Knitting Music and Programming

Reflections on the Frontiers of Source Code Analysis

Nicolas Gold
CREST, Department of Computer Science
University College London
London, UK
n.gold@ucl.ac.uk

***Abstract*— Source Code Analysis and Manipulation (SCAM) underpins virtually every operational software system. Despite the impact and ubiquity of SCAM principles and techniques in software engineering, there are still frontiers to be explored. Looking "inward" to existing techniques, one finds frontiers of performance, efficiency, accuracy, and usability; looking "outward" one finds new languages, new problems, and thus new approaches. This paper presents a reflective framework for characterizing source languages and domains. It draws on current research projects in music program analysis, musical score processing, and machine knitting to identify new frontiers for SCAM. The paper also identifies opportunities for SCAM to inspire, and be inspired by, problems and techniques in other domains.**

Keywords: Source code analysis; music programming; music analysis; graphical programming

I. INTRODUCTION

Source-code analysis and manipulation (SCAM) is fundamental to the creation and maintenance of software systems, and SCAM techniques are found in most parts of the software engineering lifecycle. From supporting program creation in an IDE, through compilation and deployment, to support for understanding and bug fixing in maintenance, source code analysis technologies are found throughout. There continue to be many challenges in creating and improving SCAM techniques for general purpose programming languages, and in finding and addressing challenges posed by new languages and domains. Both areas represent frontiers of SCAM.

This paper contributes to the development of the research agenda on the frontier of new domains and languages. Three current research projects (Section II) in the areas of computer music and machine knitting are used as concrete starting points for a reflective analysis of the types of language or representation that might be amenable to SCAM. The analysis (Section III) also considers aspects of problem domains (focusing particularly on music as an example) and the implications of these for SCAM. The result is a framework to stimulate and position future research. Possible avenues for such research, and ideas for cross-fertilization of principles and methods between SCAM and other domains, are also presented (Section IV).

II. CURRENT PROJECTS

A. *Project 1: Clone Detection in Max/MSP*

Project 1 is developing clone detection methods for music/art programming languages such as Max/MSP. Max/MSP [9] is a graphical data-flow language widely used for installation art and interactive music systems. Programs (termed patches) are constructed by laying out functional boxes and connecting them with lines to define the dataflow between them. An example is shown in Fig.1. The language is similar to Simulink [20] in programming paradigm but in Max/MSP the semantics depend not just on the connections between objects but on the way patches are spatially arranged.

A clone detection method for Max/MSP patches has been developed and evaluated [14], and clones of various types have been found. The work is being extended to analyze PureData [23], a language similar to Max/MSP but where the semantics depend partly on the creation sequence of patch lines instead of the spatial arrangement of graphical boxes.

In addition to spatial-layout and creation-order issues, techniques that tackle these languages need to consider the broader context e.g. Max/MSP supports reuse and modularity by nesting patches but programs written in Max/MSP typically are not large in comparison to those written in general purpose languages. Thus, single program

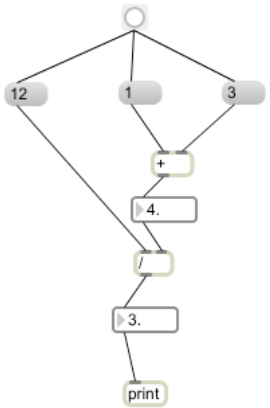

Figure 1. Example of a Max/MSP Patch

analysis may not demand high-efficiency techniques for success. Patch libraries may be large and thus techniques for efficiently analyzing many small files might be needed. In addition, it is common for programmers to develop *externals* (separate programs in C or C++ that appear as boxes in Max/MSP patches) so analysis tools may need to operate across multiple languages simultaneously.

B. Project 2: Turning Musical Scores into Music Systems

This project is researching the engineering of systems for live ensemble Human-Computer Music Performance (HCMP) of "popular" music (the specific genres being considered are jazz and contemporary church music). HCMP involves computers playing independently alongside human musicians [10]. A key issue for this research is finding an appropriate representation for musical scores. Music representation has been an open research issue for many years (e.g. see Selfridge-Field [25]) because of the multiplicity of uses for such representations. Popular music scores are primarily structural i.e. the individual notes do not generally need encoding since much of the music is improvised. Works are organized by sections like *verse* or *chorus*. This resembles block-structuring in programming languages and possesses the same advantages of modularity and flexibility.

Since no appropriate existing encoding was found, the initial work of this project [13] defined a score representation to encode three key aspects: *static score, arrangement,* and *dynamic score*. The static score represents the music as it is written on the page, the arrangement is the ordering of sections and repeat structures as defined by the performers, and the dynamic score is a static score "unfolded" according to the arrangement to enable media synchronization and playback. The language reported in [13] is satisfactory for well-formed scores, however in popular music it is not always the case that scores are well-formed and a more advanced approach is being developed to transform the encoding language into Extended Hierarchical Finite State Machines. This will allow analysis by techniques such as model-checking [8] and model-projection [2] to determine the validity of the arrangement, and will also allow the integration of functional architectural components such as cueing systems for score rearrangement at performance-time. The language needs to be simple and domain-oriented so that users with little or no programming experience can simply re-encode the musical notation on a manuscript page.

C. Project 3: Knitting with Machines

Project 3 is in the very early stages of exploring, through collaboration with an independent fashion designer, the extent to which knitting designs for seamless garments can be treated as source code and subjected to various source code transformations, e.g. to save yarn. Machines capable of knitting rapidly to bespoke designs now routinely produce knitted garments. Human software engineers program them, yet at this stage of the research it appears that this process is largely (if not entirely) a deterministic translation from a graphical design with annotations (e.g. for stitch density at various points) to a set of program statements. The "source code" is thus a mixture of graphical and textual representations arising naturally from the design process.

III. Reflections

A. Problem Characteristics

All problems addressed by SCAM have their roots in domain questions. For general-purpose languages (GPL), these questions are typically drawn from the problem domain of software engineering, e.g. in program slicing, both problem (reducing program size) and solution (slicing) exist in the same domain. In identifying problems amenable to SCAM, the first step is to consider one or more domain questions, e.g. why are certain performers' musical interpretations preferred to others, or how can partially-improvised music scores be represented?

Secondly, the availability of domain representations that can be used as source code must be established. This will require finding existing representations that embody principles typically necessary for execution, e.g. dependence, sequence, conditions, and branching. If such a representation does not exist, it may be appropriate to consider introducing one, although the ability of users to adopt explicit programming notions in their work must be considered if programming is required.

Thirdly, the nature of "successful" execution in the problem domain must be considered. Drawing on Harman's concept of "tendency to executability" [15], it is likely that many domain representations can be executed in the traditional computing sense. However, the degree to which that execution actually meets a domain-appropriate understanding of execution may vary, e.g. one can automatically "execute" a full musical score using programs that increasingly embody good musical performance practice (see Hiraga et al. [16]). For HCMP systems like Project 2, the music is partly improvised so there is no deterministic execution path when successful performance (execution) is considered from the perspective of the problem domain (performing good music). Successful execution may be seen either as deterministic movement through the ordered score sections (deterministic analyses are thus deemed to completely capture executability in the domain) or as the production of aesthetically pleasing music (executability must account for performer interpretation). New, non-deterministic concepts of execution and analysis may be needed to incorporate uncertainty and interpretation, e.g. using probabilistic formulations of dependence [3] or modeling human input.

Some domain problems and/or representations may not lend themselves to execution at all (e.g. finding clusters in performance timings [26] or audio analysis for musical features using tools like Sonic Visualiser [5]) and may be inappropriate for SCAM approaches.

The separation between formal and interpretive semantics of execution finds a natural parallel in music. A musical score is an analyzable formal representation that typically constrains a performer to an ordered set of notes. However, the music performed (i.e. executed) relies on the performer's interpretation (performers are not mere

reproducers of the composer's score but bring their own creativity to each performance [6]). It is therefore possible to perform a score deterministically using a system of performance rules, or interpretively (applying performance norms but allowing room for creative expression). Similar ideas can be seen in E-Type systems [7] where a software system installed in its environment deterministically models and executes part of the overall social-technical system. Users who work around software limitations might be seen as bringing an interpretive execution to the system as a whole. In terms of software evolution, interpretive execution of the socio-technical process might be seen as driving new requirements to minimize the non-deterministic aspects.

In summary, the applicability of a SCAM approach in a new problem domain depends on at least the following: the questions, concepts, and methods of users; the domain representations that can be used as source code; how far these tend to deterministic executability and whether this captures a sufficiently rich domain goal; the potential benefits for the problem at hand; and opportunities for new domain insights.

B. *Language Characteristics*

The projects' representations may be seen as increasingly "domain-embedded" source code languages. Project 1 is dealing with a language that is (from the perspective of general-purpose languages) unusual in programming paradigm, syntactic presentation, and execution semantics. It may be characterized as a domain-specific programming language (DSL) since its overt intention is to permit the writing of programs, with syntax and constructs designed to support a particular domain [11].

Project 2 is defining, transforming and analyzing a language for representing a particular aspect of musical scores. The method implicitly allows the user to define their own programming language through the creation and ordering of static score sections and the placing of simple conditions on their execution, although a typical user is unlikely to perceive this as programming (indeed, the intention is that they do not). The static score encoding is not intended as a programming language at all. It could be termed an *Artificial Domain-Embedded Language (ADEL)*, one that is not concerned overtly with programming but instead with representing domain concepts.

In the case of Project 3, the "language" does not arise from computing concerns at all, but naturally from domain practice. Since the designer's domain of discourse is primarily graphical, the intervention of programmers is currently required to produce programs for knitting machines, but since there is an apparently deterministic relationship between the pattern and the garment these designs might be treated as source code. This might be termed a *Natural Domain-Embedded Language (NDEL)*.

These reflections suggest a spectrum of language types from general-purpose, through domain-specific and embedded languages to non-executable domain representations (NEDR) like audio. As the language type moves from GPL towards NDEL, it is likely that end-user programming becomes the default paradigm, although not necessarily in the traditionally-studied domain of office systems [4]. Table 1 summarizes these reflections, showing characteristics in the rows, and types of language in the columns. It might be argued that the outward-facing frontiers of SCAM lie in working with languages and domains characterized by the middle three columns.

IV. New Opportunities

This section presents two groups of ideas. The first consists of opportunities for SCAM to be fruitfully applied to new problems and domains; the second consists of those from which SCAM might draw inspiration for novel techniques.

TABLE I. Table of Language and Domain Characteristics

	GPL	DSL	ADEL	NDEL	NEDR
Language Source	Language Designer	Language Designer	Language Designer	Domain User	Domain User
Language Intent	Programming	Programming/Domain Representation	Domain Representation	Domain Representation	Domain Representation
Concepts Encoded	Programming	Programming/Domain	Domain	Domain	Domain
Typical User	Programmer	Programmer	Domain User	Domain User	Domain User
Apparent Executability	High	High	Medium	Low	None
Tendency to Executability	High	High	Medium	Medium/Low	None
Likely Execution Determinism	Total	Total	Partial	Partial	None

A. Possible Domains for SCAM Solutions

Processing [24] is a Java-based DSL for arts computing. It allows a variety of programming styles and has its own IDE that supports writing and managing sketches (Processing's file collection scheme) but few analytical capabilities, e.g. slicing. Users may wish to undertake such transformations but the nature of the source code could pose challenges. Processing sketches can be quite unstructured and thus require multi-level analysis through the underlying Java code to compute a slice.

More advanced program analysis may require new slice criteria such as a temporal dimension expressed in terms of frames of interest or time elapsed in addition to position and variables. For example, a Processing sketch was written by the author for a collaborative project [1] to digitally capture and replay musicians' shape responses to musical stimuli using a graphics tablet. The sketch is structured using the standard Processing `setup()` and `draw()` routines that are executed once, and once per frame, respectively. During development, various bugs were found in the interaction between capture and replay frames, and also in the temporal sequencing of stimuli. It would have been extremely helpful to be able to slice on lines of code, but only for particular frames, or framesets. In this case, debugging was aided by explicitly controlling frame-rate and frame-type but this is not the most desirable solution.

Another potentially rich domain is Live Coding. This is a contemporary musical (and sometimes video) performance practice involving programming live on-stage, typically using textual interpreted synthesis languages such as SuperCollider [27] and with the source code projected in real-time for an audience [4]. SCAM might contribute artistically e.g. to live coding visualizations such as those discussed by McLean et al. [19] and/or through the real-time derivation and display of source code properties (e.g. cohesion and coupling or other metrics). Blackwell and Collins' discussion of programming languages in this context points to possible avenues for SCAM contributions [4]. Their observation that program behavior in live coding may not be predictable but may vary according to human and aesthetic performance dynamics lends weight to the argument for new non-deterministic models of execution and analysis. They claim that music notations do not need to support arbitrary restructuring in the same way as languages like UML. However, the experience of Project 2 [13] suggests that in some cases there is such a need and that there is untapped potential for transformation tools and techniques. They also identify maintenance and comprehension issues for this domain, at which SCAM techniques may be targeted [4].

Dependence underpins many SCAM techniques but does not easily translate to the musical domain. However, if such a mapping can be conceived, there may be opportunities to develop novel musicological analyses, for example, in automated music summarization similar to Marsden's approaches toward automating Schenkerian analysis [18], but based on alternative views of deep music structure. It is likely that any notion of dependence in music will need to be interpretive and require probabilistic or nuanced descriptions.

B. Possible Solutions for the SCAM Domain

The way music is laid out on a page has a significant influence on its comprehensibility and usability. For example, expert musicians use pattern matching and chunking when reading music and deduce note duration from note spacing [28]. It is thus important that the spacing of the music reflects the character of the music [22]. Since expert programmers use plan recognition and chunking in comprehension [21], perhaps source code layout could be inspired by the principles of music engraving and reflect something of the execution characteristics of a program in its layout. Similar ideas for making programs readable have been considered before (e.g. Knuth's literate programming [17]) but by treating a program almost as an artwork to be mentally executed (like a musical score) and laying it out accordingly, new and fruitful directions for program comprehension might be identified.

A source code file could be seen as a palimpsest (a document used and then overwritten one or more times). Documentary palimpsests are analysed using various chemical, physical or optical techniques (e.g. multi-spectral imaging [12]) to reveal the overwritten texts but the notion has been applied in other non-physical domains (e.g. cinema [29]). Software repositories offer an easy way to access the previous versions of source code files but perhaps methods for understanding evolution histories might be enhanced by adopting palimpsest-oriented cultural enquiry methods from other domains e.g. in the investigation of major software system failures and the cultures and circumstances surrounding their creation.

Further inspirations may be found in other types of musical analysis. As discussed above, dependence is not a concept that easily translates to music yet if it could be, this may offer new insights into program dependence. Other parallels might be found between programming patterns and motivic structures in music, with techniques for analysing musical motives offering new ways to approach program structure. Finally, the principles of automated Schenkerian analysis [18] for music summarization might be applied to program summarization to support comprehension and other tasks.

V. Conclusion

The impact of SCAM is beyond doubt, yet there continue to be exciting frontiers to explore in developing new and improved techniques for existing domains of application, and in the identification of new source codes and domains for analysis. The reflective framework developed in this paper identifies frontiers in the analysis and manipulation of domain-specific source codes. These can arise through domain-specific programming language design, through the introduction of representations for domain concepts, or by reusing naturally occurring domain representations. In developing the framework, the paper has drawn on three specific projects to consider the nature of their languages and domains, and how these might be addressed by SCAM. It

has also shown where SCAM might benefit from the cross-fertilization of ideas and techniques.

ACKNOWLEDGMENTS

I am very grateful to the many colleagues and collaborators with whom I have worked over the last few years for stimulating discussions and support of my interdisciplinary research, in particular David Binkley, Indu Choraria, David Clark, Roger Dannenberg, Mark Harman, Jens Krinke, Daniel Leech-Wilkinson, John Rink, and Neta Spiro. This work is supported by the UK Engineering and Physical Sciences Research Council [grant numbers EP/F059442/2 and EP/G060525/2].

REFERENCES

[1] AHRC Research Centre for Musical Performance as Creative Practice, "Shaping music in performance," http://www.cmpcp.ac.uk/smip.html

[2] K. Androutsopoulos, D. Binkley, D. Clark, N.E. Gold, M. Harman, K. Lano, and Z.Li, "Model Projection: Simplifying models in response to restricting the environment," Proc. 33rd Int. Conf. on Software Engineering (ICSE 2011), 21-28 May 2011, Waikiki, Honolulu, Hawaii.

[3] G.K. Baah, A. Podgurski, M.J. Harrold, "The probabilistic program dependence graph and its application to fault diagnosis,", Proc. Int. Symp. on Software Testing and Analysis 2008, Seattle, WA, July 2008.

[4] A. Blackwell and N. Collins, "The programming language as a musical instrument," Proc. 17th Psychology of Programming Interest Group (PPIG) Workshop, Brighton, UK, 29 June-1 July 2005, pp. 120-130.

[5] C. Cannam, C. Landone, and M. Sandler, "Sonic Visualiser: An open source application for viewing, analysing, and annotating music audio files," Proc. ACM Multimedia 2010 Int. Conf., Firenze, Italy, Oct 2010.

[6] N. Cook, Music: A Very Short Introduction, Oxford Univ. Press, 1998.

[7] S. Cook, R. Harrison, M.M. Lehman, P. Wernick, "Evolution in software systems: foundations of the SPE classification scheme," J. Software Maintenance and Evolution: Research and Practice, vol. 18, 2006 pp. 1-35.

[8] J.C. Corbett, M.B. Dwyer, J. Hatcliff, S. Laubach, C.S. Pasareanu, Robby, and H. Zheng, "Bandera: Extracting finite-state models from Java source code," Proc. 22nd Int. Conf. on Software Engineering, Limerick, Ireland, IEEE Computer Society Press, pp. 439–448, 2000.

[9] Cycling74, Max/MSP, http://www.cycling74.com

[10] R.B. Dannenberg, "A vision of creative computation in music performance," Proc. 2nd Int. Conf. on Computational Creativity, Mexico City, April 2011.

[11] M. Fowler, Domain-Specific Languages, Addison-Wesley, 2011.

[12] M. Gau, H. Miklas, M. Lettner, and R. Sablatnig, "Image acquisition and processing routines for damaged manuscripts," Digital Medievalist, vol. 6, 2010.

[13] N.E. Gold and R.B. Dannenberg, "A reference architecture and score representation for popular music human-computer music performance systems," Proc. 11th Int. Conf. on New Interfaces for Musical Expression (NIME2011), 30 May 2011-1 June 2011, Oslo, Norway.

[14] N.E. Gold, J. Krinke, M. Harman, and D. Binkley, "Cloning in Max/MSP patches," Proc. 37th Int. Computer Music Conf. (ICMC), Huddersfield, UK, 31 July- 5 Aug, 2011, in press.

[15] M. Harman, "Why source code analysis and manipulation will always be important," Proc. 10th IEEE Int. Working Conf. on Source Code Analysis and Manipulation, Timişoara, Romania, 12-13 Sept 2010.

[16] R. Hiraga, R. Bresin, K. Hirata, and H. Katayose, "Rencon 2004: Turing test for musical expression," Proc. Int. Conf. on New Interfaces for Musical Expression (NIME04), Hamamatsu, Japan, June 2004.

[17] D. Knuth, "Literate Programming," The Computer Journal, vol. 24, no. 2, 1984, pp. 97-111.

[18] A. Marsden, "Schenkerian analysis by computer: A proof of concept," Journal of New Music Research, vol. 39, no. 3, in press.

[19] A. McLean, D. Griffiths, N. Collins, and G. Wiggins, "Visualisation of live code," Proc. Electronic Visualisation and the Arts (EVA) – London 2010, London, UK, July 2010.

[20] Mathworks, Simulink, http://www.mathworks.co.uk/products/simulink/

[21] A. Von Mayrhauser and A.M. Vans, "Program comprehension during software maintenance and evolution," IEEE Computer, vol. 28, no. 8, August 1995, pp. 44-55.

[22] H.-W. Nienhuys and J. Nieuwenhuizen, "LilyPond, a system for automated music engraving," Proc. XIV Colloquium on Musical Informatics (XIV CIM 2003), Firenze, Italy, May 2003.

[23] PureData, http://puredata.info

[24] Processing, http://processing.org

[25] E. Selfridge-Field (ed.), Beyond MIDI: The Handbook of Musical Codes, MIT Press, 1997.

[26] N. Spiro, N.E. Gold, and J. Rink, "The Form of Performance: Analyzing Pattern Distribution in Select Recordings of Chopin's Mazurka Op. 24 No. 2," Musicae Scientiae, vol. 14, 2010, pp 23–55.

[27] SuperCollider, http://supercollider.sourceforge.net

[28] A.J. Waters, G. Underwood, and J.M. Findlay, "Studying expertise in music reading: Use of a pattern-matching paradigm," Attention, Perception and Psychophysics, vol. 59, no. 4, pp 477-488. 1997.

[29] D.G. Young, "Hamlet" in the Cinema: A Palimpsest of Performance, Ph.D. Thesis, The University of Oklahoma, 2007.