VizScript: Visualizing Complex Interactions in Multi-Agent Systems

Jing Jin and **Rajiv T. Maheswaran** and **Romeo Sanchez** and **Pedro Szekely**
Information Sciences Institute, University of Southern California
4676 Admiralty Way Suite 1001, Marina del Rey, CA 90292
+1-310-822-1511
{jing,maheswar,rsanchez,pszekely}@isi.edu

Abstract

We address the problem of users creating visualizations to debug and understand multi-agent systems. To expedite the process of creating visualizations, we have developed VizScript, a collection of tools, which combines a generic application instrumentation, a knowledge base, and simple scene definition primitives with a reasoning system . Using VizScript we were able to recreate the visualizations for a complex multi-agent system with an order-of-magnitude less effort than was required in a Java implementation.

INTRODUCTION

Understanding the behavior of complex software is challenging. Understanding the behavior of multi-agent systems is even more challenging given the additional timing and information sharing issues involved. We focus on building visualizations of multi-agent systems to help users understand the behavior and interactions among agents. While there are many tools to build visualizations of software (Bassil & Keller 2001; Stasko 1990), visualizations still require significant effort to build.

Our approach, embodied in a tool called VizScript, is to enable visualization authors to specify visualizations using rules of the form:

```
when <interesting-event-happens> {
  <paint-it-on-the-screen>
}
```

VizScript lowers the cost of creating visualizations by separating instrumentation and visualization. Software developers instrument the software to produce a stream of records of the form (a, o, t, p, v). Such records announce that on time instant t, agent a knows that property p on object o has value v. These records are added to the knowledge base one at a time. Authors define patterns to identify high-level events as the knowledge base is being built. They only need to understand the classes of objects in the application, their properties and values that properties can have.

Figure 1 shows two examples of the types of visualizations that VizScript can easily generate: graphs to show the evolution of numeric variables, and charts that show the evolution of nominal variables. VizScript supports the ability to play the visualizations back and forward in time to enable users to observe the state of the system at any time point.

ARCHITECTURE OVERVIEW

Figure 2 shows the architecture of the VizScript. The dotted line represents the VizScript interpreter. VizScript takes as input a collection of data streams and a collection of scripts that use rules to define the visualizations that the user wants to see. The condition part of a rule is defined using a pattern that is matched against the contents of the knowledge-base. It outputs visualization commands that are fed to a graphics library to drive the display.

VizScript is an interpreted language with an integrated knowledge base and pattern matcher. Besides standard assignment, arithmetic, conditional and function call expressions found in many scripting languages, VizScript script has a special rule definition construct of the form:

```
when Pattern where BooleanExpression{
  Statement+
}
```

Patterns specify events of interest. A pattern is a Boolean combination of primitive patterns of the form ($object$ $property$ $value$). The $object$ and $value$ elements can be variables of the form $?x$, or expressions, which evaluate to a scalar or an object in the knowledge base. The $property$ is an expression, which when evaluated yields the name of a property in the knowledge base. For example, the following are primitive patterns:

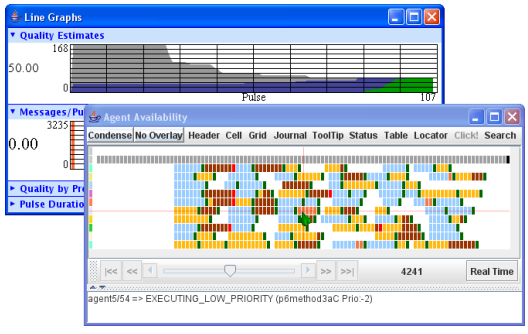

Figure 1: Two visualizations created using VizScript.

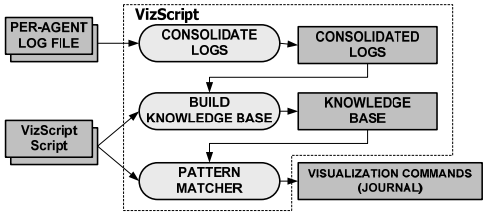

Figure 2: Components of the script interpreter.

Visualization	Lines of Code		Savings
	Original	VizScript	Factor
Quality View	122	5	24.40
Subjective View	141	21	6.71
Agent View	190	47	4.04
Probability View	258	23	11.22
Execution View	1214	93	13.05

Table 1: Effective Lines of Code for Visualizations

```
("task1" "probability" ?p)
(?x "probability" ?p)
```

The optional `where` clause of the `when` statement allows definition of additional constraints that cannot be expressed through unification of pattern variables (e.g., inequality of two variables), and it needs to be satisfied by the different variable bindings of the pattern.

A *binding* is an assignment of values to the variables in a pattern that make the pattern true. A pattern can have multiple bindings when multiple assignments satisfy it.

The body of `when` statement consists of visualization commands to build display like the one shown in Figure 1.

VizScript processes the consolidated input stream one record at a time: it adds the record to the knowledge base, and then it evaluates the `when` statements with primitive patterns that refer to the property just added. It binds variables in those primitive patterns to the object and value of the record just added. For example, suppose that the following record is added to the knowledge base:

```
("agent1" 0 "task1" "probability" 0.3)
```

Consider the following `when` statement:

```
when (?x "probability" ?p) {
  addCell(?x, ?p, "prob", ?x+":"+?p);
}
```

VizScript would evaluate this `when` statement because it contains the property `"probability"`. It would bind `?x` to `"task1"` and `?p` to `0.3`. Then it would execute the body of `when` statement, which paints a cell to (`"task1"`, 0.3) with the color "prob" and the description `"task1:0.3"`.

After binding pattern variables based on the record just added to the knowledge base, VizScript computes bindings for any variables that are still unbound in the pattern by matching against the full contents of the knowledge base. After satisfying the `BooleanExpression` in the `where` clause, it finally executes the body of the rule.

To query the knowledge base by the pattern matcher, VizScript introduces a `let` statement. Unlike the `when` statement, the bindings it produces are not required to contain the record just added.

RELATED WORK

VizScript is similar to Demetrescu, Finocchi and Stasko's (Demetrescu, Finocchi, & Stasko 2002) Interesting Event method, but we provide a reasoning system between the software and the visualization scheme, which simplifies instrumentation also making the approach more generic. Tominski and Schumann's (Tominski & Schumann 2004) approach is similar to VizScript. They use XML to define event templates, mapping them later to SQL queries. Our approach is more flexible, since VizScript makes inferences on the data set with respect to time.

EVALUATION AND CONCLUSIONS

Preliminary experience with VizScript is encouraging. Table 1 shows that VizScript enables approximately an 10X reduction in the number of lines necessary to create these visualizations compared to equivalent Java implementation.

VizScript is a promising and effective approach to creating dynamic visualizations for complex, large distributed systems. In addition, its negligible instrumentation overhead, off-line mode (streams are saved in files and loaded on demand) support, and its scalibility (handles millions of lines of log) also make VizScript a powerful, easy to use visualizations tool. Ongoing work focuses on further scale-up to data streams with several million records and on patterns to support temporal reasoning.

Acknowledgments

This work is funded by the DARPA COORDINATORS Program under contract FA8750-05-C-0032. The U.S.Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

References

Bassil, S., and Keller, R. 2001. Software visualization tools: Survey and analysis. In *9th. International Workshop on Program Comprehension (IWPC'01)*, 7–17.

Demetrescu, C.; Finocchi, I.; and Stasko, J. T. 2002. Specifying algorithm visualizations: Interesting events or state mapping? In *Revised Lectures on Software Visualization, International Seminar*, 16–30. London, UK: Springer-Verlag.

Stasko, J. T. 1990. Tango: A framework and system for algorithm animation. *Computer* 23(9):27–39.

Tominski, C., and Schumann, H. 2004. An event-based approach to visualization. In *IV '04: Proceedings of the Information Visualisation, Eighth International Conference on (IV'04)*, 101–107.