Bringing Domain-Specific Languages to Digital Forensics

Jeroen van den Bos
Netherlands Forensic Institute
Laan van Ypenburg 6
2497 GB, Den Haag
The Netherlands
jeroen@infuse.org

Tijs van der Storm
Centrum Wiskunde & Informatica
Science Park 123
1098 XG, Amsterdam
The Netherlands
storm@cwi.nl

ABSTRACT

Digital forensics investigations often consist of analyzing large quantities of data. The software tools used for analyzing such data are constantly evolving to cope with a multiplicity of versions and variants of data formats. This process of customization is time consuming and error prone.

To improve this situation we present DERRIC, a domain-specific language (DSL) for declaratively specifying data structures. This way, the specification of structure is separated from data processing. The resulting architecture encourages customization and facilitates reuse. It enables faster development through a division of labour between investigators and software engineers.

We have performed an initial evaluation of DERRIC by constructing a data recovery tool. This so-called *carver* has been automatically derived from a declarative description of the structure of JPEG files. We compare it to existing carvers, and show it to be in the same league both with respect to recovered evidence, and runtime performance.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; H.2.3 [**Information Systems**]: Languages—*Data description languages (DDL)*

General Terms

Design, Experimentation, Languages

Keywords

Digital forensics, domain-specific languages, data description languages, model-driven engineering

1. INTRODUCTION

Digital forensics is the branch of forensic science where information stored on digital devices is recovered and analysed in order to answer legal questions. The continuous growth of storage size and network bandwidth and the increased popularity of digital hand-held devices, makes digital forensics investigations increasingly dependent on highly customized data analysis tools. Only the use of extensive automation offers a means to deal with the scale of current and future investigations. Apart from raw scale, the diversity in types of devices, storage and memory layouts, protocols and file formats requires an equally impressive flexibility in these tools: in order to deal with emerging and changing data formats they must be continuously evolved, customized, and redeployed.

Data formats are often poorly documented and hence must be reverse engineered. Even if data formats are documented, there are often many variants that require changes to the implementation of the data format processor. Additionally, off-the-shelf data format processors such as spreadsheets or image viewers are often inadequate, since in digital forensics, one often has to deal with incomplete or otherwise corrupted data: such fragments may contain crucial evidence.

The challenge for software engineering in digital forensics is therefore:

> **How to construct high-quality data analysis tools that are easy to modify and customize, and yet at the same time are able to handle data in the terabyte range?**

To achieve both the required scalability and flexibility we propose an architecture that separates the development of the data analysis tools from the data format processors. This allows the data analysis tool to be optimized for maximum scalability and define how data format processors must be implemented to be usable in the tool. Additionally, data format processors are developed using a data description language that allows declarative specification of data formats. These specifications are then transformed using a code generator into the form the data analysis tool requires. This approach simplifies development (by separating data formats from processing algorithms) and allows for optimizations (by the code generator, either based on data analysis tool requirements or opportunities in the data formats).

For data description, we propose a domain-specific language called DERRIC that is designed to accomodate the workflow of a digital forensics investigator, implementing constructs that match typical activities such as reverse engineering, iterative development and using data format documentation. To evaluate our language, we describe its use in a typical forensics scenario. Additionally, to evaluate our entire architecture, we develop an instance of our system implementing a typical digital forensics data analysis tool

ICSE '11, May 21–28, 2011, Waikiki, Honolulu, HI, USA

doing file carving, the process of recovering deleted, hidden or obfuscated files from a data storage device. We compare our tool to existing relevant file carvers and show that our system performs as good as industrial-strength carvers while being much more flexible.

This paper makes the following contributions:

- An analysis of the software engineering challenges in digital forensics, mapped to practical issues.
- The digital forensics-specific data description language DERRIC.
- An evaluation of a DERRIC-based data analysis application in comparison to industrial-strength tools on standard benchmarks.

Organization of this paper

The rest of this paper is organized as follows. Section 2 discusses the software engineering challenges in digital forensics and maps them to practical problems in data analysis tools. Section 3 presents the data description language DERRIC, demonstrating its use in a typical scenario. Section 4 presents an instance of our complete architecture in the form of a file carving tool utilizing DERRIC. The evaluation comparing our system to existing file carvers is also presented. Section 5 discusses issues around suitability and applicability of our work. Section 6 discusses related work both in the area of domain-specific and data description languages as well as in digital forensics. Section 7 concludes.

2. DIGITAL FORENSICS CHALLENGES

The most important challenges in digital forensics include domain-specific data abstraction, modularization and improving scalability [10]. Data abstraction deals with the need for one or several standard formats to describe, store and use data in different formats. Modularization refers to the need to increase and deepen integration between digital forensics tools to reduce manual preparations and allow extensive reuse between types of tools (e.g., using the same tool to recover images from both a storage device and a network stream). Scalability is important to keep digital forensics investigations feasible in the face of current and future storage capacities, bandwidth and device use. Below we discuss the domain-specific aspects of these challenges in more detail.

2.1 Data Abstraction

Digital forensics investigations typically require the support of a large amount of data formats, ranging from file systems and formats to protocols and memory layouts, where each class can have several different instances depending on type, version and implementation. An example of this is the FAT file system, which has multiple types: FAT12, FAT16 and FAT32, where FAT16 has two versions. All types and versions are implemented by multiple operating systems.

Reverse Engineering

Whenever data is encountered in an unknown (or known, but proprietary) format, a process of reverse engineering starts to recover enough of the format's structure to be able to recover files of this type or extract information from recovered files.

A common problem is the distance between the encoding of identifiers discovered in the data under investigation and the format in which they must be expressed. If the notation doesn't support the same encoding, the data must first be transformed. Besides being error prone, it also obfuscates the description. Examples of different encodings are string encodings such as ASCII and unicode and numerical values of any bit size.

Fragmented or missing data is also common. If a data description method does not support the expression of parts of the data that are currently unknown, the rest of the format can either not be expressed or the unknown data must be described using some approximation. This prevents the tool using the description from using the knowledge of missing data to its advantage by choosing a method appropriate to its requirements and capabilities. Additionally, expressing what parts of a format are currently unknown instead of some arbitrary placeholder increases its value as documentation.

Using Documentation

In another situation, the format of the data that is encountered in an investigation is well-known and documented. In this case the documentation is used to create an implementation of the format in order to recover files of this type or extract information from recovered files.

A similar problem occurs here regarding the distance between the encoding of identifiers in the documentation and the format in which they must be expressed. Although data format documentation tends to map relatively cleanly to implementations in data description or programming languages, some important exceptions exist. The most common is in the formatting of strings, where data formats typically still use ASCII strings, the default format for strings in programming languages has typically evolved to a type of unicode, or is dependent on external factors such as compiler options, linked libraries or runtime platform. This functionality typically exists so applications can easily be adapted but may have unwanted consequences in a data format processor.

Another problem related to the encoding issues is that documentation may present data in a different format on purpose. An example is the Microsoft Office file formats documentation [23], that displays all bit diagrams in big-endian byte order for readability even though it requires implementations to store the actual files in little-endian byte order.

Iterative Development

Regardless of the approach used to develop the data format description, the process is typically highly iterative for several reasons. The smallest possible description that will reliably lead to recovering evidence is always sought since strict deadlines are common. To find this description, it is developed iteratively, checking at every increment whether it succeeds. This effectively requires the process to go from describing to executing to be simple and fast. In an ideal situation, this means that data analysis tools can be reconfigured or extended at runtime, or have capabilties to easily and quickly shutdown and restart.

2.2 Modularization

The diversity in types of digital forensics investigations is high, ranging from the analysis of a regular confiscated data storage device, such as a hard drive, to a highly specialized

embedded device such as a detonator. At the same time, as reuse of techniques and formats between devices is high, the reusability of the tools analyzing them should be as well. An example is a file system such as FAT, which is typically used on (older) desktop computers and servers, but also on thumb drives, memory cards and on internal memory of all kinds of embedded devices such as mobile phones and MP3 players. Interfacing with these devices often requires different hardware and accompanying software, but at some level the analyses converge and boil down to support for the FAT file system layout. Modularization can facilitate that each data format must be developed only once and then used in multiple scenarios on multiple devices.

Adding and Modifying Formats

All these independent implementations of standard data formats are rarely identical, prompting digital forensics investigators to regularly implement small changes or create derived versions of popular formats. Any implementation that is far removed from the specification of data formats will be difficult to use for regular adaptation.

For example, if a hand-written parser is used, making a small change such as changing the sign of all numbers in a data format can have significant impact on the entire implementation, such as having to change all the number variable declarations and changing all calls to parsing methods related to numbers. Apart from being time consuming it is also error prone and difficult to verify.

Modifying and Reconfiguring Tools

The combination of diversity and similarity in the domain of digital forensics leads to additional complexity. An extremely rare combination of data formats in some areas may be very common in another. To analyze data efficiently, different investigations benefit from different combinations of algorithms and formats, each optimized for both a specific type and amount of data encountered.

An example is the analysis of the contents of a confiscated hard drive. In one investigation all files of certain types may be identified and recovered. In another however, time may be extremely limited and the investigators may be looking for a possibly hidden spreadsheet created using Microsoft Excel 2007. To accomplish this, they may want to look for all ZIP files containing XML files (since Excel 2007 files are basically a set of XML files compressed with ZIP). The more difficult it is to modify or reconfigure an application to perform this analysis, the less time the investigators will have to do other analyses.

2.3 Scalability

For the past thirty years, the cost of hard drive storage has shrunk exponentially as every fourteen months the price of a single gigabyte has halved [18]. Coping with the amount of extra data would already be challenging in just this dimension, but there are more dimensions that show similar growth. The amount of households with broadband connections is steadily growing and in The Netherlands, there have been more active mobile phone subscriptions than citizens since 2006 [5]. Additionally, the digital world is becoming more and more diverse, with desktops running Mac OS and Linux operating systems slowly becoming more widespread and users choosing alternative browsers on any of these platforms are already common.

As a result, data analysis tools must scale to support these exponential increases in size as well as be able to identify and recover an increasing amount of different data formats.

Scaling To Terabytes

From a hardware perspective alone it is already challenging to have to analyse the largest hard drives available or network streams that do not fit on a single disk of the largest available size. The demands this places on the data analysis tools are even greater. Exponential growth in the encountered data means that analysis techniques and algorithms have to be extremely refined in order to be usable for any length of time before they become too slow. When they do, it is typically a lot of work for developers to modify a data analysis tool to work with new techniques that have been optimized for the current generation of data sizes.

If the base functionality of these data analysis tools, such as reading and caching data as well as implementing identification and recovery algorithms is tangled with other concerns, especially related to identifying and recovering data formats, then every scalability enhancement will have to be applied to each data format implementation. This means that as data grows and more data formats come into use, not only will changes have to be made more frequently, they will also be more complicated every time. Eventually the data analysis tool will become unmaintainable.

Trading Precision for Speed

As mentioned in section 2.2, different types of investigations may be more efficient in a custom configuration using only a specific set of data formats and a single (type of) algorithm. However, there are also cases that this approach cannot be used to save time, for instance when there is not enough information about what to look for or how to look for it. When time is limited, a typical approach can be to simply reduce the precision of all parts of the system and end up with a best effort result given the time available.

If this requires a large amount of manual modifications across a large set of components, several problems arise. The first are typical for modifying software, such as making a lot of changes under time pressure being error prone and difficult to trace. Additionally however, a set of components developed by multiple developers across a large period of time will most likely consist of very different looking and functioning code, making it extremely difficult to modify all components in such a way that they all lose a comparable amount of precision and gain the same in performance. The result will be an unevenly optimized data analysis tool with difficult to predict performance characteristics.

3. A DSL FOR DIGITAL FORENSICS

Specifying data formats is one of the main challenges identified in Section 2, so a data description language (DDL) [8] forms our starting point. We have developed Derric, a DDL designed to address the problems related to data description in digital forensics. In the following subsection we will present the language using an example, the description of JPEG [14]. The JPEG format is one of the most important data formats in digital forensics investigations, given that nearly all digital cameras and mobile phones produce files of this type and it is also the most prominent format for pictures on the world wide web.

```
1 format JPG
2
3 unit byte
4 size 1
5 sign false
6 type integer
7 endian big
8 strings ascii
9
10 sequence SOI APP0JFIF APP0JFXX? not(SOI,
11   APP0JFIF, APP0JFXX, EOI)* EOI
12
13 structures
14 SOI { marker: 0xFF, 0xD8; }
15
16 APP0JFIF {
17   marker: 0xFF, 0xE0;
18   length: lengthOf(rgb) + (offset(rgb) -
19     offset(identifier)) size 2;
20   identifier: "JFIF", 0;
21   version: expected 1, 2;
22   units: 0 | 1 | 2;
23   xdensity: size 2;
24   ydensity: size 2;
25   xthumbnail: size 1;
26   ythumbnail: size 1;
27   rgb: size xthumbnail * ythumbnail * 3;
28 }
29
30 DHT {
31   marker: 0xFF, 0xC4;
32   length: size 2;
33   data: size length - lengthOf(marker);
34 }
35
36 SOS = DHT {
37   marker: 0xFF, 0xDA;
38   compressedData: unknown
39     terminatedBefore 0xFF, !0x00;
40 }
```

Figure 1: Excerpt of the JPEG format in Derric

3.1 An Example: JPEG

A Derric description is fully textual and consists of three parts: a header, a sequence and a set of structures. As an example, an excerpt of the JPEG image file format description is shown in figure 1.

Following is a discussion of how Derric addresses the domain-specific aspects of data description in digital forensics using the JPEG format description as an illustration.

Specification and Implementation Encoding

Derric allows literal values to be expressed in a large amount of different formats, tailored to different ways the data may be encountered in an investigation. In the case of reverse engineering, this will typically be in hexadecimal format. When documentation is used, other literals may be appropriate. The JPEG format description in Figure 1 demonstrates several formats: line 14 shows hexadecimal and line 20 shows a string literal in combination with a regular decimal number. Additional formats are supported, including octal and binary.

In addition to the multiple formats for expressing values, modifiers exist to direct the interpretation of values. Modifiers exist to transform values based on byte ordering (little, big and middle endian), sign, numerical type (integer, float), string encoding (ASCII, UTF-8/16/32, etc.) and size (with different units, such as bits and bytes). Default values for modifiers can be expressed at the top of a Derric description. An example of this is shown in lines 3–8. In this case, `"JFIF"` and `0` in line 20 will be interpreted as an ASCII string and a single byte, unsigned integer respectively.

Not requiring data format developers to transform data before use reduces the distance between actual data and data descriptions, thus improving usability and readability.

Expectations and Unknowns

Whether reverse engineering or working from documentation, some fields in a data format may have a lot of different values, but typically do not in practice. An example of this is the *version* field on line 21. Even though different versions of the JPEG format do exist, the 1.2 version is encountered nearly exclusively. Therefore, the value of the *version* field should formally be defined as any value. When attempting to reassemble a heavily fragmented JPEG file that has been cut off just before the *version* field however, it may improve performance dramatically to first try parts that start with the most common value for that field. The `expected` keyword in Derric allows the investigator to express this information as a hint to the analysis tool.

The opposite of having additional information about a field may also occur: not understanding the contents of a field completely and specifying whatever part is known or guessing. To facilitate this, Derric has the `unknown` keyword, as shown in the *compressedData* field on lines 38–39. The application using a description using the `unknown` keyword may decide to take the field's specification with a grain of salt (e.g., when the specification doesn't completely match, continue anyway to determine whether the rest of the data does match).

Allowing investigators to express additional or missing information about a data format as part of the specification enables an iterative style of development.

Modification and Variation

Decoupling the ordering into a separate sequence makes it easier to extend a description. Instead of specifying ordering at data structure level (e.g., as a linked list, which is common practice in many programming languages) a distinct sequence allows specifications such as on lines 10–11, where the `not` keyword used in `not(SOI, APP0JFIF, APP0JFXX, EOI)*` automatically includes all data structures except the ones specified. Adding a data structure automatically adds it to the sequence, which maps well to the process of reverse engineering where discovering previously unknown data structures is common. Additionally, if the `sequence` keyword is not specified, a sequence is inferred where any combination or ordering of specified data structures is accepted.

Data formats often have some fixed characteristics that are shared by most internal structures. In the case of JPEG, as shown by the *DHT* structure on lines 30–34 in Figure 1, this is a 16 bit *marker*, followed by an unsigned 16 bit integer *length* specifying the size of the data structure (in this case, apparently excluding *marker*) and finally the payload named *data*. Support in Derric for inheritance makes it easy to add another structure. As shown in the specification of the

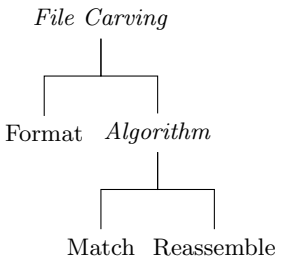

Figure 2: Variability in the file carving domain

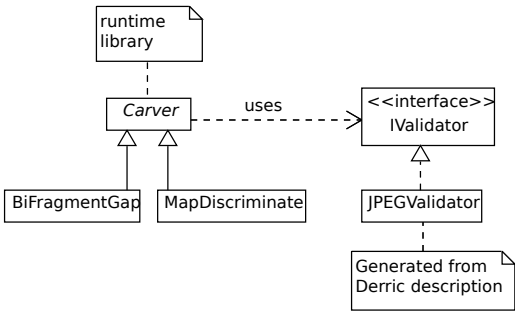

Figure 3: Overview of the Excavator architecture

SOS structure on lines 36–40, the *SOS* structure overwrites the *marker* field, reuses the *length* and *data* fields and then adds the *compressedData* field.

Decoupling the sequence from data structure specifications and inheritance make data descriptions shorter and help group related information, improving readability and expressiveness of the language.

4. APPLICATION: CARVING

We have evaluated DERRIC in the domain of *file carving* [24], which is the process of recovering deleted, fragmented or otherwise lost files from storage devices. The complete description of Figure 1 has been input to a code generator to obtain a JPEG validator. Such a validator can be used by dedicated carving algorithms [9] to recover evidence from disk images. The complete system including file format descriptions in DERRIC, code generator and runtime library is named EXCAVATOR.

4.1 Concerns in the Carving Domain

Analysis of the carving domain uncovers three concerns that are variable across typical carver implementations: (1) Format, (2) Matching and (3) Reassembly. A schematic overview of this variability is shown in Figure 2. The first type of variability entails that for each type of file that must be recovered, the file format must be defined. Carvers must know the structure of, for instance, JPEG in order to recognize that a certain sequence of bytes might be part of a valid JPEG file. Additionally, some file formats exist in different versions and variants. For instance, the Portable Network Graphics (PNG) format has three official versions [32]. Finally, manufacturers of digital devices such as mobile phones or digital cameras may implement a file format standard in idiosyncratic ways, which could be valuable for recovery. We consider all kinds of variation to be covered by the "Format" concern.

The second dimension captures (1) the ways in which files are matched in the input image, and (2) the method of reassembly if fragmentation is detected on the basis of file format structure. In Figure 2 these variation points are indicated as "Match" and "Reassemble" respectively, below the abstract "Algorithm" concern.

There are at least three matching algorithms that are used in carvers. The most basic matching algorithm is header/-footer matching that returns blocks between signatures of file headers and footers. Next, file structure-based matching uses complete structural knowledge of a file format in order to deal with, for instance, corrupted files. Finally, characteristics-based matching takes (statistical) characteristics about a file's contents into account, for instance high entropy in compressed files.

Finally, the third concern consists of algorithms for reassembling fragmented files. For instance, bifragment gap carving [9] assumes that files consist of only two fragments and that they are located on the data storage device in the correct order. The algorithm tries all possible gaps between the matched beginning and end of the file. Map/generate [6] is more elaborate in that it supports reassembling files that are arbitrarily fragmented. It exercises any combination of sectors and then prunes the search space if mismatches are found.

Currently, file carvers implement a limited combination of file formats and/or matching and/or reassembly algorithms. Off-the-shelf carvers typically do not support explicit variation points to efficiently make trade-offs between precision and performance. The implementation of data format, matching and reassembly is completely tangled. As a consequence, modification or reconfiguration of carvers is time consuming and error prone.

Additionally, the top-level dimensions of Figure 2, "Format" and "Algorithm", correspond to two different roles in the practice of using carvers in forensic investigations. On the one hand there are the digital forensics investigators that have intricate knowledge of many file formats. On the other hand, there are the software engineers that know how to implement, evolve, and optimize carving tools. With the current tools, no division of labour is possible: domain-specific knowledge about file formats has to be communicated to software engineers in order for them to make the necessary changes to the system.

4.2 Implementation

Each concern of Figure 2 corresponds to a variation point in the implementation. In EXCAVATOR, each variation point corresponds to a logical component. These components are:

1. The declarative surface syntax of DERRIC for describing the structure of file formats (*Format*).
2. A code generator that takes file format descriptions and generates matching code (*Matching*)
3. A runtime library implementing reassembly algorithms as well as defining the base types and interfaces for the generated matching code. (*Reassemble*)

Both the file format model and the code generator are implemented in RASCAL [17]. File format descriptions are input

to the code generator. The generator produces Java classes implementing the "Matching" concern. These classes are used by the Java runtime library which contains algorithms for fragment reassembly. Currently, the runtime library contains two algorithms, a brute force algorithm and bifragment gap carving discussed in Section 4.1.

The final component is the code generator. It takes a description such as that of Figure 1, and produces a Java class implementing the matching code that is used by the runtime library. This code generator uses a model-to-text approach [7]. It is implemented using RASCAL's string templates, which are ordinary strings, interpolated with arbitrary expressions and control flow statements.

An overview of the EXCAVATOR architecture is shown in Figure 3. The abstract Carver class captures the reassembly concern; implementations exist in two variations as indicated by the concrete subclasses. A carver uses implementations of the IValidator interface (matching concern). Implementations of this interface are generated from DERRIC file format descriptions.

4.3 Evaluation

In order to evaluate the resulting JPEG carver, we have compared its performance to that of three popular carvers. First, we assert that EXCAVATOR is in the same league with respect to the number of recovered files and runtime performance. For this, the carvers are run on five established benchmarks for carvers. Secondly, we argue that the flexibility induced by the domain-specific language approach of EXCAVATOR is unmatched by the other carvers.

The file carvers that we compare EXCAVATOR to were chosen based on two criteria. First, they are actively used in digital forensics investigations, both in government and industry. This ensures our comparison is relevant. Second, we required the tools to be open source in order to make a source-based assessment of the effort of customizing a carver. These criteria have lead to the selection of Scalpel [26], PhotoRec [11] and ReviveIt [22]. The precise versions and command line options that were used in the evaluation are shown in Table 1. Below, we briefly describe each carver.

Scalpel A high performance, file system-independent and cross-platform file carver written in C. It employs a header/footer based algorithm to recognize files; the structure of headers and footers is described using regular expressions. It tends to generate a relatively large amount of false positives but is extremely fast.

PhotoRec Originally designed to recover digital photographs from memory cards but has since been extended to support a plethora of file formats. This carver is completely implemented in plain C and all logic, file format, matching and reassembly, is hard-wired.

ReviveIt The most advanced of the three carvers. It employs Garfinkel's bifragment gap algorithm [9] and is configured using an external specification of file formats. This specification is then interpreted at runtime.

4.4 Forensic Benchmarks

The set of benchmarks used in the evaluation of EXCAVATOR consists of five files containing either a byte-for-byte copy of a data storage device or a synthetic data structure with similar properties. The files were selected since they all contain recoverable JPEG files and are widely recognized as benchmarks for carvers.

The size of each benchmark, together with the number of JPEG files contained in it, is shown in Table 2. JPEG 1, Basic 1 and Basic 2 originate from the *Digital Forensics Tool Testing Images* [4] collection, a project set up to share benchmarks that are useful for testing digital forensics tools. They are regularly used to evaluate new algorithms and tools in digital forensics research. DFRWS 2006 and DFRWS 2007 are taken from the Digital Forensics Research Workshop's (DFRWS) Forensic Challenge in 2006 and 2007, when the challenge focused on file carving. Together, the benchmarks exercise file carvers in nearly all relevant areas, such as recovering deleted files, reassembling fragmented files, ignoring placed false positives and dealing with file system-specific issues. Below we briefly describe each benchmark.

JPEG Search Test #1 An NTFS file system containing JPEG files in various disguises. Additionally, some traces of JPEG headers and footers have been placed in strategic locations, to confuse carvers.

Basic Data Carving Test #1 A byte-for-byte copy of a 64MB FAT32 formatted thumb drive including deleted files and some corrupted data structures, including a JPEG header.

Basic Data Carving Test #2 A byte-for-byte copy of a 128MB EXT2 formatted thumb drive including deleted and fragmented files.

DFRWS Forensic Challenge 2006 A 50MB file generated using random data and seeded with, amongst others, fragmented JPEG files which may be interleaved with other JPEG files or hand-crafted headers to confuse carvers.

DFRWS Forensic Challenge 2007 Similar to the 2006 DFRWS benchmark, only larger (331MB) and heavily fragmented.

4.5 Evaluation Details

To compare the existing carvers to EXCAVATOR, we have run all four carvers on all five benchmarks. To ensure the best results, if a tool has multiple modes of operation we have run each benchmark in each mode and recorded the best result—see Table 1 for details on how each carver was run.

File Carving Performance

Table 3 lists the results of our evaluation. The table shows the number of correctly recovered files for each carver, including the recall between parentheses.

Of all carvers, Scalpel recovers the smallest amount of files. The reason is that its simple header/footer matching algorithm prevents it from recovering any fragmented files. PhotoRec performs better, but does not find files that are prefixed with random data (JPEG 1) and has trouble dealing with fragmentation in the EXT2 benchmark (Basic 2). ReviveIt also misses the files that are prefixed with random data (JPEG 1), but does succeed in reassembling more fragmented files than any other tested carver (DFRWS 2006) through its combination of file structure matching, characteristics-based matching and bifragment gap reassembly. Finally, EXCAVATOR recovers several fragmented files as well but misses a few more than ReviveIt because it does not

Tool	Version	Command line
ReviveIt	20070804	`-e -F -t OUTDIR -c ../etc/file_types.conf INPUT`
Scalpel	1.6	`-b -c scalpel.conf -o OUTDIR INPUT`
Photorec	6.11	`/d OUTDIR INPUT`

Table 1: Carvers participating in the evaluation

#	Short name	Name	File name	Size (MB)	#JPEGs
1	JPEG 1	JPEG Search Test #1	8-jpeg-search.dd	10	7
2	Basic 1	Basic Data Carving Test #1	11-carve-fat.dd	62	3
3	Basic 2	Basic Data Carving Test #2	12-carve-ext2.dd	123	3
4	DFRWS'06	Forensic Challenge 2006	dfrws-2006-challenge.raw	48	14
5	DFRWS'07	Forensic Challenge 2007	dfrws-2007-challenge.img	331	18

Table 2: File carving tests participating in the evaluation

implement characteristics-based matching. However, it does recover the random data prefixed files (JPEG 1).

Table 3 shows the number of files that (1) are completely recovered and (2) are actually present in the test image. The first condition is checked by feeding the recovered file to an image viewer. The second condition is verified using the MD5 checksums provided with each benchmark. Any file that is recovered, but is not viewable or does not match an MD5 checksum, is a false positive.

We chose not to include the number of false positives (and hence, the precision) in the results for two reasons. First, when a file matches none of the MD5 checksums, it is not automatically useless in forensic investigations. For instance, it may be a partial file containing crucial evidence. Thus, a false positive is not necessarily a bad thing. Second, the degree as to which a false positive is useless, is hard to quantify. During our experiments, we have observed that some files were partially recovered by multiple tools, but that some tools recovered a larger part than others. Which part of a file is important depends on the case at hand. We have therefore chosen to only measure the number of true positives and recall[1].

Nevertheless, a large number of false positives is not desirable, since they have to be manually inspected. In all of our tests, Excavator had no more false positives than the best performing tool of all the tools in the evaluation.

From the results it can be concluded that Excavator, on average, finds as many files as the other carvers. In fact, the only benchmark where Excavator performs worse than any other carver is DFRWS 2006: ReviveIt recovers two more files because it employs characteristics-based matching. We expect that adding support for characteristics-based matching to Excavator will make it as good as ReviveIt on DFRWS 2006 as well.

Runtime Performance

The runtime performance results are shown in Table 4. On the whole, ReviveIt performs worst on all benchmarks. This can be explained by its use of characteristics-based matching, which requires it to process much more data than the other tools. The other tools typically finish within a couple of seconds with a few exceptions.

The high running times of ReviveIt and Scalpel on DFRWS 2007 can be explained from the fact that the image is much larger than the others, and both tools recover many partial files. On the DFRWS 2006 benchmark, ReviveIt and Excavator use bifragment gap carving to recover some fragmented files that both Scalpel and PhotoRec miss; this explains the additional time required.

Based on the numbers in Table 4, we conclude that the runtime performance of Excavator is similar to the performance of the fastest carvers in these benchmarks. Important to note however is that in real-life digital forensics investigations, the data sets will typically be much larger, since hard drives of several terabytes in size are becoming common. Unfortunately, no publicly available benchmarks exist of this size. As a result, we have not been able to determine how Excavator scales compared to the other carvers.

Flexibility

Efficiently implementing new file formats or modifying existing ones is an important requirement in digital forensics investigations. The carvers in our evaluation all support this requirement with varying degrees of flexibility. Below we provide a qualitative assessment of the domain-specific language approach of Excavator in comparison to the other carvers.

PhotoRec requires a file format definition to be directly implemented in code, along with a matching algorithm. This tangling of concerns makes it practically impossible for a non-programmer to make changes. Furthermore, to leverage advances in matching algorithms, existing file format implementations must be adapted.

Apart from PhotoRec, all other carvers have separate file format definitions that can be modified without altering the application code. The definition that Scalpel uses, however, is very basic: header and footer matching along with some basic options (such as case sensitivity). This means that the built-in header/footer matching is hard to replace with a more advanced matching algorithm. Furthermore, reassembly algorithms typically require the matching to be much more precise in order to do scalable reassembly.

The remaining two, ReviveIt and Excavator, support full file format descriptions. The definition that ReviveIt uses however is tied to concepts of the matching algorithms it

[1]This is in accordance with the rules used in the DFRWS Forensic Challenges.

	JPEG 1	Basic 1	Basic 2	DFRWS 2006	DFRWS 2007	Total
ReviveIt	4 (57.1%)	3 (100%)	3 (100%)	10 (71.4%)	1 (5.6%)	21 (46.7%)
Scalpel	6 (85.7%)	1 (33.3%)	1 (33.3%)	6 (42.9%)	0 (0%)	14 (31.1%)
Photorec	4 (57.1%)	3 (100%)	1 (33.3%)	8 (57.1%)	1 (5.6%)	17 (37.8%)
Excavator	6 (85.7%)	3 (100%)	3 (100%)	8 (57.1%)	1 (5.6%)	21 (46.7%)

Table 3: Number of true positives and recall per carver, per benchmark

	JPEG 1	Basic 1	Basic 2	DFRWS 2006	DFRWS 2007
ReviveIt	11.8s	14.6s	17.8s	37.0s	7m58s
Scalpel	0.4s	1.9s	3.6s	2.8s	21.7s
Photorec	0.2s	0.5s	0.6s	0.2s	3.8s
Excavator	0.2s	0.4s	0.8s	18.6s	3.1s

Table 4: Runtime performance per carver, per benchmark (wall clock time)

implements. For instance, its definitions explicitly mention characteristics-based matching, which in our view belongs to the matching concern and not to the definition of a file format. As a result, these file format definitions are hard to reuse for alternative matching algorithms and even harder for different types of data analysis. Excavator's file format definitions are strictly declarative; both matching algorithm and file formats can be varied independently.

Furthermore, Excavator separates matching and reassembly algorithms, allowing variation between these dimensions in a similar manner. None of the other carvers expose explicit variation points to independently vary matching and reassembly[2]. Excavator can be run using both bifragment gap and brute force reassembly algorithms without having to adapt file format descriptions.

The results of Tables 3 and 4 show that the separation of concerns achieved in Excavator did not incur a penalty in either carving performance or runtime performance. Moreover, this flexibility did not come at the price of more code either. Table 5 shows the size statistics of Excavator. The entire system, currently encompassing the language grammar, JPEG and PNG descriptions, code generator and a runtime library containing two reassembly algorithms, consists of just above a thousand non-commented lines of source code.

5. DISCUSSION

Although techniques such as separation of concerns and declarative specification are commonly regarded as improving quality whenever they are used, it is difficult to assess whether any given solution applies these principles completely, correctly and whether an even better solution could exist. Nonetheless, given the very small amount of code required to develop Excavator and the results achieved, we believe the general effectiveness of the approach is clear. However, some issues surrounding suitability and applicability exist and are addressed in the following subsections.

5.1 Scalability

One of the challenges discussed in Section 2 is improving scalability. To measure this, benchmarks or scenarios must be used that push a data analysis tool to the limit in terms of data size it can handle. However, the largest publicly available benchmark is the DFRWS Forensic Challenge 2007 image, which is included in our test set. At 331MB, this does not come near a size that requires an analysis tool to take special measures in the area of scalability. This challenge therefore has not been addressed.

[2]Scalpel does not support reassembly at all.

5.2 Universal Data Description

Derric can be used to describe any data format, but in the current evaluation has only been used to describe JPEG. The language's usability however depends on its ability to describe a large range of data formats. In order to develop our language, we have described a large set of data formats, including other image formats such as PNG and GIF, along with several document formats such as Microsoft Office Word and Excel and container formats such as ZIP and RAR. To test our language and code generator, these descriptions were successfully tested on sets of files of those types. The benchmarks are all focused on JPEG, so our measurements use these results.

The application we have developed recovers data but does not process it further. Additional capabilities that may be related to the data description language, such as processing embedded files in a container format or detecting encryption are therefore not evaluated. However, recognizing a file's type without processing it further is useful outside of data recovery, for example in network filtering and content detection (in network proxies and web browsers).

5.3 Usability

The eventual users will be the final judge of Derric's usability. Even though we do not have numbers on user satisfaction, we believe that there are several reasons that Derric can be considered an improvement over other approaches. First, when data format processing code is developed by a software engineer, the digital forensics investigator would need to transfer knowledge of a data format to the engineer. Derric provides a tailored notation that can be directly used by digital forensics investigators.

Second, if the investigator develops the data format processor directly, then Derric still only requires the same information that would otherwise need to be expressed in any programming language, but stripped of all implementation details such as memory management. Therefore we believe

Component	Implementation	Size (SLOC)
Grammar	SDF	52
JPEG description	Derric	92
PNG description	Derric	58
Structure-based matching (code generator)	Rascal	510
Bifragment gap (runtime)	Java	72
Brute force (runtime)	Java	44
Utilities (runtime)	Java	256
	Total:	1084

Table 5: Sizes of the Excavator components

Derric can be considered a step forward from direct implementation since it requires nothing more, but does remove a lot of work for the investigator.

6. RELATED WORK

There is extensive work in the area of model-driven engineering (MDE) [27], DSLs [30] [28] [16] and DDLs [8].

The work in [29] investigates the factors that influence industrial adoption of MDE. One of the conclusions is that generic, well-established modeling languages are favoured over more advanced modeling technologies, such as dedicated DSLs. As such, this identifies an open research question regarding our work.

A case-study of MDE in an industrial context is described in [2]. There, the use of MDE has been found to lead to significant productivity and quality improvements. In one division that was investigated 65%–85% of the code could be generated from high-level models. Moreover, the model driven perspective also lead to improvements in some phases of the software process: it turned out that the time to correctly fix a defect was regularly reduced by a factor between 30 and 70. This tremendous gain is attributed to the fact that many defects could be fixed and tested at the model level. This can be seen as additional supporting evidence for the observation that MDE may significantly improve changeability of software.

In [21] a survey of the techniques and tools related to the different stages of DSL development is presented. One important conclusion is that these nearly all focus on the implementation phase and ignore earlier phases such as decision, analysis and design.

A study of the success factors of DSLs is described in [13]. There, learnability, usability and expressiveness of the DSL, reusability of the code and development costs and reliability of the resulting software are identified as the most important factors contributing to the success of using a DSL.

Most data description languages are either tied to a specific type of application, such as PacketTypes [20] and Zebu [3] to network protocols, or technology, such as XML Schema to XML.

Some general data description languages that allow specification of binary formats do exist, such as PADS [19] and DataScript [1]. Of these, PADS supports extensive error handling. Derric distinguishes itself by having a syntax that maps onto common activities in the field of digital forensics such as reverse engineering.

The technology behind file carving is strongly related to parsing [12]. However, traditional grammar formalisms, such as ANTLR [25] and SDF2 [31], are specifically targeted at describing textual computer languages. They are generally unsuitable to build parsers for binary file formats, since these often require complex data dependencies between elements of a file. Data-dependent grammars extend traditional parsing technology to allow the definition of such dependencies [15] and may be usable in some applications of Derric.

7. CONCLUSION

Data storage size and network bandwidth is growing continuously and popularity of digital hand-held devices is increasing. Additionally, the software market is diversifying and growing steadily. This brings serious challenges to digital forensics investigators who must cope with large quantities of data and an evolving set of data formats to consider.

We present a practical interpretation in the area of software engineering of these digital forensics challenges, identifying which activities are directly affected by them, so they can be addressed systematically.

Next, we present the domain-specific language Derric, designed to fit into the workflow of a digital forensics investigator. It allows declarative specification of data formats, thus separating the task of data description from data analysis tool development, enabling increased data abstraction and modularization.

To evaluate Derric we have developed Excavator, a data analysis tool in the area of file carving making full use of Derric to describe data formats. Excavator is compared to popular existing file carvers used in practice on a test set consisting of standard carving benchmarks and challenges used in digital forensics research. Our comparison shows that Excavator is in the same league as the existing file carvers in terms of carving results and runtime performance, while requiring minimal effort to develop and allowing reuse of its data format specifications.

Directions for Future Work

In order to better validate our efforts to address the challenges in the areas of data abstraction and scalability, we intend to develop a test set that is large enough to allow evaluation of scalability and contain a large amount of files in different data formats that are representative of the domain, including image, movie, document and container formats.

Second, to evaluate whether our language is usable in multiple areas of digital forensics, we intend to develop different data analysis tools based on Derric, for instance to analyze network streams and memory layouts.

Finally, a user evaluation of DERRIC must be performed among actual users of the language, once a system using it is actually deployed and used in real-world digital forensics investigations.

8. REFERENCES

[1] G. Back. DataScript - A Specification and Scripting Language for Binary Data. In *Proceedings of the 1st ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE'02)*, volume 2487 of *LNCS*, pages 66–77. Springer, 2002.

[2] P. Baker, S. Loh, and F. Weil. Model-Driven Engineering in a Large Industrial Context—Motorola Case Study. In *Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems (MODELS'05)*, volume 3713 of *LNCS*, pages 476–491. Springer, 2005.

[3] L. Burgy, L. Reveillere, J. L. Lawall, and G. Muller. A Language-Based Approach for Improving the Robustness of Network Application Protocol Implementations. In *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS'07)*, pages 149–160, 2007.

[4] B. Carrier. Digital Forensics Tool Testing Images. `http://dftt.sourceforge.net/`.

[5] Centraal Bureau voor de Statistiek. *De digitale economie*. 2009. In Dutch.

[6] M. I. Cohen. Advanced Carving Techniques. *Digital Investigation*, 4(3-4):119–128, 2007.

[7] K. Czarnecki and S. Helsen. Feature-Based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3):621–646, 2006.

[8] K. Fisher, Y. Mandelbaum, and D. Walker. The Next 700 Data Description Languages. *Journal of the ACM*, 57(2):1–51, 2010.

[9] S. L. Garfinkel. Carving Contiguous and Fragmented Files with Fast Object Validation. *Digital Investigation*, 4(S1):2–12, 2007. Proceedings of the Seventh Annual DFRWS Conference.

[10] S. L. Garfinkel. Digital Forensics Research: The Next 10 Years. *Digital Investigation*, 7(S1):S64 – S73, 2010. Proceedings of the Tenth Annual DFRWS Conference.

[11] C. Grenier. PhotoRec, 2009. `http://www.cgsecurity.org/wiki/PhotoRec`.

[12] D. Grune and C. Jacobs. *Parsing Techniques—A Practical Guide*. Springer, 2008.

[13] F. Hermans, M. Pinzger, and A. van Deursen. Domain-Specific Languages in Practice: A User Study on the Success Factors. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09)*, volume 5795 of *LNCS*, pages 423–437. Springer, 2009.

[14] ITU/CCITT. Recommendation T.81 (JPEG Compression Specification), 1992.

[15] T. Jim, Y. Mandelbaum, and D. Walker. Semantics and Algorithms for Data-Dependent Grammars. In *Proceedings of the 37th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'10)*, pages 417–430. ACM, 2010.

[16] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Press, March 2008.

[17] P. Klint, T. van der Storm, and J. Vinju. Rascal: A Domain Specific Language for Source Code Analysis and Manipulation. In *Proceedings of the Ninth IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'09)*, pages 168–177. IEEE Computer Society, 2009.

[18] M. Komorowski. A History of Storage Cost, 2009. `http://www.mkomo.com/cost-per-gigabyte`.

[19] Y. Mandelbaum, K. Fisher, D. Walker, M. Fernandez, and A. Gleyzer. PADS/ML: A Functional Data Description Language. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'07)*, pages 77–83. ACM, 2007.

[20] P. J. McCann and S. Chandra. Packet Types: Abstract Specification of Network Protocol Messages. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'00)*, pages 321–333. ACM, 2000.

[21] M. Mernik, J. Heering, and A. M. Sloane. When and How to Develop Domain-Specific Languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

[22] J. Metz. ReviveIt 2007. `http://sourceforge.net/projects/revit/`.

[23] Microsoft. Microsoft Office File Formats, 2008. `http://msdn.microsoft.com/en-us/library/cc313118.aspx`.

[24] A. Pal and N. Memon. The Evolution of File Carving. *Signal Processing Magazine, IEEE*, 26(2):59–71, 2009.

[25] T. Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, 2007.

[26] G. G. Richard, III and V. Roussev. Scalpel: A Frugal, High Performance File Carver. In *Proceedings of the Fifth Annual DFRWS Conference*, 2005.

[27] D. C. Schmidt. Model-Driven Engineering. *Computer*, 39:25–31, 2006.

[28] D. Spinellis. Notable Design Patterns for Domain-Specific Languages. *Journal of Systems and Software*, 56(1):91–99, 2001.

[29] M. Staron. Adopting Model Driven Software Development in Industry—A Case Study at Two Companies. In *Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems (MODELS'06)*, volume 4199 of *LNCS*, pages 57–72. Springer, 2006.

[30] A. van Deursen, P. Klint, and J. Visser. Domain-specific languages: An annotated bibliography. *SIGPLAN Notices*, 35(6):26–36, 2000.

[31] E. Visser. *Syntax Definition for Language Prototyping*. PhD thesis, University of Amsterdam, 1997.

[32] W3C. Portable Network Graphics (PNG) Specification, 2003. `http://www.w3.org/TR/PNG/`.