Furthermore it is given that for each item a in the context and candidates list $a_{start} <= a_{end}$. The result will contain only the region-values of the matching candidate nodes. To recall from the previous chapter: `select-narrow` should only return those candidates that are contained in one of the context-regions, and `select-wide` should return all those candidates that overlap a context region at least partially. In Figure 5.6 pseudo-code for the select-narrow join is given. The other joins are fairly similar. If the code on lines 15-29 is replaced with:

```
/* loop through the not even contained regions */
while(j < |inner| && inner[j].end < outer[i].start) {
    /* skip nodes which are not contained */
```

```
 1 [region] CMDselect_narrow_join([start,end] context,
 2                          [region,start,end] candidates) {
 3   inner[] <- candidates;
 4   outer[] <- context;
 5   result[] <- empty;
 6   i <- 0;
 7   j <- 0;
 8   while (i < |outer|) {
 9       next_i <- i + 1;
10       /* skip self overlapping regions in the source list */
11       while(next_i < |outer| && outer[next_i].end <= outer[i].end)
12           next_i++;
13       if (next_i == |outer|)
14           outer[next_i].start <- MAX_int;
15       /* loop through the not strictly contained regions */
16       while(j < |inner| && inner[j].start < outer[i].start) {
17           /* skip nodes which are not contained */
18           j++;
19       }
20       /* loop through the strictly contained regions */
21       while(j < |inner| && inner[j].start < outer[next_i].start) {
22           if (inner[j].end <= outer[i].end) {
23               /* insert nodes which are contained */
24               result.insert(inner[j]);
25           } else {
26               /* skip nodes which are not contained */
27           }
28           j++;
29       }
30       i <- next_i;
31   }
32   return result;
33 }
```

Figure 5.6: Non loop-lifted pseudo-code for Burkowski's Select-Narrow join

```
        j++;
    }
    /* loop through the partly contained regions */
    while(j < |inner| && inner[j].start <= outer[i].end) {
        if (inner[j].end >= outer[i].start) {
            /* insert nodes which are contained */
            result.insert(inner[j]);
        } else {
            /* skip nodes which are not contained */
        }
        j++;
    }
```

the pseudo-code becomes a select-wide join. Note that except for the comments only lines 16, 21 and 22 are actually different. The reject-narrow join can be constructed by taking the select-narrow join and swapping the inserting and skipping of nodes by replacing lines 15-29 with:

```
    /* loop through the not strictly contained regions */
    while(j < |inner| && inner[j].start < outer[i].start) {
        /* insert nodes which are not contained */
        result.insert(inner[j]);
        j++;
    }
    /* loop through the strictly contained regions */
    while(j < |inner| && inner[j].start < outer[next_i].start) {
        if (inner[j].end <= outer[i].end) {
            /* skip nodes which are contained */
        } else {
            /* insert nodes which are not contained */
            result.insert(inner[j]);
        }
        j++;
    }
```

The reject-wide join is constructed by taking the select-wide join and swapping the skips and inserts by replacing lines 15-29 with:

```
    /* loop through the not even contained regions */
    while(j < |inner| && inner[j].end < outer[i].start) {
        /* insert nodes which are not contained */
        result.insert(inner[j]);
        j++;
    }
    /* loop through the partly contained regions */
    while(j < |inner| && inner[j].start <= outer[i].end) {
        if (inner[j].end >= outer[i].start) {
            /* skip nodes which are contained */
        } else {
            /* insert nodes which are not contained */
            result.insert(inner[j]);
        }
        j++;
    }
```

Although the proof has been left out, it should not be difficult to proof that the provided algorithms will finish in $O(|context| + |candidates|)$ time and consume only a small constant amount of memory.

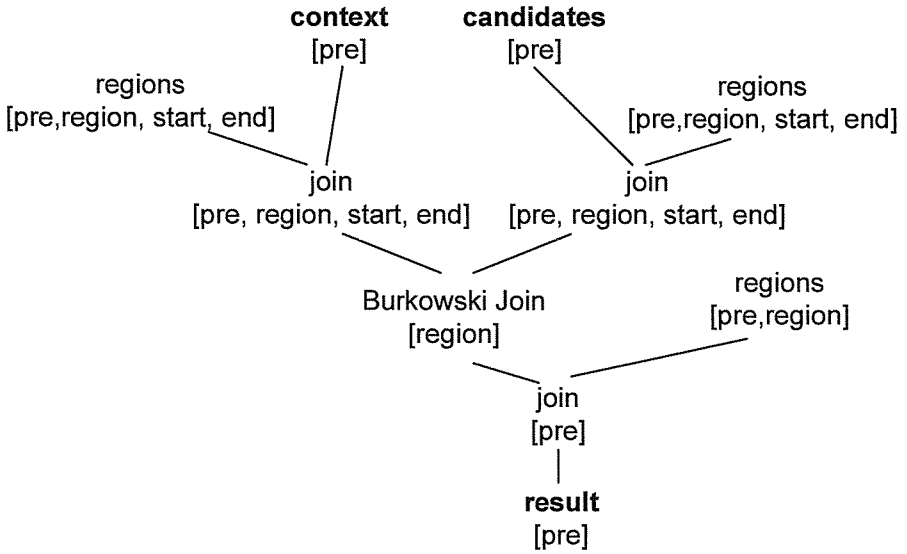


Figure 5.7: Burkowski Query Execution Plan

Burkowski Steps (XPath)

The Burkowski Join MIL-functions have been used to implement new steps in MonetDB/XQuery. The new XPath steps (`/select-narrow::`, `/select-wide::`, `/reject-narrow::` and `/reject-wide::`) have been added to the parser and scanner and generate their own code. Pathfinder interprets the Burkowski steps as regular XPath steps. In Figure 5.7 the query execution plan for a Burkowski step is drawn. The Burkowski Join needs both input lists sorted on the start-values. By using an order-preserving join for fetching the context and candidate regions, a sort operation for both region-lists can be avoided.

Optimization

Initially the region attributes were extracted from the XML document on the fly (at the time of querying). The steps can be optimized in a few aspects. The regions in a document can be pre-calculated (creation of indices) Because Pathfinder identifies the steps as regular XPath steps, the implementation of the Burkowski Steps can be boosted by implementing a *loop-lifted* version of the steps.

Maintaining a region Index To speed up the step, a pre-calculated list of all regions in an XML document is stored. This *region-index* can be created at the time the document is loaded into the database (during shredding). In Figure 5.8 an index is presented that has been added to the internal storage schema for an XML document. The index on the regions is sorted on start-offset. This index is created at the time the document is added (shredded) to the database and is stored in separate *BAT*s[4]. in the *working set container* of a document. The added BATs are `region_pre`, `region_start`, and `region_end`.

Candidate Matching The candidate nodes can be used to downsize intermediate results in a query by providing a list of possible results (candidates) for the XPath step by for example retrieving the list of nodes that match the name-test of the step. Normally the candidate nodes are selected from the result *after* the step has been taken, but if the number of candidates is relatively small compared to the total number of nodes that are the result of the query, it will help to first select those regions that are possible candidates and afterwards perform the step.

[4]BATs are the internal data structures of MonetDB

Regions

region	pre	start	end
0	1	1	20
1	2	4	8
2	6	6	10
3	4	12	14

Figure 5.8: Decomposition of internal XML document representation in MonetDB/XQuery with an index for the Burkowski Step

Loop Lifting In XQuery, results are often not a sequence of nodes, but a set of sequences. In XQuery nested loops are frequently occuring patterns. If there are m loops over an average of n source regions and p destination regions, the naive implementation of the Burkowski step will work in $O(m * (n + p))$ time. By loop-lifting the steps, this time can be lowered to $O(m * n + p)$.

Therefore a loop-lifted version of the Burkowski join has been implemented which operates in near-linear time (approximately $O(m * n + p)$) if measured against the total number of input regions. The loop-lifted version can handle multiple input sequences in a single scan through the destination-elements. This loop-lifting is done by maintaining a stack of possible candidates in the source-elements while walking through the destination elements. The details of this implementation are omitted. [5]

5.3.2 BLOB access

A version of `/blob()` has been implemented. For a set of nodes it will only scan the BLOB once, and return all the requested strings. It is still slow, it removes non-textual characters from the strings and it trims all strings longer than 64KB, to avoid having to cough up the whole BLOB in a mal-formed query. The `/blob()` operation will first fetch the regions, then it will retrieve the data and will finally return the result as a list of strings. A loop-lifted version of the operation has not yet been implemented.

5.4 Stand-off Annotation Updates

This section explains how updates in stand-off XML documents are implemented in XIRAF. The issues introduced in the previous chapter will be solved, or actually a temporary work-around will be presented. MonetDB/XQuery will soon be extended with an UpdateX implementation (planned introduction: 1^{st} of November)[7]. Unfortunately this is too late to be used for the XIRAF prototype. Stand-off document handling is in the current XIRAF system far from complete and certainly not efficiently implemented A very important aspect of stand-off annotation, as already mentioned in Chapter 4, is that the BLOB and the stand-off XML document need to be kept synchronized. This can only be done if the ACID properties are guaranteed in the stand-off document (XML document *and* BLOB).

Create stand-off document

Creating a new stand-off document, provided an initial XML document and a location to store the future BLOB, is quite straight-forward:

1. add `blob`-attribute to document
2. shred the XML document into the database
3. create a repository for the BLOB (`mkdir`)

[5] (Documented) source-code can be found in the file `sjoin.mx` in the `xml`-package of MonetDB.

Add stand-off-fragment to stand-off document

Adding a stand-off-fragment or a set of stand-off-fragments to a stand-off document, will involve the following operations (in a pessimistic approach):

1. locate the position the fragment needs to be inserted
2. recalculate the region offsets to match the stand-off document
3. insert the XML part of the fragment
4. append the BLOB part of the fragment (only if present)
5. create a new item in the patch-table

Locating the position of the fragment should be done by finding the parent-annotation where the new annotations stem from. The location of the parent-annotation is known at the time the input-descriptor runs (before the tool starts annotating), but the XML-document might be modified in the meantime. Therefore positions in XIRAF are retrieved by using unique identifiers, which are given as attributes to each element in the XML document. These identifiers will keep pointing to the same element, even if the XML document has been modified in the meantime. If the identifier does not exist anymore (the element has been deleted), then the insert operation will be rolled back.

In the XML fragment (which are the resulting XML structures the tools produce) the regions will have relative offsets. These offsets are translated to absolute offsets, but calculating the absolute offsets can only be done if the absolute location of the part of BLOB that is being annotated is known. If the location and the absolute offsets have been determined, the XML-part of the fragment is inserted in the stand-off XML document. The BLOB-part of the fragment is appended to the BLOB and an entry in the patch-table is added. Although the above described update mechanism still has many flaws, it is sufficient for the XIRAF prototype using a single-threaded feature extraction process.

5.5 Future Work

While implementing XIRAF and the stand-off document extensions, some issues were encountered and a few items were left un-implemented.

1. Updates in stand-off annotation documents are very important for XIRAF. Improving the stand-off updating mechanisms should be high on the priority list.
2. Currently all input-files for a tool are physically extracted from the BLOB. It would dramatically improve performance if a kind of file-system-driver would be available which extracts the by the input descriptor requested data for a tool when it is actually used and avoids the materialization of non-used data.

Chapter 6

Evaluation

To answer the research questions given in Chapter 1, it is necessary to evaluate the XIRAF system. The *efficiency* experiments were done to measure the performance of the Burkowski steps, the robustness of the *feature extraction framework* has been tested, and the expressiveness of XQuery in combination with the new steps has been analysed using the author's personal hard disk and some sample data from the NFI. The extracted features were queried using the example questions formulated in this thesis. The chapter is structured as follows: first the Burkowski steps are analysed (Section 6.1). Then, the framework is evaluated in Section 6.2, and finally the expressiveness of the query language is measured in Section 6.3.

6.1 Efficiency

The efficiency of querying stand-off annotation using the Burkowski steps is analysed in this section. The main questions are: Is it possible to query over large XML documents using the Burkowski steps? And what influence did the optimization iterations have on the performance of the steps? Because the steps inherited many aspects of regular XPath steps they are compared with regular XPath steps in terms of performance.

Test Setup

The test-data consists of a set of five modified XMark[1] documents, varying in size from 11MB to 1100MB. The modified XMark documents were generated by taking original XMark documents and separating the content from the annotations and adding `start`- and `end`-attributes to the elements referring to the location of the content in the BLOB. To make the test a little more realistic, the annotations were stored in a different tree structure as in the original document; the child-parent relations between elements were obfuscated, thereby simulating concurrent XML hierarchies. In Table 6.1 the generated test sets are shown. The total size of a modified XMark document (XML+BLOB) is roughly 1.5 times the size of the original. The XML-part is only marginally smaller than the original XMark document. While at first sight one would assume that the size of the XML document and the BLOB together would be approximately equal to the

[1] More information about the XMark benchmark can be found at http://www.xml-benchmark.org/.

XMark	Modified XMark Doc			Time to shred (s)	
XML (MB)	XML (MB)	BLOB (MB)	# nodes x 10^6	Original	Modified
11	9.9	7.8	0.165	1.092	0.965
55	50.8	38.9	0.819	2.959	4.717
110	102.7	78.1	1.638	5.566	9.570
550	533.6	391.9	8.214	29.061	80.880
1100	1077.9	783.8	16.426	66.834	257.619

Table 6.1: Size (BLOB and XML) and shredding times for the original and modified XMark document in various sizes

original document, it is not. This is due to the attributes added to the stand-off XML document. Shredding the modified XMark document into the database takes longer than the original document. This can be explained by the fact that MonetDB/XQuery is slower when shredding attributes.

Four of the XMark benchmark queries (Q1,Q2,Q6 and Q7) have been rewritten by replacing the steps in the queries as Burkowski steps (select-wide for parent and ancestor steps and select-narrow for child-steps). However, not all queries from the XMark benchmark suite could be rewritten using the Burkowski steps, because most queries use `/text()` or `/data()` and the stand-off XML equivalent `/blob()` was not available at the moment of evaluation. The original queries together with the modified queries are provided below.

Q1 Original:

```
for $b in doc("xmark.xml")
    /site/people/person[@id="person0"]
return $b
```

Modified:

```
for $b in doc("xmark.burk.xml")
    //site/select-narrow::people/select-narrow::person[@id="person0"]
return $b
```

Q2 Original:

```
for $b in doc("xmark.xml")/site/open_auctions/open_auction
return <increase> { $b/bidder[1]/increase } </increase>
```

Modified:

```
  for $b in doc("xmark.burk.xml")
      //site/select-narrow::open_auctions/select-narrow::open_auction
  return <increase>{
    $b/select-narrow::bidder[1]/select-narrow::increase
  }</increase>
```

Q6 Original:

```
for $b in doc("xmark.xml")/site/regions
return count($b//item)
```

Modified:

```
for $b in doc("xmark.burk.xml")//site/select-narrow::regions
return count($b/select-narrow::item)
```

Q7 Original:

```
for $p in doc("xmark.xml")/site
return count($p//description)
     + count($p//annotation)
     + count($p//email)
```

Modified:

```
for $p in doc("xmark.burk.xml")//site
return count($p/select-narrow::description)
     + count($p/select-narrow::annotation)
     + count($p/select-narrow::email)
```

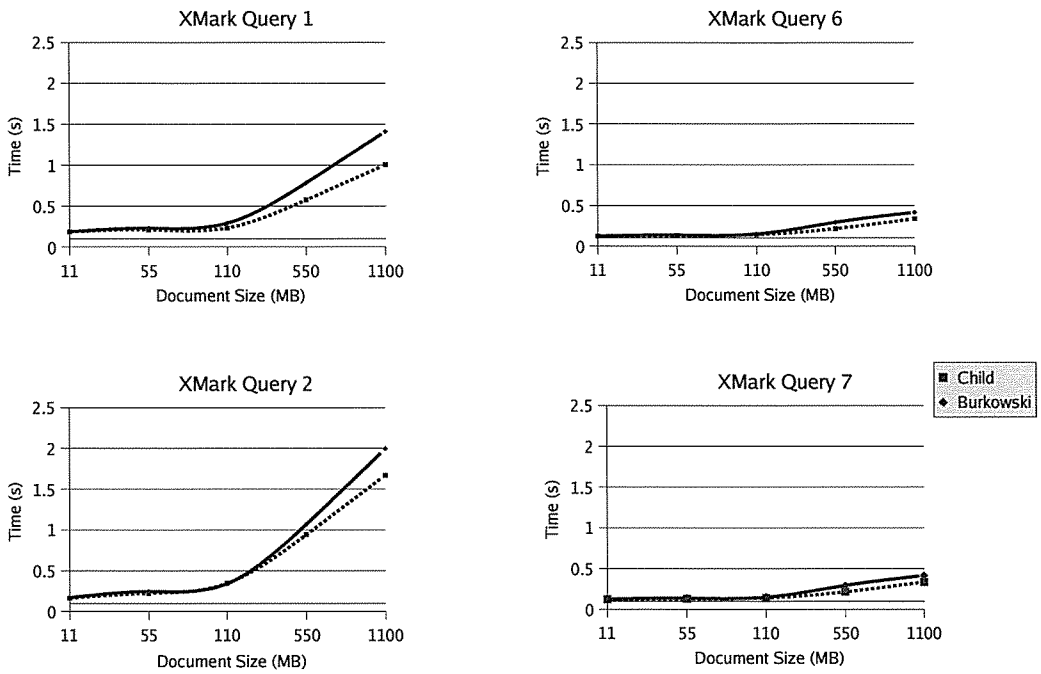


Figure 6.1: Results for regular XMark queries 1, 2, 6 and 7 compared to the results for the modified queries using loop-lifted Burkowski Steps

	Q1 non-ll	ll	Q2 non-ll	ll
11MB	0.15	0.12	5.2	0.14
55MB	0.21	0.17	107.8	0.24
110MB	0.29	0.27	DNF	0.47
550MB	0.93	0.89	DNF	1.53
1100MB	6.73	7.09	DNF	2.46

Figure 6.2: Difference between loop-lifted and non-loop-lifted Burkowski Steps on two modified XMark queries 1 and 2 (DNF = Did Not Finish)

Query 2 of the XMark benchmark will now be analysed in more detail. To get a better understanding of the figures presented in the next paragraphs a short introduction into the XMark benchmark document structure is given here. The XMark documents are constructed with the following amount of elements; in every modified document there is a single `site`-element, and a single `open_auctions`-element. Furthermore, in the 11MB document there are 7217 `open_auction`-elements, 6182 `bidder`-elements, and 6182 `increase`-elements. These numbers linearly increase for the larger documents.

6.1.1 Test Results

In Figure 6.1 experimental results from querying over a modified XMark document are presented. In Chapter 5 it was already assumed that the loop-lifted Burkowski Step should perform quite decently on many queries. Decent in the sense that the execution time scales linearly with the child and descendent step. The graphs show that the Burkowski Step actually does perform as expected; a little slower than a regular child-step, but not by a factor two. The difference between the loop-lifted version and the non-loop-lifted version is shown in Figure 6.2. Note that not only Query 2 profits from the loop-lifted implementation, and Query 1 is in fact (marginally) slower. This is (probably) due to the cost of maintaining the stack.

Q2 (msec)	11MB	55MB	110MB	550MB	1100MB
Total Query Execution Time	169	246	345	1075	1991
/select-narrow::open_auctions	0.641	0.662	0.624	0.700	0.655
/select-narrow::open_auction	1.222	3.074	5.719	27.184	52.479
/select-narrow::bidder	3.885	16.437	31.934	162.064	318.001
/select-narrow::increase	3.592	14.190	27.170	133.007	262.083

Table 6.2: A profile of the Burkowski Steps in Q2 of the XMark Benchmark

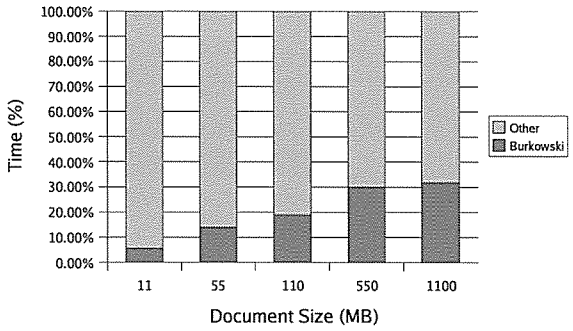


Figure 6.3: Profile of XMark Query 2 showing the actual time spend performing the Burkowski Steps compared to the Total Query Execution Times

Profiling the XQuery

To get a better grasp of the Burkowski steps, XMark Q2 has been analysed in more detail (Table 6.2). In the table the total execution time as well as the total execution time per step are shown for the various modified XMark documents. The 'Total Query Execution Time' includes parsing, scanning, MIL-code generation, element construction, and executing the other operations used in the XQuery. The profile in Figure 6.3 (a summary of Table 6.2) shows were the actual time in Query 2 is spent. The Burkowski Step slowly reaches the 32% of the total time, when document sizes increase. This number seems to stabalize (notice the almost logarithmic scale on the X-axis). This means that only limited improvement (maximally 32%) can be made by optimizing the Burkowski Step itself. The figure does also indicate that the Burkowski Step is considerably slower than the child-step; if for example in the 1100MB document 31.8% of the time is spend calculating the Burkowski Step (31.8% of 1.991s = 0.633s) then 68.2% (1.358s) is spend elsewhere, and the time spend in the child-axes of the orginal XMark Q2 is (total time minus time spend elsewhere is 1.672s - 1.358s =) 0.314s. This indicates that the Burkowski Step is at the moment about twice as slow as the child-step.

Profiling the Burkowski Step

The result of the previous section calls for a further investigation into the costs of the Burkowski Step. To get a better grip on the Burkowski Step, the step has been profiled internally. It can be divided into 5 core operations (as already mentioned in Chapter 5):

1. region fetching; retrieve all regions in the document. The result of this operation will be used for fetching regions that match the context and candidate nodes.
2. fetching source regions; join the list of all regions against the pre-numbers given as the context. This list is used as part of the input of the Burkowski Join.
3. fetching destination regions; join the list of all regions against the pre-numbers given as the candidates. This is the other part of the input for the Burkowski Join.
4. Burkowski Join; perform the Burkowski Join using the source and destination region list.

/select-narrow::	open_auctions	open_auction	bidder	increase	total
# result nodes:	1	7217000	6182000	6182000	
region fetching	0.01%	0.00%	0.01%	0.01%	0.02%
source region	0.04%	0.04%	7.30%	7.63%	15.01%
candidate region	0.05%	5.44%	28.00%	29.80%	63.30%
burkowski join	0.00%	1.53%	10.32%	2.60%	14.45%
result preparation	0.00%	1.27%	4.59%	1.36%	7.22%
total time	0.10%	8.29%	50.22%	41.39%	100.00%

Table 6.3: A profile of the Burkowski Step for Q2 in the 1100MB modified XMark document (% of total time spent in Burkowski Steps)

5. result preparation; match the result of the Burkowski Join (given as region-ids) against the candidate list to get the `pre`-numbers of the result.

In Table 6.3 a profile of the Burkowski Step is given for the modified XMark Query 2. In the table the performance behaviour is presented. Remarkable is that the Burkowski Join, the fourth operation, takes up only 6% of the total cost of the step. Furthermore it is strange that the candidate region fetching costs more than the source region fetching even though the source region has a similar amount of regions to select. The almost zero cost of the source region fetching seems to indicate that the region-fetching is a virtual operation; internally no data is touched. These virtual operations are internal optimizations of MonetDB.

Conclusion

Although the graphs show that the current implementation works quite well, there are still a few improvements possible/needed. The steps probably have not reached their full potential yet; for example, the current implementation does not use skipping or pruning as proposed in [16]. Using these techniques might speed up the steps even further. Only four of the 20 XMark queries could be rewritten to the modified XMark documents. Future work should study the behaviour in the other 16 queries.

6.2 Framework

The feature extraction framework was designed to be able to extract all features from a piece of case data without human intervention and without crashing, while applying the forensic tools exhaustively on the data. In this section the feature extraction framework is tested, and the questions whether the system is robust are asked:

- is it able to cope with a large amount of data?
- are the malfunctioning tools obstructing the system?
- are tools applied to the correct objects?
- what is the compression factor of the index of XIRAF (the ratio of XML versus BLOB)?
- Are there any design flaws in the feature extraction framework of XIRAF?

Answering these questions should give an indication of the robustness of the XIRAF feature extraction framework.

Test Setup

To test the framework some test sets are necessary. Due to internal restrictions at the NFI actual case data was not at hand. Therefore test data from the author's laptop and desktop have been used.

set 1 a small test-set created of tests used for individual tools,

	set 1	set 2	set 3	set 4
operating system:	mixed	Linux	Linux	Windows
size (on disk):	.014 GB	1.577 GB	7.739 GB	28.485 GB
files:	56	35500	73074	266961
folder:	8	3556	5191	25234

Table 6.4: Test-set details

	set 1	set 2	set 3	set 4
elapsed time (s):	0m33	35m12	94m52	600m
size of XML annotations (MB):	0.208	28.772	65.125	225.575
size of BLOB (x 1000 MB):	0.014	1.720		33.743
# elements:	3672	455503	990220	3482974
# extracted regions:	507	83198	229897	472466
# different mime types:	13	62	71	96
nestedness of features:	4	3	3	4
XML depth:	12	18	20	25

Table 6.5: some results from the extraction process

set 2 a home-dir from a Linux (Fedora Core 4) environment except the hidden files,

set 3 a home-dir from a Linux (Fedora Core 4) environment including the hidden files, and

set 4 a copy from a Windows XP environment on a 30GB hard-disk.

A summary of the data-sets is depicted in Table 6.4. The sizes of set 3 and set 4 are beyond the 32-bit limit, which from a software engineering perspective is a critical point. It is assumed that if the framework behaves correctly on these test-sets it will remain behaving correctly for larger amounts of data.

During the XIRAF project a number of tools has been successfully wrapped:

- Skype-log analyser (tool by NFI)
- Internet History analyser (tool by NFI)
- Email-examiner (a Windows application)
- email-address extractor (tool created from scratch)
- generic tools (zip/pdf/jpeg/gzip/tar)

These tools (except for the Email-examiner) have been applied to the data.

Test Results

Table 6.5 depicts the results of unleashing the feature extraction framework on the data sets. 'Nestedness of features' means the number of cycles the feature extraction framework had to make through all the tools in order to find all annotations. The XML-depth depicts the number of steps towards the deepest XML element. To summarize some results of the testing:

- Many tools do not generate region information.
- The IE-history-tool, and emailexaminer-tool could only annotate objects in the Windows test set, but did not obstruct the processing of other test sets even though they were applied to the data.
- One of the forensic analysis tools (NetAnalysis) could not be wrapped. It did not have a command-line interface and neither did it have another API.

- A problem already identified in Chapter 4 is that tools that annotate the same type of objects should produce annotations that are structured in the same way. An example: a quick implementation of the unzip-tool stored the relative folder-name together with the file-name of a file in the name-attribute of the file-element. This obfuscated the results of the case: Input Descriptors depending on the start of the name of a file did not find the desired results. Another short-coming of the unzip-tool was the lack of being able to identify the mime-type of the extracted files. These problems illustrate the need for consistent naming and careful specification of DTD's.
- Wrapping some of the tools appeared to be difficult, because some tools required the naming (especially the extensions) of the files to be correct, because they checked the file-type based on these names. None of the tools in this category can be called robust as they are easy to fool by renaming files.
- One of the problems experienced during the preparation of the tests, was that tools usually produce textual output in ASCII encoding. XML uses Unicode encoding for text. The conversion from ASCII to Unicode needs to be handled properly, but in the current implementation an adhoc solution has been implemented, which ignores special ASCII characters, and only converts the ASCII characters that are equal to Unicode characters.

Conclusion

Taken as a whole wrapping of tools seemed a little trickier than expected. On the other hand, the application of tools to the case data went better than expected. The system was already designed to allow for using tools on the projects that would not find any objects. It turned out that this mechanism worked quite well.

The big question is however whether XIRAF is scalable. Knowing that MonetDB/XQuery can handle documents up to 11GB, the current annotations seem no problem to the database, not even if hard disks with 100's of gigabytes of data are used. However, if more tools will be used, more annotations will be produced by XIRAF. The annotations can in turn be limited again, by for example eliminating known-to-be-good-files from the process. The second problem is that the feature extraction process takes to much time. The overhead introduced by the feature extraction framework should be reduced, otherwise XIRAF will not scale to fullsize investigations.

6.3 Expressiveness

Test Setup

For testing expressiveness of XIRAF, various queries have been formulated over the four test sets mentioned in the previous section. The goal of the tests was to see to which extend XIRAF would be capable of answering queries asked in forensic investigations. One of the problems with measuring this ability is that XIRAF was meant to be able to answer queries in a broad range of forensic investigation types. Another problem was that XIRAF is heavily dependant on the feature extraction capabilities of the wrapped forensic analysis tools.

To be able to measure the expressiveness, XIRAF has been equiped with a small set of tools, and a set of queries has been created. Given the queries listed in Chapter 2, the following XQuery translations have been used:

1. Is there a cookie for site 'google' present?

```
let $d := doc("result.xml")
return $d//folder[@name[ends-with(.,"Cookies")]
                ]//file[@name[contains(.,"google")]
                       ]/@name
```

2. Is there a logitem for 'sourceforge.net' present?

```
let $d := doc("result.xml")
return $d//logitem[url[contains(.,".sourceforge.net/")]]
```

3. What is the latest visit to site 'google.com'?

```
let $d := doc("result.xml")
return max($d//logitem[url[starts-with(.,"http://www.google.com")]
                       ]/mtime/@unixtime)
```

4. What events took place on date 'November 9, 2004'[2]

```
let $d := doc("result.xml")
let $x := 1099954800  (: "Nov 9, 2004 0:00:00" :)
return $d//date[(@unixtime > $x)
            and (@unixtime < $x + 86400)
               ]/ancestor::*/@name
```

5. Which instant messenger applications (Skype, Yahoo, MSN) are installed?

```
let $d := doc("result.xml")
for $i in $d//file[
                  @name[. = "Skype.exe"
                     or . = "msmsgs.exe"
                     or . = "yahoo.exe"]
              and mime[. = "application/x-dosexec"]
          ]
return ($i/ancestor::*/@name, $i/@name)
```

6. Does any of the files in list z appear on the disk? (assume z comprises 'NTUSER.DAT' and 'aap.xml')

```
let $d := doc("result.xml")
let $z := ("NTUSER.DAT", "aap.xml")
for $i in $z
return
  $d//file[@name[. = $i]]/@name
```

An observation made while trying to translate queries, is the need for external sources. Specifying for example a query to search for chat-programs actually requires a list of programs which are known to be chat-programs together with a list of executable files, with their MD5 checksum that should be present.

Test Results

The dependency of the tools is immediately clear: there is currently no tool wrapped that analyses known-files, or MS-Word files, and these can therefore not be queried. Another problem is the lack of a uniform XML-schema for queries/tools. If a DTD for a tool has not yet been determined, it is hard to formulate such a query. The given queries in the above list therefore only work if the tool actually writes its output as required by XIRAF. For example, integrating two tools that annotate similar objects, only makes sense if both annotations can be queried as a whole. Therefore tools, annotating similar objects, should use similar DTDs for these objects.

In Table 6.6 query times are depicted for the 28GB test-set of which the index totalled 225MB of XML. The third column shows the number of results for the query. For query 3 there was no result at all. This is caused by the malformed query. The names of the urls in the history-log do not match the names of the files in the Temporary Internet Files folder. Internet Explorer modifies the names of the files. A better query has not yet been found. Besides query 3, each query returned a limited set of results; at most 31. The queries can be said to be quite specific, and an investigator would be able to read all results.

[2]time/date functions are not (yet) supported supported in MonetDB; therefore the time has been measured in seconds since '1-Jan-1970'. One day equals 86400 seconds.

	time (s)	results (#)
query 1:	2.142	3
query 2:	2.034	31
query 3:	2.031	0
query 4:	4.684	27
query 5:	4.605	6
query 6:	18.946	10

Table 6.6: Execution time results of forensic queries on the 28GB test set

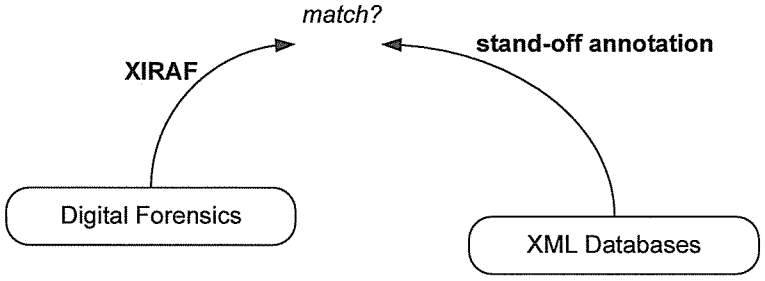
Conclusion

In the queries it is difficult to abstract from the knowledge of the individual tools (the number of tools in the tool repository is too small to forget about the existence of the tools). A better experiment should be set up in which the user of the system does not have direct knowledge of the tools used by XIRAF.

Chapter 7

Conclusion

In this thesis an effort has been made to join the field of digital forensics and databases. On the one hand common problems in the field of digital forensics have been identified. On the other hand the field of database management have been analysed and short-comings in XML annotation on external data have been identified. XIRAF was created to bring the field of digital forensics closer to the database field. Stand-off annotation has been proposed to make XML-databases more suitable for annotating external data.



7.1 Research Questions

The XIRAF project was intended to explicitly identify the needs of the NFI (or more general, the field of digital forensics) and find corresponding solutions already present at the CWI. The questions have been answered by combining MonetDB/XQuery from the CWI and forensic analysis tools from the NFI. Here the research questions from the introduction are restated:

1 (a) Can output from forensic analysis tools be converted to XML and (b) does, the quality of the tool's output degrade due to this conversion?

2 (a) How should stand-off annotation be shaped, and (b) can stand-off annotation be queried using widely accepted XML query languages such as XQuery? If not, (c) which extensions are necessary, and (d) how can these extension be implemented efficiently?

3 Is it possible to formulate forensic questions using a stand-off annotation query language?

4 Is a unified approach for integrating forensic analysis tools feasible?

As the answer to question (1a): most of output of tools can be represented in XML, forensic analysis tools either have their own XML-representation or the tools can be easily wrapped (as shown in Chapter 5) to produce XML. From the tools tried only the Email-Examiner-tool could not be wrapped due to the lack of a scripting interface.

Question (1b) can be answered by discussing the overhead introduced by XIRAF and the loss of information due to conversion to XML. The overhead introduced by XIRAF when extracting the features is currently high. Both in time and space. The overhead can however be reduced as has been pointed out in Chapter 4 and Chapter 5. When the extracted features are represented in text the conversion from text to XML does not degrade the annotations. The specific formatting of output by a tool can be easily reproduced by using a stylesheet or a similar formatting technique. Tools that have a text search feature, will have to depend on the search-capabilities of MonetDB/XQuery.

The XML language can annotate case data, but only allows for a single hierarchy. However, languages like GODDAG or LMNL are not needed to express the structures found in digital forensics, because the notion of stand-off XML makes it possible to query over multiple hierarchies. XQuery can be used to query these hierarchies, but when querying over *multiple* hierarchies XQuery needs to be extended to use 'region' information. And this would be the answer to research question 2. The approach used in this thesis is the implementation of the Burkowski steps. These steps have been implemented in the MonetDB/XQuery system and in Chapter 6 it has been shown that the steps are efficiently implemented. Other stand-off-related extension, such as access to the data in the BLOB from within XQuery and the stand-off update functions still need to be implemented properly. The Burkowski steps are only a minor addition to the XQuery/XPath language, and do not involve a major change in the semantics of XQuery.

A database approach allows forensic investigators to express a much broader range of queries. The query-power introduced by converting the output of the tools to XML goes beyond the current level of querying at the NFI. A new type of queries has to be invented/formulated by working with XIRAF. XIRAF lacks a text search, but with a proper Information Retrieval system linked to MonetDB/XQuery the question formulation could be unlimited. It is noted that performance decreases when queries become more complex. It remains to be seen whether the questy actually can be formulated by forensic investigators, and the author assumes that the use of XQuery patterns will make query formulation less powerful but much better understandable. This should be the answer to research question 3.

When DBMS's gain IR capabilities, than IR-related forensic tools will be obsolete or should be incorporated into the DBMS. Currently the user interfaces of other 'digital wash tunnels' are more powerful by providing only those functions that have been proven to work in earlier investigations. XIRAF will have to mature, and suitable tools should be added to the tool repository. Besides this changes, the underlying database, in this case MonetDB/XQuery should adopt some IR functionality, otherwise retrieval by content would be very difficult to implement. Techniques such as suffix arrays should be looked at. To give an optimistic answer to research question 4: yes, it is be possible to create a unified approach and, although not perfect, XIRAF would be a good example of such an approach.

7.2 Recommendations/Future Work

XIRAF is still far from perfect. The aim of the XIRAF project was to touch all aspects of XIRAF. None of the aspects have received full attention or a complete solution. This was mostly due to limited time. A list is presented below with some of the unsolved issues. Hopefully they can be addressed over time. In this list the implementation specific issues of XIRAF are not mentioned.

1. There should be some research performed on how stand-off XML annotation documents can be updated properly. For this to be done, it might be wise to wait till a standard for updating standard XML has been proposed.

2. A better implementation of stand-off annotation might be useful, for example being able to shred stand-off documents into the database system and be able to access stand-off data (the BLOB) from within the query language.

3. As already mentioned a few times throughout this thesis; a region should be implemented as a set of contiguous segments and not as a single contiguous segments.

4. More testing of the XIRAF system is needed. XIRAF needs to be applied to real cases (can be old cases, but actual case data and especially actual questions are needed).

5. XIRAF, or actually the underlying database system, should be integrated with an IR system. This could be the integration of MonetDB/XQuery with TIJAH.

6. A suitable, forensic investigator friendly user interface should be created to be able to have an actual end-user use the system.

7. To allow (and encourage) forensic analysis tool manufacturers to add tools to XIRAF, a standard should be developed. This standard should specify how tools have to be wrapped.

7.2.1 MonetDB/XQuery

MonetDB has been used as the back end of XIRAF. MonetDB/XQuery, although it has become much more usable than it was at the beginning of the year, has not reached a stable state. Too many errors still have to be reported and fixed, which takes up a lot of time. Besides the immaturity the system also lacks some important features. Many queries involve time-line analysis, in which dates needs to be grouped, ordered, and selected. A serious drawback of MonetDB/XQuery is the lack of date/time-support. The most crucial negative point when querying using XIRAF is the lack of IR functionality in MonetDB/XQuery, the data in the BLOB itself cannot be queried over, it is merely the querying of the structure derived with the tools. If XIRAF would be extended with an IR-support, or better, MonetDB/XQuery would be extended with IR support, the querying could be much more effective.

7.2.2 Stand-off Annotations and IR

In the database world it has been argued for a while that databases should be more integrated with IR functionality[9][11]. INEX has been an initiative for inforation retrieval using structured queries. XIRAF will benefit from the integration of IR and databases.

Stand-off annotation might be a good starting point for integrating DB and IR. In stand-off annotation the XML-document is seen as an index on the BLOB. In this way, many other indices can be created on top of the BLOB and thereby not disturbing the index created by the DB environment. Certainly as MonetDB is capable of supporting different front-ends with a single underlying language (the MonetDB Interpreter Language (MIL)).Not only the ranking of results (such as proposed for example in XQuery-fulltext) is interesting, but also simple inverted-word-lists or other IR-related index-structures like suffix arrays could help current DB technology (for example speedup functions like `contain()` or create an `about()`).

7.3 XIRAF

XIRAF has shown that querying over multiple hierarchies is possible using an extension to the XQuery language which has been named the Burkowski Joins. XIRAF also showed that many forensic analysis tools can be 'wrapped' using a simple framework construction. The benefit of being able to query over multiple tools will hopefully make up for the overhead introduced by XIRAF. This thesis has made the definition of stand-off annotation more explicit, and has hopefully opened a new research field within the area of information retrieval.

Bibliography

[1] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, D. McBeath, M. Rys, and J. Shanmugasundaram. Xquery 1.0 and xpath 2.0 full-text working draft.

[3] N.L. Beebe and J.G. Clark. A Hierarchical, Objectives-Based Framework for the Digital Investigation Process. In *Digital Investigations Process Framework*, pages 1–17, Baltimore, Maryland, U.S., August 2004.

[4] S. Boag, D. Chamberlin, M.F. Fernndez, D. Florescu, J. Robie, and J. Siméon. Xquery specification 1.0.

[5] P.A. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*. Ph.d. thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, May 2002.

[6] P.A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. Pathfinder: XQuery-The Relational Way. In *Proceedings of the 31st VLDB Conference*, Trondheim, Norway, September 2005. (Demo). Accepted for publication.

[7] P.A. Boncz, S. Manegold, and J. Rittinger. Updating the Pre/Post Plane in MonetDB/XQuery. In *Proceedings of the International Workshop on XQuery Implementation, Experience and Perspectives (XIME-P)*, Baltimore, MD, USA, June 2005.

[8] A.P.A. Broeders. *Op zoek naar de bron.* Proefschrift, Universiteit van Leiden, Rijswijk, The Netherlands, 2003.

[9] F.J. Burkowski. Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Text. In *Proceedings of the 1992 SIGIR Conference*, pages 112–125, 1992.

[10] B. Carrier. The sleuthkit informer. Electronic Journal.

[11] S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating DB and IR Technologies: What is the Sound of One Hand Clapping? In *Proceedings of the 2005 CIDR Conference*, 2005.

[12] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, November 1999.

[13] A. Dekhtyar and I.E. Iacob. A Framework for Management of Concurrent XML Markup. *Conceptual Modeling for Novel Application Domains*, 2814(2003):311 – 322, Sep 2003.

[14] S. DeRose. Markup Overlap: A Review and a Horse. In *Extreme Markup Languages 2004*, 2004.

[15] S. DeRose, E. Maler, and D. Orchard. XML Linking Language (XLink) Version 1.0. W3C Recommendation, June 2001.

[16] T. Grust, M. van Keulen, and J. Teubner. Staircase Join: Teach a Relational DBMS to Watch its (Axis) Steps. In *Proceedings of the 29th Conference on Very Large Databases (VLDB)*, Berlin, Germany, September 2003.

[17] I.E. Iacob and A. Dekhtyar. Towards a Query Language for Multihierarchical XML: Revisiting XPath. In *Proceedings of the 8th International Workshop on the Web and Databases (WebDB 2005)*, Baltimore, Maryland, USA, 2005.

[18] I.E. Iacob, A. Dekhtyar, and W. Zhao. XPath Extension for Querying Concurrent XML Markup. Technical report, University of Kentucky, February 2004.

[19] M. Kay. Comparing XSLT and XQuery. In *XTech 2005: XML, the Web and beyond*, 2005.

[20] H. Liefke and D. Suciu. XMill: an efficient compressor for XML data. In *Proceedings of SIGMOD*, pages 153–164, Dallas, Texas, U.S.A., 2000.

[21] J. List, V. Mihajlović, G. Ramírez, A.P. de Vries, D. Hiemstra, and H.E. Blok. TIJAH: Embracing IR Methods in XML Databases. *Information Retrieval*, 8, 2005. To appear.

[22] M. Hilbert and O. Schonefeld and A. Witt. Making CONCUR work. In *Extreme Markup Languages 2005*, Montreal, 2005.

[23] G. Navarro and R. Baeza-Yates. Proximal Nodes: A Model to Query Document Databases by Content and Structure. In *ACM Transactions on Information Systems*, volume 15, pages 400–435, 1997.

[24] M.G. Noblett, M.M. Pollitt, and L.A. Presley. Recovering and Examining Computer Forensic Evidence. *Forensic Science Communications*, 2(4), October 2000.

[25] P. Ogilvie. Retrieval Using Structure for Question Answering. In *The First Twente Data Management Workshop (TDM'04)*, pages 15–23, 2004.

[26] P. Durusau and M.B. O'Donnell. Coming down from the trees: Next step in the evolution of markup? In *Extreme Markup Languages 2002*, Montreal, 2002.

[27] P. Durusau and M.B. O'Donnell. Concurrent Markup for XML Documents. In *Extreme Markup Languages 2003*, Montreal, 2003.

[28] G. Palmer, editor. *A road map for digital forensics research*, 2001.

[29] E. Pianta and L. Bentivogli. Annotating Discontinuous Structures in XML: the Multiword Case. In *Proceedings of LREC 2004 Workshop on "XML-based richly annotated corpora"*, pages 30–37, Lisbon, Portugal, 2004.

[30] W. Piez. Half-steps toward LMNL. In *Proceedings of the fifth Conference on Extreme Markup Languages*, 2004.

[31] G. Ramìrez and A.P. de Vries. Combining indexing schemes to accelerate querying XML on content and structure. In *The First Twente Data Management Workshop (TDM'04)*, pages 44–51, 2004.

[32] G. Ruibin, Chan Kai Yun, and M. Gaertner. Case-Relevance Information Investigation: Binding Computer Intelligence to the Current Computer Forensic Framework. *Int. Journal of Digital Evidence*, 4(1):1–13, 2005.

[33] E. Schop. Databerg nekt digitale recherche. *Automatisering Gids*, 2005.

[34] Alan F. Smeaton, Paul Over, and Wessel Kraaij. Trecvid: evaluating the effectiveness of information retrieval tasks on digital video. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 652–655, New York, NY, USA, 2004. ACM Press.

[35] C. M. Sperberg-Mcqueen, T. Consortium, and L. Burnard. *Guidelines for Electronic Text Encoding and Interchange: Volumes 1 and 2: P4*. University Press of Virginia, 2003.

[36] C.M. Sperberg-McQueen and C. Huitfeldt. GODDAG: A Data Structure for Overlapping Hierarchies. *Lecture Notes in Computer Science*, 2023:139 – 160, January 2004.

[17] I.E. Iacob and A. Dekhtyar. Towards a Query Language for Multihierarchical XML: Revisiting XPath. In *Proceedings of the 8th International Workshop on the Web and Databases (WebDB 2005)*, Baltimore, Maryland, USA, 2005.

[18] I.E. Iacob, A. Dekhtyar, and W. Zhao. XPath Extension for Querying Concurrent XML Markup. Technical report, University of Kentucky, February 2004.

[19] M. Kay. Comparing XSLT and XQuery. In *XTech 2005: XML, the Web and beyond*, 2005.

[20] H. Liefke and D. Suciu. XMill: an efficient compressor for XML data. In *Proceedings of SIGMOD*, pages 153–164, Dallas, Texas, U.S.A., 2000.

[21] J. List, V. Mihajlović, G. Ramírez, A.P. de Vries, D. Hiemstra, and H.E. Blok. TIJAH: Embracing IR Methods in XML Databases. *Information Retrieval*, 8, 2005. To appear.

[22] M. Hilbert and O. Schonefeld and A. Witt. Making CONCUR work. In *Extreme Markup Languages 2005*, Montreal, 2005.

[23] G. Navarro and R. Baeza-Yates. Proximal Nodes: A Model to Query Document Databases by Content and Structure. In *ACM Transactions on Information Systems*, volume 15, pages 400–435, 1997.

[24] M.G. Noblett, M.M. Pollitt, and L.A. Presley. Recovering and Examining Computer Forensic Evidence. *Forensic Science Communications*, 2(4), October 2000.

[25] P. Ogilvie. Retrieval Using Structure for Question Answering. In *The First Twente Data Management Workshop (TDM'04)*, pages 15–23, 2004.

[26] P. Durusau and M.B. O'Donnell. Coming down from the trees: Next step in the evolution of markup? In *Extreme Markup Languages 2002*, Montreal, 2002.

[27] P. Durusau and M.B. O'Donnell. Concurrent Markup for XML Documents. In *Extreme Markup Languages 2003*, Montreal, 2003.

[28] G. Palmer, editor. *A road map for digital forensics research*, 2001.

[29] E. Pianta and L. Bentivogli. Annotating Discontinuous Structures in XML: the Multiword Case. In *Proceedings of LREC 2004 Workshop on "XML-based richly annotated corpora"*, pages 30–37, Lisbon, Portugal, 2004.

[30] W. Piez. Half-steps toward LMNL. In *Proceedings of the fifth Conference on Extreme Markup Languages*, 2004.

[31] G. Ramìrez and A.P. de Vries. Combining indexing schemes to accelerate querying XML on content and structure. In *The First Twente Data Management Workshop (TDM'04)*, pages 44–51, 2004.

[32] G. Ruibin, Chan Kai Yun, and M. Gaertner. Case-Relevance Information Investigation: Binding Computer Intelligence to the Current Computer Forensic Framework. *Int. Journal of Digital Evidence*, 4(1):1–13, 2005.

[33] E. Schop. Databerg nekt digitale recherche. *Automatisering Gids*, 2005.

[34] Alan F. Smeaton, Paul Over, and Wessel Kraaij. Trecvid: evaluating the effectiveness of information retrieval tasks on digital video. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 652–655, New York, NY, USA, 2004. ACM Press.

[35] C. M. Sperberg-Mcqueen, T. Consortium, and L. Burnard. *Guidelines for Electronic Text Encoding and Interchange: Volumes 1 and 2: P4*. University Press of Virginia, 2003.

[36] C.M. Sperberg-McQueen and C. Huitfeldt. GODDAG: A Data Structure for Overlapping Hierarchies. *Lecture Notes in Computer Science*, 2023:139 – 160, January 2004.

Bibliography

[1] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, D. McBeath, M. Rys, and J. Shanmugasundaram. Xquery 1.0 and xpath 2.0 full-text working draft.

[3] N.L. Beebe and J.G. Clark. A Hierarchical, Objectives-Based Framework for the Digital Investigation Process. In *Digital Investigations Process Framework*, pages 1–17, Baltimore, Maryland, U.S., August 2004.

[4] S. Boag, D. Chamberlin, M.F. Fernndez, D. Florescu, J. Robie, and J. Simèon. Xquery specification 1.0.

[5] P.A. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*. Ph.d. thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, May 2002.

[6] P.A. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. Pathfinder: XQuery-The Relational Way. In *Proceedings of the 31st VLDB Conference*, Trondheim, Norway, September 2005. (Demo). Accepted for publication.

[7] P.A. Boncz, S. Manegold, and J. Rittinger. Updating the Pre/Post Plane in MonetDB/XQuery. In *Proceedings of the International Workshop on XQuery Implementation, Experience and Perspectives (XIME-P)*, Baltimore, MD, USA, June 2005.

[8] A.P.A. Broeders. *Op zoek naar de bron*. Proefschrift, Universiteit van Leiden, Rijswijk, The Netherlands, 2003.

[9] F.J. Burkowski. Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Text. In *Proceedings of the 1992 SIGIR Conference*, pages 112–125, 1992.

[10] B. Carrier. The sleuthkit informer. Electronic Journal.

[11] S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating DB and IR Technologies: What is the Sound of One Hand Clapping? In *Proceedings of the 2005 CIDR Conference*, 2005.

[12] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, November 1999.

[13] A. Dekhtyar and I.E. Iacob. A Framework for Management of Concurrent XML Markup. *Conceptual Modeling for Novel Application Domains*, 2814(2003):311 – 322, Sep 2003.

[14] S. DeRose. Markup Overlap: A Review and a Horse. In *Extreme Markup Languages 2004*, 2004.

[15] S. DeRose, E. Maler, and D. Orchard. XML Linking Language (XLink) Version 1.0. W3C Recommendation, June 2001.

[16] T. Grust, M. van Keulen, and J. Teubner. Staircase Join: Teach a Relational DBMS to Watch its (Axis) Steps. In *Proceedings of the 29th Conference on Very Large Databases (VLDB)*, Berlin, Germany, September 2003.

Appendix A

Overview of Forensic Tools

Tool	Manufacturer	
The Sleuth Kit / Autopsy	Brian Carrier	Extracts/recovers file-systems from images
EnCase	Guidance Software	Is a suite with multiple tools
Forensic Toolkit (FTK)	AccessData	Is a suite with multiple tools
Rifiuti	Foundstone Inc.	Examines the Recycle Bin
Pasco	Foundstone Inc.	Examines the Internet Explorer cache
Galletta	Foundstone Inc.	Examines cookie files
NTlast	Foundstone Inc.	Examines system logs
Foremost	U.S. Air Force	Recovers files based on their headers, footers and internal data structures
Net Analysis	Paraben Forensics	Extracts log-items from several log file formats
ZyIndex	Zylab	Indexes several types of documents using XML
E-mail Examiner	Paraben Forensics	Extracts e-mail messages from e-mail archives
GenTree / GenX / GenText	Vogon International	Extracts the file-system from images and creates an inverted word list

[37] G. M. Sur, J. Hammer, and J. Simeon. An XQuery-based language for processing updates in XML. In *Proceedings of PLAN-X 2004*, 2004.

[38] H.S. Thompson and D. McKelvie. Hyperlink semantics for standoff markup of read-only documents. In *Proceedings of SGML Europe '97*, Barcelona, Spain, May 1997.

[39] A. Trotman and B. Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In *INEX 2004 Workshop Proceedings*, pages 16–40, 2004. http://www.cs.otago.ac.nz/postgrads/andrew/2004-4.pdf.

[40] M.R. Windhouwer, A.R. Schmidt, and M.L. Kersten. Acoi: A System for Indexing Multimedia Objects. In *International Workshop on Information Integration and Web-based Applications & services*, Yogyakarta, Indonesia, November 1999.

[41] F. Yergeau, T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) Version 1.1, W3C Recommendation, February 2004.

Appendix B

XIRAF Documentation

In this chapter the XIRAF system itself is documented. The documentation is intended for XIRAF users and developers. XIRAF users should be able to run investigations with XIRAF using this manual, and developers should be able to add new forensic analysis tools to the tool repository. For developers to be able to extend XIRAF, it would be wise to read at least chapter 4 and chapter 5 of the thesis.

B.1 XIRAF Investigation

Before an investigation (a project) can start, the XIRAF project first needs to be initialized. Each XIRAF Project needs a few parameters:

- blb-storage-folder : the place to store all the BLOB (must be an area with at least free space equal to the size of the image)
- a logical name for the XIRAF Project XML-document in the database

After a XIRAF project has been initialized, every action on the project is done through wrapped tools. The first thing to do is adding the case data to the project. Adding case data to the BLOB should also be done by a tool. Such a tool might pre-process the data, but it could also be a simple copy command, and the case data can be in any binary format, as long as it can be recognized by one of the tools in the tool repository. This is done by calling `add.sh`. The script will take the location of the case data, the project to add the data to, and the initial tool (the one that adds the data to the BLOB) as arguments.

B.2 Run a set of tools

The normal use of XIRAF, after a project has been initialized and the case data has been added, is to run a (selected) set of tools on the data. This is done by calling the `complete.sh` script. The script will exhaustively apply the tools listed in `/src/tools/toollist.txt` to the case data.

B.3 Run a single tool

In XIRAF it is possible to run a single tool on the case data. This can either be done before, after or instead of `complete.sh`. Use `invoke.sh` to do this. `invoke.sh` uses the current directory for temporary storage. Make sure that the current directory has a considerable amount of free space, because XIRAF might need more than the size of the case data for temporary storage (although it usually does not). `invoke.sh` requires the following command-line arguments:

1. path to the tool wrapper (example: `$XIRAFPATH/tools/skypetool/`)
2. XIRAF project info (both the name of the project and the location of the blob-area)

Calling `invoke.sh` without any arguments will provide give a usage description.

B.4 Creating a Wrapper

In this section the process of wrapping a forensic tool is explained. A forensic analysis tool must meet the following requirements for being wrapped and added to the tool repository: A tool-wrapper consists of the following two items:

input descriptor To wrap a tool, first its input needs to be specified with an 'Input Descriptor'. A tool must specify its input as an XQuery.

tool-executable-wrapper The actual communication between XIRAF and the tool happens through the tool-executable wrapper. XIRAF invokes the tool-executable-wrapper, and in turn the tool-executable-wrapper invokes the forensic analysis tool in the correct way. A tool-execution-wrapper must accept a stream or file as input data (this may either be an image of a file-system, a physical file (for example: pdf-document or e-mail-archive), or an arbitrary piece of data represented as a file)

Input Descriptor

The input descriptor is an XQuery, or actually part of a bigger XQuery. The input descriptor is used to decide which parts of the blob need to be fed to the tool. When a tool is invoked, XIRAF will prepare the data by exporting the data specified by the descriptor input of the tool.

```
//file[extension='.pdf']
```

The tool should be able to cope with non-expected data and preferably not run forever if given. XIRAF should implement a timeout for each tool.

The input descriptor should return XML-nodes from the stand-off XML document. In principle it can return any type of node, but XIRAF is currently accepting only region-elements (elements having an `start`- and `end`-attribute and will ignore all other results.

An input descriptor should find a balance between selectivity, and specificity. For example: if looking for Microsoft Outlook's personal folder files (usually identified by a `.pst`-extension, it could find all `.pst`-files or it could find only `.pst` files in folder `/Application Data`. It might then miss the users backups somewhere else.

Tool-executable-wrapper

The tool-executable-wrapper of a tool will be called by the framework for each input object. After the tool finishes it should have produced some XML describing the input object.

A tool-executable itself should accept a name of a file as input (the file will contain the data and will be readable for the tool). The tool-executable-wrapper should accept two names of files for the output (one for the XML output and one for the BLOB-output). The files will not yet exist, but the tool will get sufficient rights to create the files. The tool should create at least the XML-file (the BLOB-file will be neglected otherwise).

The resulting file with XML should be valid XML except for that it does not have to be a valid XML-*document* with a single root-node (there may be multiple XML-fragments in the result). It should not contain processing-instructions nor should it contain comments. The tool should not use namespaces. To validate the XML data, `xmlwf` is used. Invalid output will not be shredded into the project.

In the current implementation containers are used to group annotations of a tool. These container-elements are automatically added to the output of the tool. They contain information about the tool.

B.5 List of wrapped tools

B.5.1 Skype Tool

tool-name skype-call-tool v0.1

author Christina Dijkshoorn, Wouter Alink

date June 24, 2005

description Analyses Skype-log-files and extracts the phone-call information from it

DTD The DTD is depicted in Figure B.1.

input-descriptor `//file[@name[starts-with[.,"call"]] and ends-with[.,".dbb"]]`

new-blob-data no

known-issues does not check whether it produces correct XML

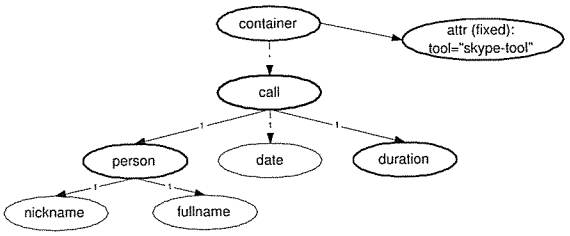


Figure B.1: Skype Tool

B.5.2 File-System Tool

tool-name filesystem-tool v0.1

author Wouter Alink

date June 1, 2005

description Analyses a file-folder structure and generates an XML fragment and a BLOB from it.

DTD The DTD is depicted in Figure B.2.

input-descriptor `//root`

params path: the full path to the root of the folder-structure to analyse

new-blob-data yes

known-issues does not check whether it produces correct XML and it does not specify the regions of the folders yet.

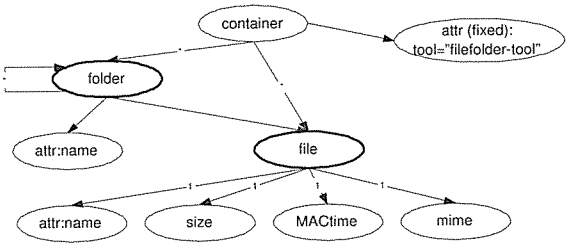


Figure B.2: File-System Tool

B.5.3 PDF Tool

tool-name pdf-tool v0.1

author Wouter Alink

date June 1, 2005

description Analyses pdf-files and extracts the text from it (produces a new blob)

DTD The DTD is depicted in Figure B.3.

input-descriptor `//*[mime="application/pdf"]`

new-blob-data yes

known-issues does not extract any metadata yet.

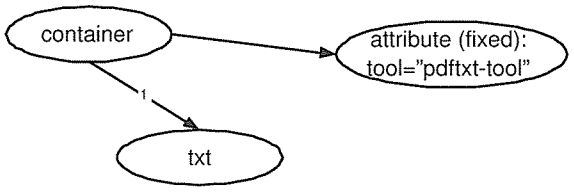


Figure B.3: PDF Tool

B.5.4 Image Tool

tool-name image-tool v0.1

author Wouter Alink

date June 1, 2005

description Analyses image-files and extracts metadata from it (using ImageMagick's `identify`)

DTD The DTD is depicted in Figure B.4.

input-descriptor `//*[mime[contains("image")]]`

new-blob-data no

known-issues does not check whether it produces correct XML, and does not yet retrieve all metadata.

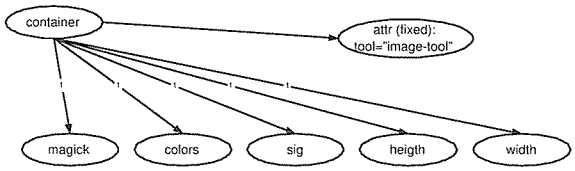


Figure B.4: Image Tool

B.6 XIRAF internal XML Structure

XIRAF has two reserved element names for its XML documents: <XIRAF/> and <container/>. XIRAF has seven reserved attributes names: xstart, xend, rstart, rend, xup, xid, and blob. Except for rstart and rend these element- and attribute-names should not be used by the tools.

B.6.1 Containers

Containers are the primary elements in a XIRAF stand-off XML document, and will always be automatically generated. They can be identified by their element-name: container; They can contain a few attributes:

- region (OPTIONAL) : the corresponding new area in the BLOB-file
- up (OPTIONAL) : a reference to the parent region
- blob (OPTIONAL) : the root-element of the XIRAF-XML file specifies the file that is used for the BLOB.

If the xup-attribute is given, the information in the container talks about the region specified by this attribute. It refers to the xid-attribute of a region. If the 'xup'- and the 'blob'-attribute are both given, this means that the data in the region specified by 'xup' should be replaced by 'blob'.

note: a containers 'up'-attribute may only point to regions that are specified in earlier containers (to avoid cycles).

B.6.2 Region

A region can be any node inside a container. It can be identified by having a xstart, xend and xid attribute.

- rstart (REQUIRED) (natural number) :
- rend (REQUIRED) (natural number) :
- xid (REQUIRED) :

The rstart- and rend-attributes denote the relative offset (measured in bytes from the beginning of the container) to where the data can be found in the BLOB-file. The value of the rstart-attribute should always be less or equal to the rend-attribute. When a tool annotates a region, it needs to specify this using rstart and rend, the relative start and end positions. XIRAF will automatically convert these values and add an xid-attribute.

B.7 Testset Generator

The xml2XIRAF-tool has been created to be able to test the XIRAF location steps. Its input is an XML document and it will convert it to an XIRAF document set (a BLOB file split into many XIRAF containers)

```
xml2XIRAF input output-blob output-XML 'non-region ' 'container ' 'includes '
```

input the input-XML document. The document should not already be a XIRAF document. It should not contain any <XIRAF> or <container> nodes, nor should it contain the attributes xstart, xend, and xid.

output-blob the document in which the data is put.

non-region-nodes Some data in the XML document can be marked as being annotation data. This data will not be included in the BLOB, but in the layers (as meta-data). This feature can be used by naming all the nodes which contain meta-data.

container-nodes Some nodes should be the starting point for new layers. these nodes will appear in the current layer as nodes, and will start a new container in the next layer.

The testset generator will discard all `comments` and `processing instructions`. The drawback of the software is that it does not generate overlap. This is a direct result of the source being a valid XML-document. A solution to this problem is that maybe the start/end-positions could be randomized a little so that some overlap will be created.

Appendix C

XIRAF User Interface

C.1 List of generic XQuery patterns

Below a list of XQuery patterns is presented. This list can be used to create XQuery's without (much) knowledge about the query language. With the element names (1), (2), (3), etc. names of objects are expected (e-mails, names, timestamps, files, folders, etc). A list of all possible options for such variables can be obtained by querying:

```
distinct-values(//*/name())
```

or for the attributes (1'),(2'), (3'), etc:

```
distinct-values(//@*/name())
```

With (a), (b), (c), etc. a resultset is denoted (the nodes are a result of a previous query).

1 query: give all (1)'s:

```
//*[name()=(1)]
```

2 query: give all (1)'s in (2)'s:

```
//*[name()=(2)]//*[name()=(1)]
```

3 query: give all (1)'s in which the word 'y' appears:

```
//*[name()=(1) and contains(., "y")]
```

4 query: give only those (a)'s in which the word 'y' appears:

```
$(a)[contains(., "y")]
```

5 query: give only those (a)'s in which (1') is 'y':

```
$(a)[@(1') = "y"]
```

6 query: give only those (a)'s which are inside (b)'s:

```
$(b)//* intersect $(a)[
```

7 query: display from result (a) the fields (1), (2) and (3), etc:

```
for $i in $(a)
return element { "result" } {
  $i/(1),
  $i/(2),
  $i/(3)
}
```

8 query: display top-k of (a) ordered by (1):

```
for $t in 1 to k
let $a := for $i in $(a)
          order by $i/(1)
          return $i
return $a[$t]
```

9 query: give the number of results in (a):

```
return count($a)
```

An example using these patterns would be to ask for "all e-mails in the 'My Documents'-folder in which the word 'suspect' appears, and show only the subject and sender." Depicted below is a possible query-plan using the query-patterns described above.

```
(1) let $a := //*[name() = "folder"]
(5) let $b := $a[@name = "My Documents"]
(1) let $c := //*[name() = "e-mail"]
(6) let $d := $a//* intersect $c
(4) let $e := $d[contains(., "suspect")]
(7) for $i in $e
    return element { "result" } {
      $i/subject,
      $i/sender
    }
```

The query-plan generated by using the patterns is more verbose. The query could be rewritten (manually optimized), but this is mostly done internally by the XQuery compiler:

```
for $i in //folder[@name = "My Documents"]//e-mail[contains(., "suspect")]
return element { "result" } {
  $i/subject,
  $i/sender
}
```

C.2 List of XIRAF specific XQuery patterns

The list in the previous section is standard XQuery. The XQuery extensions that have been added to XIRAF can help to formulate more specific queries.

1 query: give all (1)'s that are found inside the data of (2)'s:

```
//*[name()=(2)]/select-narrow::*[name()=(1)]
```

2 query: give all (1)'s that have data incommon with (2)'s:

```
//*[name()=(2)]/select-wide::*[name()=(1)]
```

3 query: return the case data for the regions of (a):

```
$a/so-data()
```