Domain-Specific Languages for Digital Forensics

Jeroen van den Bos

Centrum Wiskunde & Informatica
Nederlands Forensisch Instituut
jeroen@infuse.org

Abstract. Due to strict deadlines, custom requirements for nearly every case and the scale of digital forensic investigations, forensic software needs to be extremely flexible. There is a clear separation between different types of knowledge in the domain, making domain-specific languages (DSLs) a possible solution for these applications. To determine their effectiveness, DSL-based systems must be implemented and compared to the original systems. Furthermore, existing systems must be migrated to these DSL-based systems to preserve the knowledge that has been encoded in them over the years. Finally, a cost analysis must be made to determine whether these DSL-based systems are a good investment.

1 Problem Description and Motivation

Digital forensic investigations are nearly exclusively performed using software, but high pressure in the form of strict deadlines combined with case-specific requirements severely complicates its use. The problems associated with constantly changing software are well-known [11]. This research attempts to address this problem by introducing and migrating to DSL-based systems.

The use of software in digital forensics is the result of constantly increasing storage device sizes (a gigabyte halves in price every fourteen months [10]), increasing connectivity (the amount of households having a broadband connection has more than doubled in the past five years [4]) and the pervasiveness of digital devices (currently there are more active mobile phones in The Netherlands than there are citizens [4]).

This explosion in connectivity and storage capacity has many advantages, but also drawbacks, one of them being the increased use of these capabilities by criminals. This in turn has caused both the amount and scale of digital forensic investigations to explode. Extensive automation appears to be the only feasible approach to deal with these increases, as manual inspection of even a single gigabyte for possible evidence could take years to complete.

Some things haven't changed however, including legal requirements around (pre-charge) detainment of suspects. This means that forensic investigations nearly always have very strict deadlines. Additionally, the variety of devices, applications and communication channels ensures that digital forensic investigations typically require custom tools.

Application-specific knowledge, in the form of communication protocols, storage device layouts and embedded systems implementation details are often case-specific while the software used to recover data are tools implementing general algorithms.

The algorithms rarely have to change (although new ones regularly emerge), but the specifications they are working with constantly change.

Solutions exist that employ methods of software reuse and abstraction to reduce modification time, but in practice it turns out that these still require software engineers to actually make the changes. In a situation where a large amount of changes to software must be made within a couple of days, the process of transferring knowledge to software engineers who then make the actual changes is time-consuming, error-prone and hard to trace.

2 Brief Overview of Related Work

Extensible digital forensic applications exist, such as TULP2G [2] for analyzing embedded devices and ReviveIt [14] and Scalpel [16] for recovering lost and hidden files from storage devices. They all have similar limitations however in that they tangle implementation of application-specific knowledge (e.g. file formats and memory layouts) and recovery algorithms. In TULP2G, the application-specific knowledge in the form of communication protocols must be defined in an imperative language along with the recovery algorithms. Both ReviveIt and Scalpel use an external notation to specify application-specific knowledge in the form of file format specifications, but their notation is heavily based on the actual recovery algorithms used, making it difficult to develop new algorithms without changing the search patterns.

There is a lot of research in interpreting structured data. Parsing techniques [7] using grammar formalisms such as ANTLR [15] and SDF2 [18] are targeted at textual programming languages however and lack features to support complex data dependencies between elements in a protocol, file format or memory structure. Data-dependent grammars extend traditional parsing technology to allow the definition of such dependencies [8]. These grammars are used to derive parsers for Data Description Languages (DDLs) [6]. PADS/ML [12], DataScript [1] and Zebu [3] are all examples of a such DDLs. These DDLs are typically positioned as productivity-enhancing tools for programmers, making them less suitable for use in forensic investigations where users are often reverse engineering data structures instead of developing them from scratch.

There is extensive work in the area of developing DSLs [5] [17] [13] [9], however some open questions remain such as how to use knowledge from existing systems in their design.

3 Proposed Solution

Our proposed solution is to develop one or more DSLs to meet the challenges posed by digital forensics. Model-driven engineering in general and DSLs in particular may be a good fit for digital forensic software.

3.1 General Approach

An analysis of both literature and practice shows that digital forensic investigations often have multiple concerns that may be valuable in applying a model-driven approach:

Application-specific knowledge Information that is application-specific refers to knowledge about devices, formats and protocols that are specific to a certain brand, type, standard or version of something. For example, when analyzing communication streams, this includes the protocols and compression and encryption methods. When analyzing a digital storage device (e.g., a hard drive), this includes the disk layout, file systems and file formats. When analyzing an embedded system (e.g., a mobile phone), this includes the memory layout.

Recovery algorithms Methods that belong to this category are techniques that take application-specific knowledge and use algorithms to recover data from the actual stream, storage or embedded device. They are often specific for a certain area of digital forensics, such as reassembly and identification algorithms for recovering deleted or hidden files from a confiscated hard drive. Although specific for a certain area, they are independent of specific changes to a format or version.

The proposed solution to increase usability and modifiability is to create one or several DSLs for forensic investigators. This will allow them to express application-specific knowledge that they obtain during their investigations in a language that is appropriate to them and abstract away all implementation details such as recovery algorithms. These in turn will be implemented in general-purpose languages and be maintained by software engineers.

However, to be able to do this several research questions must be addressed, such as:

1. How can qualities such as flexibility and adaptability be compared between an existing system and a DSL-based system?
2. How can the difference in development and maintenance costs between an existing system and a DSL-based system be measured?
3. How should the form of a DSL be determined?
4. How can the knowledge encoded in an existing system be used in the design of a DSL?
5. How can a DSL be developed so that it can be maintained by general developers?

3.2 Example Application

An example application of the general approach discussed is a file carver. File carving is the process of recovering deleted or damaged files from data storage devices (e.g. hard drives). The implementation of file systems allows deleted files to often be recoverable, although the contents of these files may be spread out across different sections of a device. To undo this so-called fragmentation, file carving algorithms use heuristics, operating system implementation details and file format recognizers to attempt to recover complete files.

If these recognizers are implemented using a general-purpose language, their method of recognizing file types is typically hard-coded in each instance. Furthermore, they may become tangled with the implementation of the recovery algorithms. A result is that changing a recovery algorithm or adding a new file format requires a significant software engineering effort.

However, using a data description language to define file formats along with a code generator to transform these declarative descriptions into recognizer implementations may reduce the required effort significantly. Furthermore, separating definition of file formats from the implementation of recovery algorithms may make it easier to modify or add new algorithms afterwards. Finally, a code generator may perform additional optimizations that would be difficult to perform manually (such as take several file format definitions and generate a single recognizer, improving scalability by reducing reads).

4 Research Method

To validate our hypothesis and answer the research questions, the following steps will be undertaken:

DSL Development Experiments will be performed by implementing one or several DSLs in the digital forensic domain. This will require experiments in the area of domain analysis as well.

DSL Validation The implemented DSL-based systems will be validated by comparing them in general use to existing digital forensic software, to determine whether the approach is viable in areas such as functionality, runtime performance and flexibility.

Automated Analysis Several techniques will be employed to aid DSL development:

- Automated model extraction to assist DSL design.
- Automated comparison between the existing systems and the DSL-based systems, to find implementation differences and preserve knowledge.

5 Conclusion

To keep up with the size of storage devices, speeds of network connections and amount of digital devices in use, digital forensic investigations rely heavily on custom software applications to perform large parts of analyses. However, the continuous introduction of new consumer applications and devices requires forensic software to be exceptionally flexible and adaptable.

To realize these requirements, using domain-specific languages to raise the level of abstraction and separate different concerns in the domain may be a viable approach. However, this requires analysis, design and implementation of systems employing these techniques as well as evaluation to compare existing systems to these alternative solutions.

This research will perform these steps by implementing one or several DSL-based systems, comparing them to existing systems to determine their relative performance and employ automated analysis techniques to aid in the design and preserve knowledge.

References

1. Back, G.: 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). LNCS, vol. 2487, pp. 66–77. Springer (2002)
2. van den Bos, J., van der Knijff, R.: An Open Source Forensic Software Framework for Acquiring and Decoding Data Stored in Electronic Devices. International Journal of Digital Evidence 4(2) (2005)
3. Burgy, L., Reveillere, L., Lawall, J.L., Muller, G.: 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). pp. 149–160 (2007)
4. Centraal Bureau voor de Statistiek: De Digitale Economie. CBS (2009)
5. van Deursen, A., Klint, P., Visser, J.: Domain-Specific Languages: An Annotated Bibliography. SIGPLAN Notices 35(6), 26–36 (2000)
6. Fisher, K., Mandelbaum, Y., Walker, D.: The Next 700 Data Description Languages. Journal of the ACM 57(2), 1–51 (2010)
7. Grune, D., Jacobs, C.: Parsing Techniques—A Practical Guide. Springer (2008)
8. Jim, T., Mandelbaum, Y., Walker, D.: Semantics and Algorithms for Data-Dependent Grammars. In: Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'10). pp. 417–430. ACM (2010)
9. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press (2008)
10. Komorowski, M.: A History of Storage Cost (2009), `http://www.mkomo.com/cost-per-gigabyte`
11. Lehman, M.: Programs, life cycles, and laws of software evolution. Proceedings of the IEEE 68(9), 1060 – 1076 (1980)
12. Mandelbaum, Y., Fisher, K., Walker, D., Fernandez, M., Gleyzer, A.: PADS/ML: A Functional Data Description Language. In: Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'07). pp. 77–83. ACM (2007)
13. Mernik, M., Heering, J., Sloane, A.M.: When and How to Develop Domain-Specific Languages. ACM Comput. Surv. 37(4), 316–344 (2005)
14. Metz, J.: ReviveIt 2007, `http://sourceforge.net/projects/revit/`
15. Parr, T.: The Definitive ANTLR Reference: Building Domain-Specific Languages. Pragmatic Bookshelf (2007)
16. Richard, III, G.G., Roussev, V.: Scalpel: A Frugal, High Performance File Carver. In: Refereed Proceedings of the 5th Annual Digital Forensic Research Workshop (DFRWS'05) (2005)
17. Spinellis, D.: Notable design patterns for domain-specific languages. Journal of Systems and Software 56(1), 91–99 (2001)
18. Visser, E.: Syntax Definition for Language Prototyping. Ph.D. thesis, University of Amsterdam (1997)