Available online at www.sciencedirect.com
ScienceDirect
Computer Standards & Interfaces 30 (2008) 8–19
COMPUTER STANDARDS & INTERFACES
www.elsevier.com/locate/csi

An open source forensic tool to visualize digital evidence

Emmanouil Vlastos [a], Ahmed Patel [b,c,*]

[a] Department of Information and Communication Systems Engineering, University of the Aegean, Samos, Greece
[b] Centre for Applied Research in Information Systems, School of Computing and Information Systems, Kingston University, Kingston upon Thames, Surrey, UK
[c] Department of Computer Science, Faculty of Information Science and Technology, University Kebangsaan Malaysia, Bangi, Selangor, Malaysia

Received 13 February 2007; accepted 31 March 2007
Available online 25 May 2007

Abstract

Visualizing digital evidence in an easy and constructive manner is a major problem because of the advanced techniques for hiding, wiping, encrypting and deleting digital data developed during the last few years. To tackle this problem, a system for visualizing digital data in 3-Dimensional (3D) mode has been developed. XML was used as a common language to allow fine-grained management of digital data with flexibility and ease. The extensibility of the implementation makes it particularly suitable as a research and development platform in the sector of open source computer forensics tools for the future. This article examines real-life problems that benefit from using this tool in a congenial and constructive manner to validate its key underlining concept. The design decisions that have been taken in producing the system architecture, and the features it supports are elaborated upon. To determine the effectiveness of the tool, an actual case study is presented which examines the results of the tool and why it is necessary to go for an open source model as a standard. The paper concludes with performance measurements of the tool and suggests possible extensions to make the tool even smarter.

Keywords: Computer forensics; Digital evidence; Digital investigation; Computer crime; Cybercrime; Java; Visualisation; Open source; XML

1. Introduction

This article attempts to overcome the limitations of existing digital evidence presentation methods and text or command line utilities by presenting a tool for visualizing file systems that facilitates an intuitive view of deleted files, wiped files, encrypted and transformed files with the aid of a 3D visualization technique. The tool facilitates searching for data in a specific block or sector, navigation through a range of blocks as 3D square box drawings, exportation and viewing of the content of a specific file, and exploring the file list in a way which offers a better view in presenting digital evidence in cybercrime investigations. We believe that these methods of presentation are superior compared to non-visual or other data or text presentation systems. We present the manner in which the tool was implemented and tested from detailed system analysis with screenshots to technical architectural design choices in order to give an appropriate appreciation and understanding on how the whole system has been developed. For the purpose of designing and developing our open source forensic tool to visualize digital evidence, we used open source components. We present a case study to show that the tool satisfies user and system requirements and briefly discuss the limitations of the tool and what future work is necessary to make it better.

2. Related work

There are numerous tools, such as EnCase Forensic Edition [1], The Sleuth Kit/TSK [2], The Coroner's Toolkit [3] and LTOOLS [4], available for both extraction and presentation of computer data in digital forensic cases. In our research project we took this fact into account by splitting our tool into two smaller modules, one for the *extraction* process of the data and the other for *presentation* of the data so that the system components become manageable and practicable. For example, this concept has also been shown to be the case by the Sleuth Kit system whereby it contains smaller modules, each one acting as

* Corresponding author. Centre for Applied Research in Information Systems, School of Computing and Information Systems, Kingston University, Kingston upon Thames, Surrey, UK.
E-mail address: whinchat@gmail.com (A. Patel).

0920-5489/$ - see front matter © 2007 Published by Elsevier B.V.
doi:10.1016/j.csi.2007.03.002

a separate entity that outputs specific data from a hard disk drive. For this approach to work effectively, a common data exchange interface is needed between the modules of the tool to communicate in a common language that overcomes any specifics related to distinct syntax and semantics of the data set.

Features related to graphical and visual effects can be a very subjective issue in the design of a system. Our project did not have enough resources and time to investigate in any detail as to what is available beyond the field of digital evidence presentation and visualization. However, one particular information presentation and visualization system of interest was found to be Starlight [5]. It can be regarded as a forerunner in the field that utilizes advanced information modeling and data management functionality with a powerful graphical visualization-oriented user interface. Although it can handle digital data in the form of files and act as an information analysis and visualization tool, it is not specifically designed for cybercrime or digital investigation. It is a general purpose information analysis visual tool and a platform for conducting advanced visualization research. It is not an open source tool and therefore cannot be easily adapted and extended by its users in the business of digital investigation and evidence presentation. Nonetheless, the techniques used for visualizing data and information in a variety of 3D form are recognised as a benchmark against which such tools should be measured in the future.

When a tool or system is split into modules that need to fulfill a common goal, then the data exchange between modules has to take place through some common language and interface. The Computing Department of the University of Glamorgan has developed a draft Document Type Definition (DTD) defining an XML-based language that is responsible for exchange of structured data between extraction and presentation modules in a common way. The extraction module, Forensic Extraction Tool (FET), returns the data in a prescribed way defined by the DTD, but its implementation applied only to the Linux 'ext2' filesystem [6]. This extraction tool also makes use of other open source tools for extracting digital data from a hard disk drive developed in earlier days. The tool can be extended to support other filesystem types while preserving the ability to output the data in the same common way defined by the DTD.

3. Forensic tools for XML (FTXml)

In some cases re-construction of digital evidence can be a very complex task. This is more so when data have to be obtained from a variety of sources from which digital evidence can be produced. The technologies and methods for wiping, deleting, hiding and transforming digital data have improved over the last few years, which makes it necessary that the process of finding those files in a hard disk drive be also improved. In building our system, called FTXml, to provide an improved solution to the problem, we used other open source tools to facilitate rapid development in recovering, extracting and presenting such digital data.

The major advantage of our visualization tool called Java Forensic Analysis Viewer (JFAV) is that it has no need to deal with the development of methods to retrieve deleted files, wiped files, encrypted files or transformed files because it has the capabilities to read any kind of data found in the XML file that each extraction tool outputs. For example, there will be other extraction tools that find only wiped files from a hard disk drive. Our visualization tool (JFAV) can easily show not only the wiped data in the same manner but all other associated data contained in the XML file, which is more comprehensive in its details. The function of this tool is to analyse, interpret and understand the file generated from FET and to output the data for the user to perform analysis by looking at data found in more detail. This tool has three major screens where its category of data is visualized:

- a 3D screen which visualizes the data (only block data in this version) in a 3D virtual world;
- a tree screen which shows the tree structure of the hard disk; and
- a hex viewer of the hard disk and its blocks.

A detailed description of the functionality of these three screens is presented later in paper.

The implementation that is described here is a visualization system offering flexible and extensible capabilities. The project's base implementation extends the grammar of the proposed FTXml DTD. In the paper, several issues and concepts of related technologies are also discussed. The actual implementation described in this paper is a solution based on these issues and concepts, with the express aims and objectives of the project summarised as follows:

- The implementation and description of a visualization system partially based on the proposed DTD, but also extending it to address the identified weaknesses.
- The provision of an architecture that will be as extensible and flexible as possible for future improvements, research and addition of features as well as the integration of the implemented system into a wider presentation tool.
- Implementation of the offered services in such a way that they will be able to be used from a variety of programming languages.
- Evaluation and testing of the implementation.
- Assessing possible future extension or refinements to the implemented tool.

The rest of the paper describes the requirements, architecture, design and implementation of the system as well as evaluates the system and possible extensions to enhance our tool (JFAV).

4. User and system requirements

The most important requirements for the system and the user are as follows:

- The user shall be able to open and load the XML file created by FET tool, to search for data (file name, block number or a hash value) in the partition analysed, to export data both for files and blocks, and to print out the investigation data. In addition the user shall be able to navigate in a 3D world between blocks in partition displayed as square boxes and choose any block number to view the data that box contains.

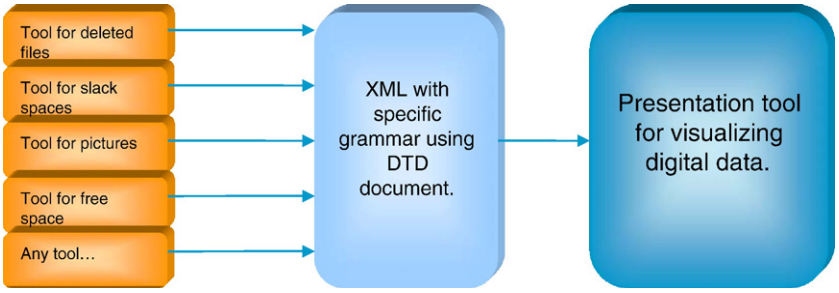

Fig. 1. High level architectural design.

Finally, the user shall be able to see information such as partition details (name, size, filesystem type), block data, categorized file structure (files, directories, deleted entries, block devices, character devices, FIFO's and sockets) and hard disk details (name, size and number of partitions).

- The system shall read and load the data for further analysis, provide appropriate viewings for the user to see the data in hex, text, tree and 3D format, and provide a search function for files, blocks and hashes. The system shall also provide a printing facility to output the results of the analysis. Finally, it is planned for the system to provide an advanced 3D navigation facility to analyse data in real time.

5. Technical approach and design choices

The following list summarises the major choices made and decisions taken during the design phase:

- *Choice of using Java as the main programming language*: Platform independence is an attractive feature for a forensic tool, allowing it to be loaded and executed on a variety of platforms running under almost every modern operating system. The Java Programming Language provides that feat in addition with sufficient robustness.
- *Choice of using Java3D as the programming framework to build the 3D virtual world*: The Java3D is more useful for standalone applications and furthermore it is better for professional use rather than for building a simple virtual world for the purposes of the World Wide Web.
- *Choice of using XML for data exchange between the extraction tool and the presentation tool*: One of the reasons for that choice is that XML can be highly beneficial in the proposed project for maintaining the forensic analysis data in a structured way, which can then be used for visualizing the data. XML is also convenient as a common format for data exchange between different implementations.
- *Choice of using Document Type Definition (DTD) as a standard grammar for creating and reading the XML file*: The reason of using DTD instead of XSD is that with DTD a simple grammar can be built from scratch and without specific knowledge of XML technologies.
- *Choice of Xerces2 Java Parser 2.5.0*[1]: Out of many available XML parsers for Java, the Xerces 2 implementation of the Apache XML Project is the best and the one that offers the most features compared to all other parsers. For the purposes of this project the Simple API for XML (SAX) will be used as the parser interface for reading and validating the XML document that the extraction tool generates.

6. System architecture

The main design characteristic of the implemented tool is the flexibility and the extensibility for future improvements by ensuring that the tool would be open source under the GNU GPL licensing[2] and can very easily be distributed to other programmers in order to improve it. The flexibility stems from the fact that the program uses XML for interchange of digital data from any forensic tool that makes use of XML language for storing the data while extracting them from any kind of source as depicted in Fig. 1. Any tool can use this project's implementation to visualize digital data with the only requirement to ensure that it produces a compatible XML file using the specified grammar defined in a DTD. This assists the program to work with a variety of tools that extract digital data from a storage device.

Another major characteristic is the 3D environment in the presentation tool. This environment provides the option to the specialist to navigate through a certain number of blocks in the range 1 to 200. The environment has been built using Java3D, as mentioned above, and is platform independent. Every additional package that was needed and used during the development of the program has been written in Java. The program can therefore work under a number of operating systems for which the Java Runtime Environment is available, including Microsoft Windows, GNU/Linux, MacOSX and Solaris.

7. JFAV architectural overview

JFAV consists of two major parts, a set of extra packages written in Java and the visualization tool, as shown in Fig. 2.

8. System design and implementation

FoXMLParser is the class responsible for handling the data represented in XML, i.e. reading and separating the data found in the FET-produced XML file to ensure that the visual part of

[1] For full details visit: http://xml.apache.org/xerces2-j/index.html.

[2] See http://www.gnu.org/licenses/licenses.html.

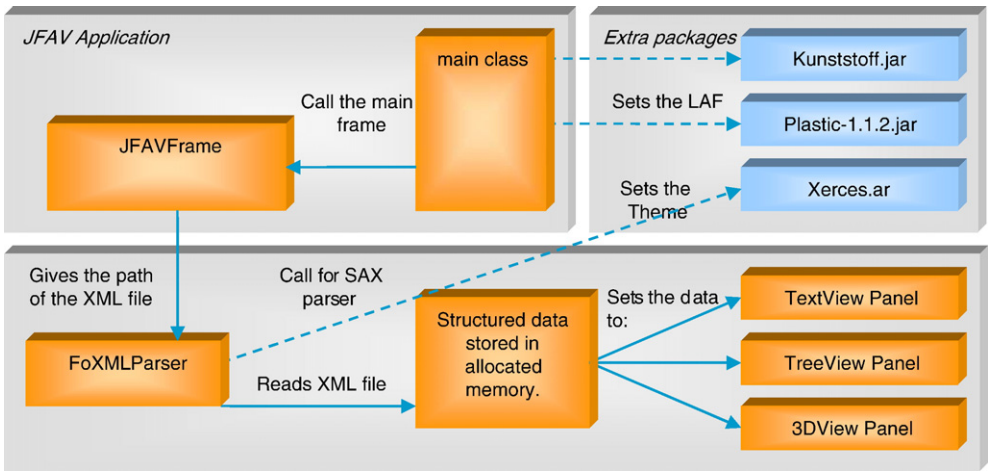

Fig. 2. Architectural overview of the program.

the tool will display the correct data for each field in the program. See Fig. 3 for a detailed technical diagram of creating and use the parser from the main program.

Once the FoXMLParser is in place and operating, the program loads all the main panels and the tool is ready for use. If the parser reads something it cannot understand, the program stops and a message appears showing the error that the Java virtual machine has caught. This provides complete error reporting. An individual explanation of each panel and its contents within the program framework is given below.

9. Presentation panels

The decision of using separate panels for showing the same data in different ways and forms was made to easily accommodate new panels capable of presenting data in more advanced ways for the visualization tool. Each panel in the JFAV program acts independently from one another, and therefore the panels can run as standalone programs as long as the parser's result data is made available to these panels. When the parser has finished reading the XML file, the main class becomes responsible for

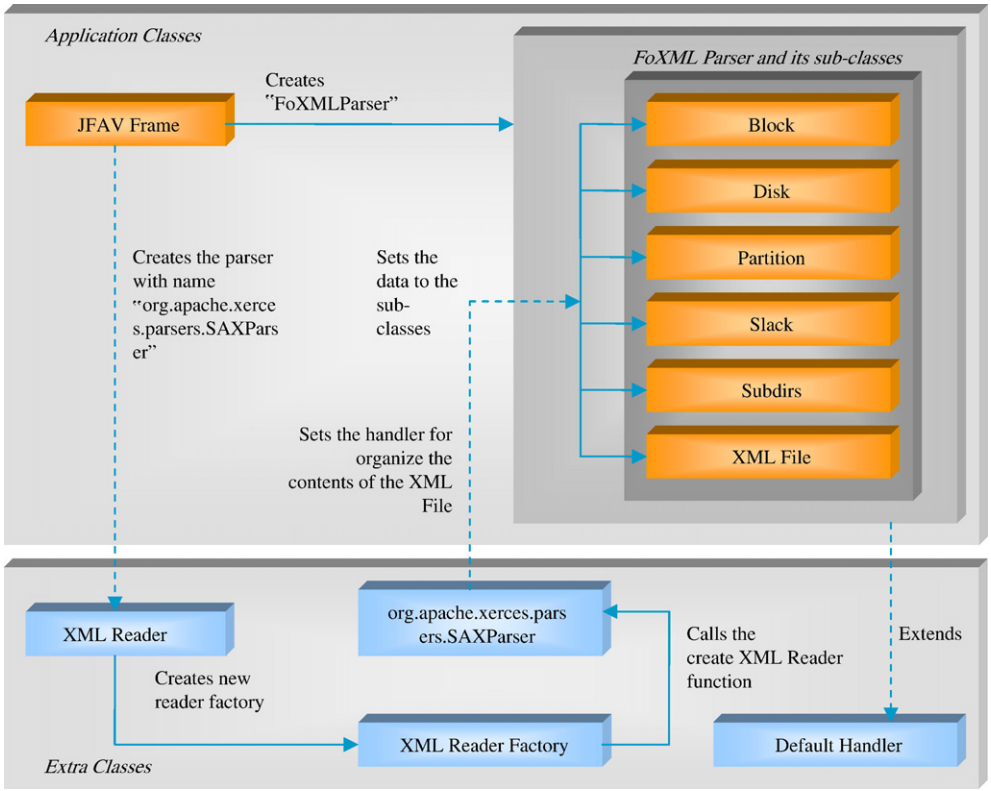

Fig. 3. Content reading and parser creation.

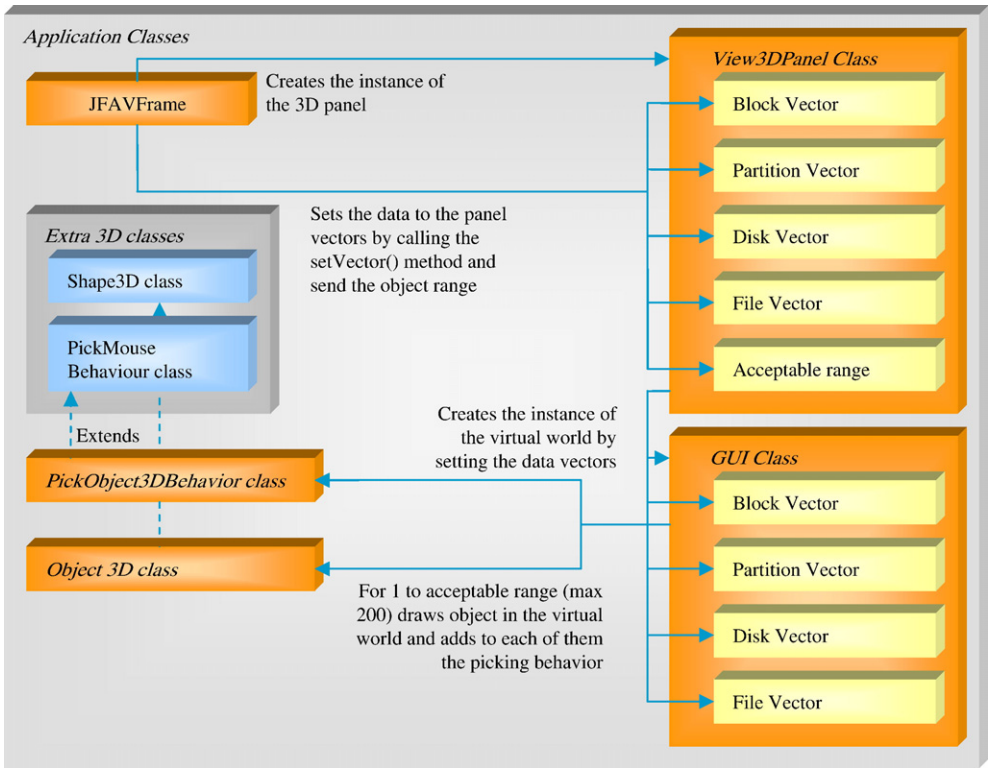

Fig. 4. 3D panel architecture.

building the panels and supporting them with the appropriate data to work correctly. All the data are kept in memory during program execution to provide faster information retrieving. In each panel class there is a public method that sets the data to the panels fields. The Parser separates each category of data read from XML file in separate vectors[3] and thus the application can retrieve those vectors quite easily.

10. 3D panel architecture

The 3D panel has some characteristics that are worth mentioning. First, the fact that the program shows the results in a 3D way makes the program quite heavy to load into the physical memory at runtime. In the first version of the program, during the development, we tried to implement the virtual world to draw all the objects found in the XML file at once on the screen. This made the program to request an extremely big amount of physical memory, which could not be provided. The final version includes the option for the user to choose a range of objects between 1 and 200. Fig. 4 depicts the overall concept of this panel.

The system allows for 3D presentation and visualisation in three different views: the block view, the explorer view, and the tree view. A flat file view is also available. All of them are shown below as examples. The objects shown in 3D block view are square boxes, allowing the user to select and open any box with the mouse. The colour of the box changes when it is selected and the detailed text in hex and text format appears in the lower part of the panel. Fig. 5 shows a screenshot of that panel. This is an initial attempt made to draw the objects as boxes. The objects can also be drawn as the representation of the file type.

As shown in Fig. 5, every box has a number above it, which is the "block number" in the partition. When a box is chosen, the colour changes to green and the associated contents are displayed in the lower part as shown by (1) on the right hand side of the picture. The blue coloured boxes are blocks that contain metadata while the red colour is used for blocks with data.

This design was chosen because this was our initial attempt to visualize this kind of data in a 3D manner. There were some difficulties regarding where to show the contents of the blocks in the 3D world. That's why a new panel, underneath the virtual world is created to show the hexadecimal and text representation of the block.

The 3D explorer view lists all the files as they appear on the hard disk drive. The circles are directories and the square boxes are single files. Dark blue circles depict system directories that are hidden, and transparent square boxes are the hidden system files. A double click on a directory opens that directory and views its contents in the same arrangement. A double click on a single file shows its data in the text section sub-window below the 3D view. A 3D explorer view showing the contents of the selected file as plain text and in hex format is illustrated in Fig. 6.

The 3D Tree view as shown in Fig. 7 lists all the files as in a common tree file explorer. The triangles on the left are the directories listed on the hard disk drive. A double click on a triangle displays a transparent level with the files contained

[3] A vector is a dynamic array used very frequently in Java.

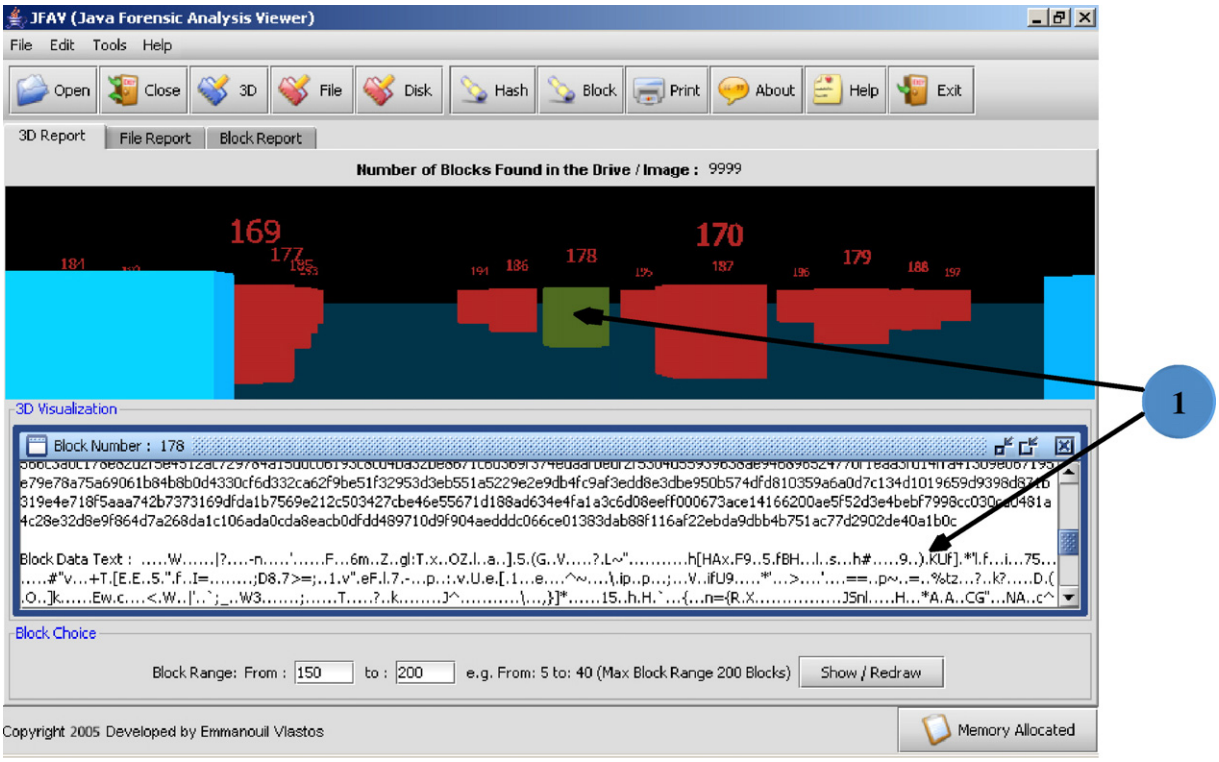

Fig. 5. 3D block view screenshot.

in the corresponding directory. The square boxes which appear as towers indicate the size of each file. Double clicking on a square box tower causes the program to execute the selected file.

11. Tree panel architecture

The tree panel shows to the user in a very attractive and intuitive way the files found in the XML data file. These files

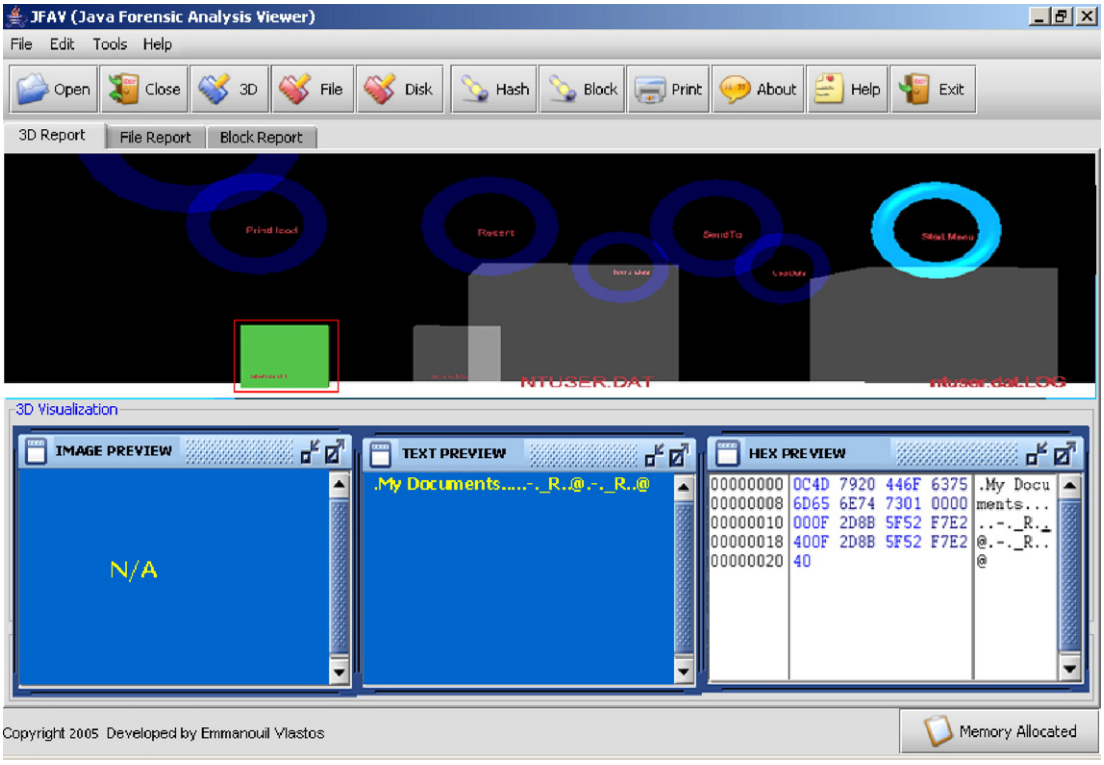

Fig. 6. 3D explorer view Screenshot.

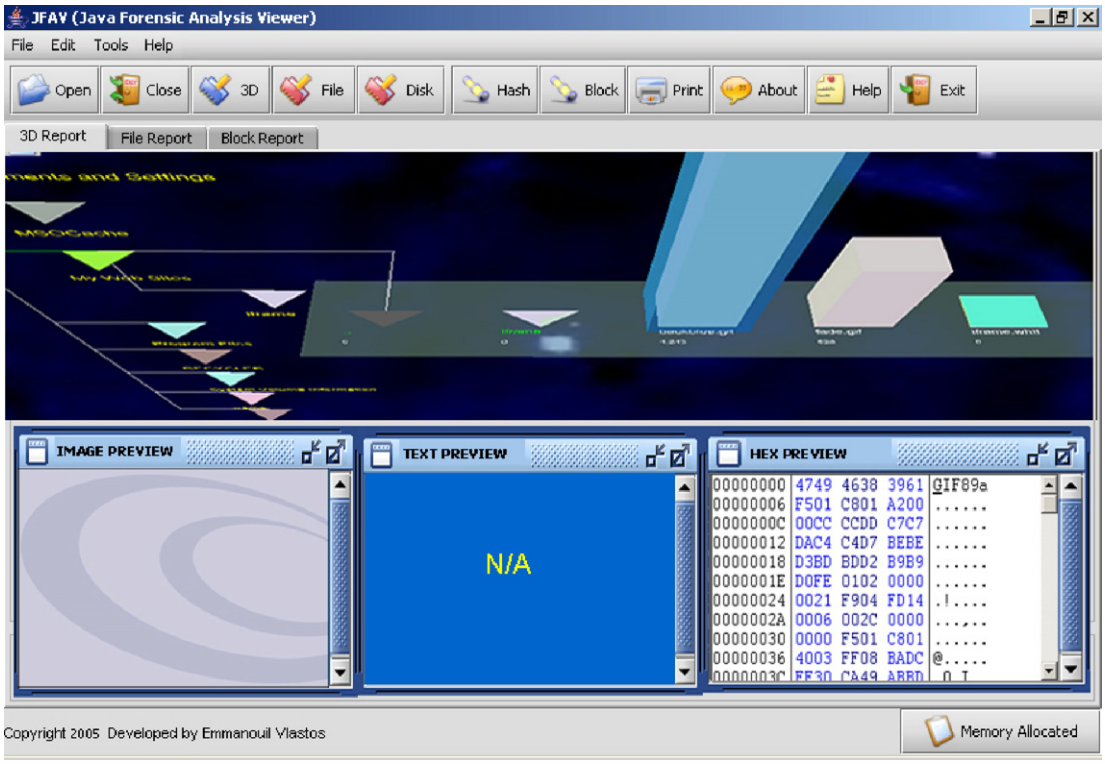

Fig. 7. 3D tree view screenshot.

are situated and separated into several categories under the Files subtree. Fig. 8 shows the panel screenshot which can be found in the left side of the panel. By selecting a leaf in the tree, the user can see its details in a text area next to the tree and the user has the option to save the tree list in a plain text format. Also the user can save the data of a selected file in its original format. For

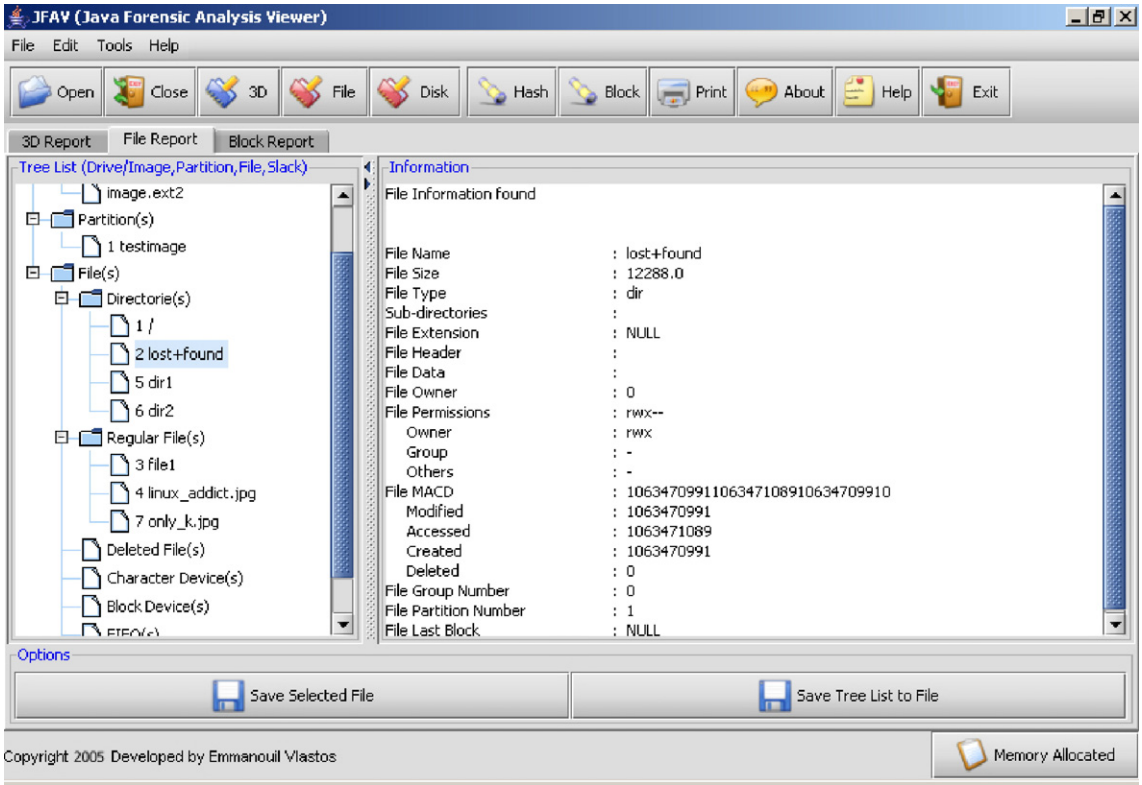

Fig. 8. Tree panel screenshot.

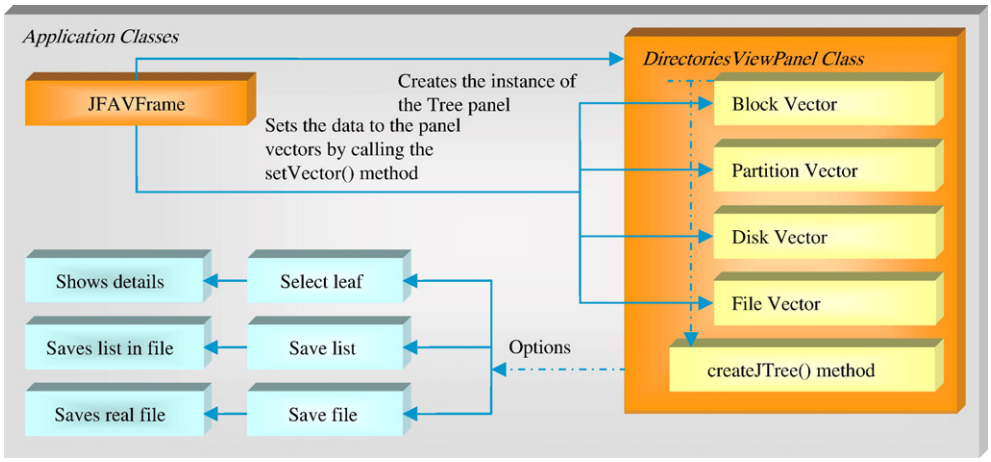

Fig. 9. Tree panel architecture.

these options two buttons in the lower part of the panel were added. The diagram in Fig. 9 shows the interactions in an architectural perspective of this panel.

12. Text panel architecture

The third panel, which is the last view option, has to do with more technical reporting of the retrieved data. It shows the data of each block found in the image of a disk drive in hex and plain text formats. The text panel gives the option to the user to find a block in many possible ways, as shown in Fig. 10. A screenshot of the panel, shown in Fig. 11, illustrates the following features:

- a dynamic slider, located in the right side of the panel, which ranges over the blocks found in the examined partition;
- a button called “go there” that allows the user to give a specific block number and views directly the data of that block;
- “previous” and “next” buttons to navigate slowly between blocks in numeric order.

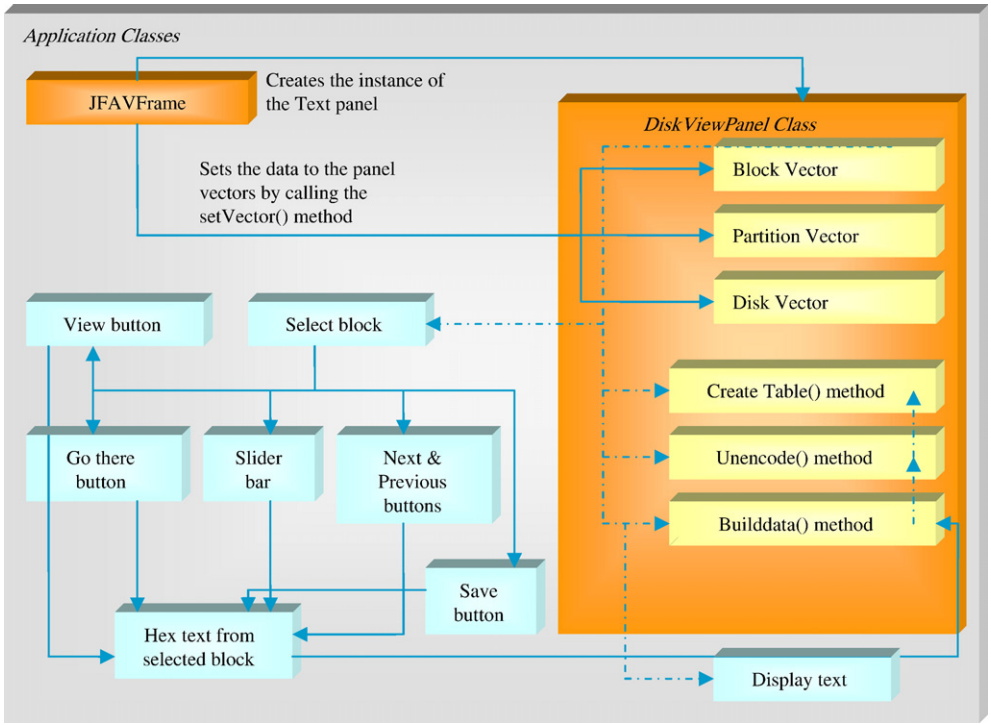

Fig. 10. Text panel architecture.

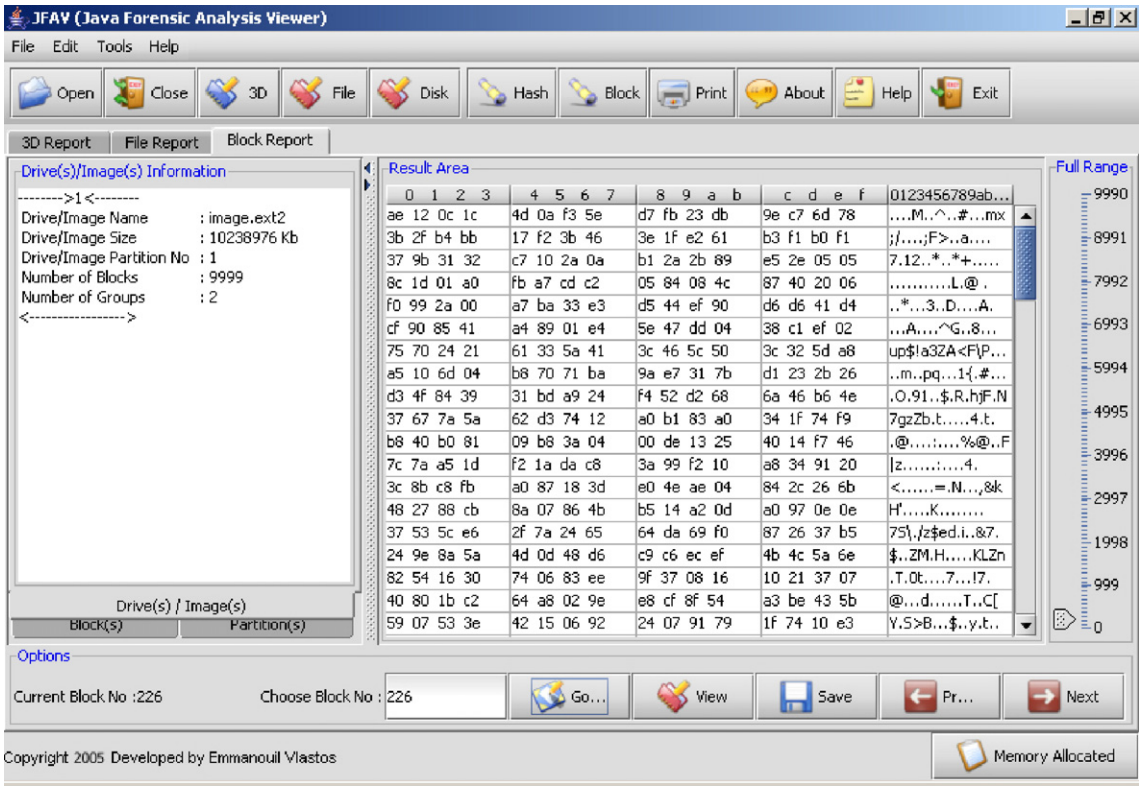

Fig. 11. Text panel screenshot.

- two extra buttons called “view and “save” for viewing separately from the program the data of a block and saving that data in a plain text file.

In all the diagrams above the single dashed arrows indicate the extra classes or packages that were used during the development, the double dashed arrows indicate the internal calls of methods from the parent class, and the single line arrows indicate an action taken during the runtime.

An important issue faced during the development of the visualization tool was the large amounts of data that the parser had to retrieve each time a new XML file was loaded. This was the main reason for the program to be too slow when parsing the XML file. For instance, consider an image file that contains 100,000 blocks and 1000 files, and the data of both blocks and files should be included in the XML file. Given the typical block size of 1024 bytes, its representation in hex (as required by the chosen encoding for XML) is 2048 characters. The time that the parser needs to read all that amount of information is too long. The solution to this problem was to use two new files where one contains only the data of the blocks and the other one the data of the files (block_data, file_data), allowing the presentation tool to find the data in each file separately and sometimes in parallel, thus making it efficient.

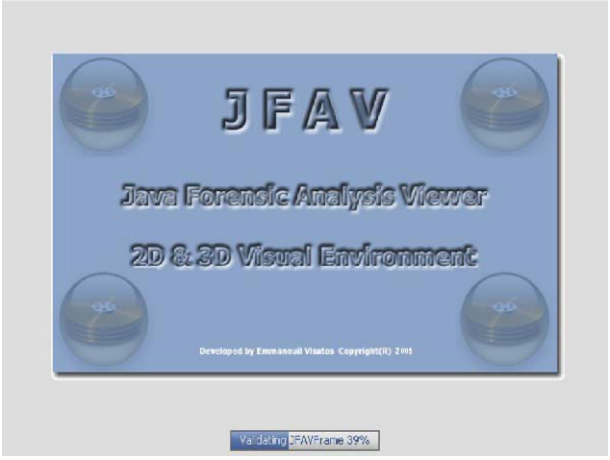

Fig. 12. Splash screen.

13. Case study

To put our tool in context, let us consider a real forensic example where we have an *ext2*[4] partition, under GNU/Linux operating system, on a hard disk drive with the size of 20 MB. With the help of the other tool (FET tool) three files were created to represent the data found in that partition. FET, firstly, creates an XML file containing all the details concerning the structure of the partition, then creates a file containing all the block data and finally a file containing all the file data of the files found. When that phase is finished, the JFAV tool starts (see Fig. 12). The user opens the XML file (causing JFAV to read all 3 files) and begins the investigation. A confirmation dialog appears just to show that the parsing phase was completed successfully.

[4] A GNU/Linux filesystem.

After that, the data read is shown by the tool for investigation. (Figs. 5, 6 and 9) above show a first look at the presented data in 3D, Tree and Hexadecimal views respectively.

The user can check if the presented data are the same as the original data in the partition being examined. By opening the image of the partition with another forensic tool we can see that the data in specific blocks are the same in both tools. This means that FET extracted the data correctly, and JFAV presented them as they are in the partition without any loss of information. Then by choosing the Tree view in JFAV the user can save any file from the partition examined.

For this case study we can check some performance data including time and memory consumption. For this reason the application provides to the user a very useful tool which is responsible for showing the memory allocated by the program at any time as illustrated in Fig. 13.

Table 1 shows typical measurements made using different partition sizes and loading different number of drawn objects (box refers to one block).

14. Results and evaluation

In general, JFAV acted without any significant problems at runtime. It has been realised that some options offered can be improved or replaced by other, more useful ones, but this is now put on our enhancement list of things to do in the future. More errors must be caught before they occur in order to achieve higher stability. Some important extensions that might be considered for further improving the functionality of this program are as follows.

- *Support for filesystems*: It is very important for the visualization program to support all the available filesystems because it increases its portability and extensibility to all the platforms and operating systems. For example, consider a presentation tool that can visualize data from any filesystem, no matter what the size of the disk and of the data. In addition the tool should be capable of being executed on any platforms and most modern operating systems. A very good understanding on the filesystems and how they manipulate the data in a storage device is a prerequisite before such improvements can be applied to the program.

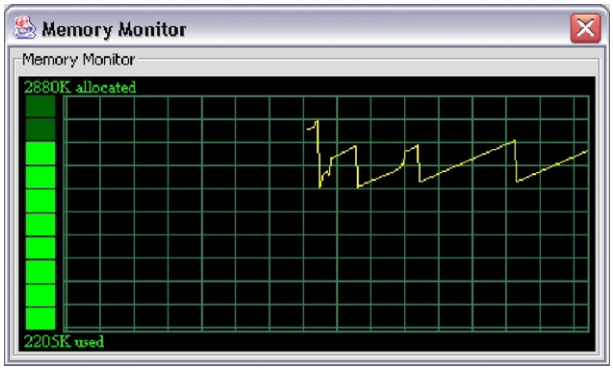

Fig. 13. Memory monitor.

Table 1
JFAV performance measurements

Action	Partition size (mb)	Time (s)	Memory (kb)
Open file	20	1.5	4584
Load 50 boxes		0.87	6388
Load 100 boxes		0.45	16,843
Load 150 boxes		1.3	21,352
Load 200 boxes		0.6	26,543
Open file	100	5.8	7506
Load 50 boxes		0.94	9388
Load 100 boxes		0.59	18,843
Load 150 boxes		1.28	23,352
Load 200 boxes		0.72	29,543

- *Extend the 3D graphics*: The major issue here is the memory load if more than 200 objects will draw simultaneously in the screen and then the program crashes with the "Out of Memory" error. This problem can be solved by implementing an algorithm that will check how many objects are found in the XML file and then will change the size of the object in proportion to how many objects found. The design of the 3D graphics part of the program has been implemented in such a way that it can be extended very easily. Some additions to the 3D graphics could be a mechanism to draw a file as a 3D object but instead of a solid colour on it, it will draw the data of the file. Another issue is the navigation control of the 3D world with the keyboard and the mouse as well. Keyboard navigation can be improved by imposing limits in the 3D environment, disallowing further movement when those limits are reached. The mouse movement can be added to offer a multidimensional movement on the world.
- *Improvement in parsing the XML file*: Assuming the fact that the data of each block and of each file is contained in the XML file, the only way to improve the performance of the parser is to implement a new parser that will not be able to parse any other XML file except the specific one that complies with the DTD grammar. A possible solution, avoiding the implementation of the parser, is for the FET tool to provide together with the XML file the original file of the partition in the storage device. Then the JFAV tool will search in that file with the correspondent offset and size, in bytes, for seeking the data, instead of creating a new file (block_data) with the data.

15. Summary discussion

During the initial development, the program looked like a common forensic tool to visualize digital evidence. Subsequently, it became apparent that building a 3D world to visualize evidential information made the program heavier in terms of development effort, memory requirements and runtime performance, but it was more effective for the eye to see intuitively, and largely unique compared to other open source forensic tools available. Although a 3D world is not essential for a professional to work on and solve real forensic cases, it is useful in quickly understanding the evidential information in a user friendly, intuitive and visual manner. One possible extension to

the 3D view is to implement a layered structure in 3D, like a large building with many floors and rooms, giving a global view of the forensic evidence. This would be in line with what the general purpose Starlight system offers today. It is deemed that this will help the analyst and investigators to go through complex data in a forensic analysis in an easy-to-use, painless and confident manner.

16. Why go for open source model as standards?

3D graphical tools are very complex software systems. Giving away parts of their design and source code is also a complex business, economic and security issue. It is widely acknowledged that the debate between open source software and closed source software is one of the most crucial issues in IT, computer science and especially in software development. Analysts and specialists are trying to find out which one of these two methods is most appropriate in different circumstances and environments, but there's still a long way to go for the discussion.

The open source model works best when the users of software are also developers, and when there are enough of them to sustain and share the development and maintenance effort. Small, highly specialised non-technical (or, more precisely, non-programmer) user communities are served by closed source software manufacturers who make, sell and support high margin, low volume, niche software. There is typically zero availability of open source solutions.

At the first glance, the target user base of forensic analysis tools is a prime example of such a community. It is no surprise that most other forensic tools currently available are closed source. Because users are not perceived to be able to contribute effectively to the development of such software, developers view open-sourcing as a competitive disadvantage.

Another point in favour of closed source model for forensic analysis tools arises specifically from the law enforcement domain: certification. It is envisaged that only certified software will be allowed in the chain of custody and for presenting evidence in a court of law. Certification is expensive business, and revenues from software sales are needed to recover the cost. More importantly though, certified software cannot be changed without re-certification, which makes open-sourcing pointless.

However, there are many reasons why open source model can be viable and advantageous. For instance, certification is only important for the relatively simple trusted chain-of-custody software; the much more complex analysis tools which help experts find, identify and match relevant pieces of digital evidence can be uncertified — as long as the end result can be demonstrated with certified tools.

Aside from lower costs, there are other strong points for the open source model, characteristic of the forensic analysis domain. They include training, the freedom to collaborate and improve the software, and software re-use.

Using these tools is highly skill-hungry, and therefore training is very important. Open source software is generally strongly preferred for training because of no-surprise licensing (it can be freely installed in as many copies as needed, taken home, given multi-user access etc.) and to help (few, but precious) curious students willing to get their hands dirty "under the bonnet". Cost also becomes an issue, as training tends to be a much higher-volume activity with respect to software usage than subsequent professional use.

The freedom to improve the software – the landmark feature of open source – also applies in this domain, considering that investigative departments working with digital evidence often employ programmers for in-house tasks supplementing the main software tools. This fits well with the user–developer principle of open source. Moreover, such departments are often tax-funded, and therefore in many countries are obliged to make their developments publicly available. It makes perfect sense for them to adapt an open source model to facilitate effective collaboration with other departments and user-driven software development, while closed source would stifle innovation and collaboration, hurting efficiency and public policy.

Forensic analysis tools have to deal intensively with technical details related to filesystem structure and layout. A wealth of open source software to do that is available under licenses which require derived products to also be open source. Filesystem software used in popular open source server platforms running such operating systems as Linux or FreeBSD[5] is bulletproof-tested by millions of users, 100%-compatible with on-disk images, and is readily available to an open source tool developer. A closed-source software developer has to re-implement this functionality and continuously keep pace with all new features and filesystems.

The modular approach taken by this project, which separates evidence gathering and analysis, linking them with an open standard, is proposed to facilitate both open and closed source sides. An open, extensible and full-featured standard for data exchange is crucially important to achieve interoperability between commercial and user-driven, closed source and open source, certified and uncertified software tools and thus to take the best of both worlds.

Digital forensic analysis tools require an increasingly high degree of presentation of content in an easy to understand form as well as an ability to export to other more powerful tools for further extraction, analysis and presentation. The idea of separating these tools and applying the concept of open source and close source to make them available freely has been introduced a few years ago by researchers and developers [7,8].

17. Conclusion

This paper described the design and implementation of a visualization tool called JFAV for visualizing data extracted from storage devices and represented using an XML file method to give a more comprehensive view of digital evidence. Many strengths and weaknesses identified in designing and developing the JFAV tool have been highlighted throughout this paper. The technical approach and design decisions taken during the development of the tool were discussed and described together with the system architecture of our tool. The

[5] FreeBSD is a registered trademark of The FreeBSD Foundation.

currently implemented visualization styles presented in this paper are only the beginning of more advanced features to come in future versions of the tool. The research project, although very small in terms of resources and funding when compared to very large projects, offers both extensibility as well as flexibility for further improvements in an open source environment, geared for developing tools specifically for digital investigation purposes.

Acknowledgements

We would like to thank Ian Sutherland and Ioannis Koukouras of the Computing Department of University of Glamorgan for collaborating in this project and Panos Soufleris for testing the implemented visualisation software tool during the development phase of the prototype system. We also wish to thank Nikita Schmidt for exercising and taking the JFAV tool through its paces, as well as offering his numerous suggestions for improvements to the tool and to this paper.

References

[1] EnCase Forensic Edition (2006), see http://www.digitalintelligence.com/software/guidancesoftware/encase/.
[2] The Sleuth Kit (2006), see: http://www.sleuthkit.org/.
[3] The Coroner's Toolkit (2006), see http://www.porcupine.org/forensics/tct.html.
[4] LTOOLS (2006), see http://www.it.fht-esslingen.de/~zimmerma/software/ltools/ltools.html http://www.linuxjournal.com/article/4138.
[5] Starlight (2006), see http://starlight.pnl.gov/.
[6] Ext2fs Home Page (2006), see http://web.mit.edu/tytso/www/linux/ext2.html http://e2fsprogs.sourceforge.net/ext2.html.
[7] Brian Carrier, Open source Digital Forensic Tools, The Legal Argument, see: http://www.digital-evidence.org/papers/opensrc_legal.pdf.
[8] Rob Grey, Commentary: A Comparison: Open Source vs. Closed Source in Operating Systems, 2006 see: http://www.captechventures.com/news/commentary/grey1.asp.