

ThisStar: Declarative Visualization Prototype

Joseph A. Cottam*

Computer Science Department, Indiana University

Andrew Lumsdaine†

Computer Science Department, Indiana University

ABSTRACT

Library-based and pre-compiled visualization tools incur many penalties that hinder the adoption of visualization as a technique for many fields. Libraries necessitate familiarity with the data structures and control flows that are incumbent in traditional programming, but not central to visualization. Task-specific visualization applications alleviate these needs, but induce users to move data between applications as their needs change. An ever expanding tool chain and corresponding context switches are inefficient. We propose a generative programming approach to visualization tool construction based on domain specific languages. Through it, we provide the flexibility of a general purpose programming language that abstracts out many of the control flow and data structure issues. Our initial prototype, ThisStar, creates star-maps based on these general ideas. ThisStar demonstrates the viability of these concepts and illuminates opportunities.

Keywords: Star Map, Visualization, Domain Specific Language, Declarative Language, Generative Programming.

Index Terms: I.3.4 [Computer Graphics:]: Graphics Utilities—Software support I.2.2 [Computing Methodologies:]: Automatic Programming—Program synthesis D.3.2 [Programming Languages:]: Language Classifications—Specialized application languages H.1.2 [Information Systems:]: User/Machine Systems—Human information processing

1 INTRODUCTION

Information Visualization has encountered many roadblocks to its adoption beyond academic research or very isolated applications. Common practice follows one of two routes: 1) Employ or train an expert to handle the visualization task; 2) Purchase off-the-shelf software. These two methods incur penalties that limit their utility beyond research applications or very specific problems as they both imply high up-front costs (e.g. employing experts or evaluating/purchasing off-the-shelf applications) and high maintenance costs (e.g. retaining experts or a lengthening tool chain as more task-specific applications are purchased). In short, there is no way for visualization problems to ‘played with’ without either committing substantial resources or limiting their scope in early stages.

We believe that a declarative programming language that holds close to the core concepts of information visualization will ease the adoption and, combined with a generative programming style, will ease maintenance difficulties. The need for flexible, user extensible visualization has been recognized before in [5], but our system goes a step further by allowing the visualizations themselves to be specified, not just the coordination between them. The need to improve the programming of visualizations was recognized by [3], but the Processing system retains a traditional language structure while simplifying other parts of the process. To expand beyond this prior work, we describe the embodiment of our concepts in an abstract

*e-mail: jcottam@cs.indiana.edu

†e-mail: lums@cs.indiana.edu

Tuple-Space Mapper (TSM) and a preliminary implementation of them in ThisStar, an for the creation of star maps.

2 TUPLE-SPACE MAPPER

The Tuple-Space Mapper aims to reduce the problems associated with adoption of visualization. There are three central concepts in the TSM. First, most visualization is mapping entity features to visual attributes. Second, a domain specific language (DSL) provides a better conceptual fit than a general-purpose language but need not be less flexible. Third, modification of the visualization should be accessible for rapid prototyping and extension. (A more complete description of the TSM can be found in [1].)

The first concept leads to the formulation of tuple streams. Tuples are an elemental data representation that can be used to express many data types, and arbitrary attributes on them. By using streams of tuples, we alleviate the programmer of the burden of understanding the specifics of the data storage (a requirement and major obstacle when using contemporary libraries directly in a traditional language). This lets the programmer focus on the core operation of mapping the incoming tuple characteristics to visual characteristics on corresponding glyphs. This presents a challenge for the TSM system as it must infer proper data storage from a minimal description of the input data. This is not simple, but aided by the restriction that all programs generated by the TSM are visualization focused.

To expose these conceptual abstractions, we employ a DSL. This allows the tuples, mapping actions and selection actions to be directly expressed a straightforward manner. To provide the full flexibility of a general purpose language, we provide a mechanism to execute arbitrary code from the host application. Our implementation is patterned after the ad-hoc syntax-directed transformations of compiler compilers (such as YACC and ANTLR). This mechanism should remain constant as we migrate the TSM application to different host environments (regardless of the original host language). It also resembles the ‘advanced coordination’ option of the declarative coordination language presented in [5]. We employ a declarative language style as it generally incurs a low conceptual overhead and (combined with the simple data model) makes it easy to ‘play’ with a visualization specification. Since the TSM language implies no control structures, these must be inferred through analysis in the TSM Generator. Since data structures are also generated by the TSM system, this is simplified but remains non-trivial.

The TSM system does not generate complete visualization applications; instead it is a lightweight tool that generates components used in larger applications. These components can be based on any existing visualization library (see Figure 1). It is the responsibility of the TSM generator to convert the abstract mapping actions into concrete data structures and control flows. The resulting components conform to a straightforward interface, allowing the components generated from successive iterations of a specification to be interchanged (provided the characteristics of the incoming tuple streams remain congruent). The accessibility of the language permits non-experts to modify the visualization and the common interface allows new revisions to be simply incorporated into existing TSM software. However, since the TSM generator is only used when programming a visualization, it is not part of the tool chain during analysis operations. Presenting a consistent interface despite a variety of inputs and potential outputs is another major implementation challenge for the TSM system.

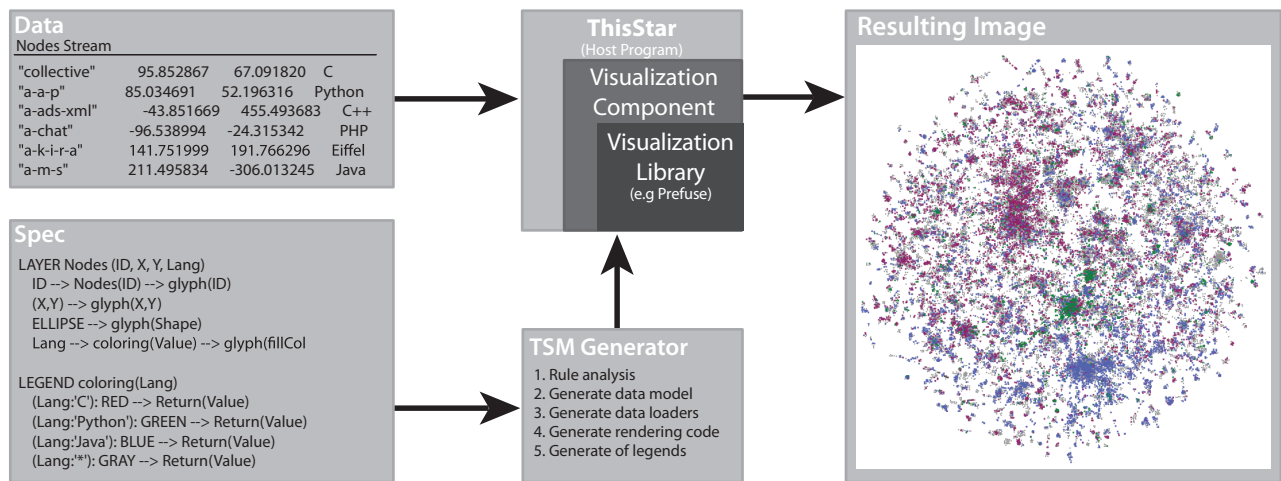


Figure 1: General architecture of a TSM application.

The concepts do three things to avoid the problems encountered in contemporary methods. First, they present a low barrier to entry by keeping visualization specific concepts in the foreground. Second, they provide a high degree of customizability. This allows applications built on the TSM to be extended, reducing the need to introduce new applications into a tool chain (the TSM generator is either orthogonal to the analysis tool chain or incorporated into an existing application). Finally, the simple data model facilitates chaining TSM-derived applications to existing tools when required by having a very simple, but general data structure.

3 THISSTAR

ThisStar is an application to generate star maps. It implements several of the ideas presented above by providing: A simple DSL/generation system encapsulating the core concepts described above and a host application. The DSL engine configures a Prefuse [4] back-end to visualize data contained in delimited text files. Our selection of Prefuse means the TSM applications can access libraries written in Java ad-hoc fashion (e.g. for analysis). Specifications, given as a whole, are used to configure a run-time visualization system (no program restart is required, though this is technically a compile step it is most similar to dynamic compilation [2]). The host-application portion of the ThisStar system creates the tuple streams from files and coordinates the replacement of the visualization as the specification is changed.

4 RESULTS

The initial system was used to visualize the interactions of open-source software development projects. The TSM language subset implemented in ThisStar was simple enough that small permutations to rules could be automatically generated and fed into the program. This automated the process of visualizing many similar but distinct aspects of our data. Results from the VxOrd layout application were easily presented as tuples to ThisStar and integrated with information from a database. Images from all rule permutations were then reviewed by hand for features of interest. Final composite images were generated in ThisStar using hand-modified rules to accent interesting points.

ThisStar is a success for several reasons. First, we were able to automatically generate the visualization rules required to show the data set (enabled by the simplicity of the DSL). Second, we were able to easily blend data from several sources (enabled by the simple data model). Finally, we were able to modify the initial rules

in a straight-forward, interactive fashion to produce our eventual visualizations (enabled by the declarative DSL and the ability to iteratively the declaration).

5 FUTURE WORKS

The results of ThisStar were generally successful, but highlighted areas for improvement. It is clear that better facilities for interaction with the resulting visualization are necessary. This will be part of our next iteration, representing the mouse and keyboard as tuple streams. Another avenue for improvement is the inclusion of reflexive constructs to the TSM language to allow visualizations to be self-referential in declaring mappings and conditions. These improvements will expand the range of applications that can be generated from a TSM-derived language. To demonstrate that the TSM concepts expand beyond Prefuse, we are currently working on generators to target Piccolo and Processing. We eventually plan to target non-Java libraries.

6 CONCLUSIONS

Our results from ThisStar are encouraging. Declarative languages have been successful in broadening the adoption of technologies in the past, and the success of ThisStar makes us believe that the same may apply for visualization in the future.

ACKNOWLEDGEMENTS

This work was supported in part by a grant from the Lilly Endowment.

REFERENCES

- [1] J. A. Cottam. Tuple space mapper: Design, challenges and goals. Technical Report TR648, Indiana University, Bloomington, IN, June 2007.
- [2] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools and Applications*. ACM Press/Addison-Wesley Publishing Co, New York, NY, 2000.
- [3] B. Fry. *Computational Information Design*. PhD thesis, Massachusetts Institute of Technology, 2005.
- [4] J. Heer, S. K. Card, and J. A. Landay. Prefuse: a toolkit for interactive information visualization. In *CHI '05: Proceeding of the SIGCHI conference on Human factors in computing systems*, pages 421–430, New York, NY, USA, 2005. ACM Press.
- [5] C. L. North. *A User Interface for Coordinating Visualizations Based on Relational Schemata: Snap-Together Visualization*. PhD thesis, University of Maryland, College Park, May 2000. Chair-Ben Shneiderman.