On the monitoring and analysis of OO systems

A position paper for the OOPSLA '93 Visualization workshop

Jeff McAffer

Department of Information Science
The University of Tokyo
and
Object Technology International
jeff@is.s.u-tokyo.ac.jp

August, 1993

1 Introduction

Current monitoring and analysis tools are only partially applicable to object-oriented systems. Many assume and require system models where code is fixed, types exist and are known, variables and storage are predeclared and system behaviour is *regular*. In large complex object-oriented systems, very few of these premises hold, especially during development. Perhaps the most damning is the lack of regularity. *Irregular* systems are those whose computational and communication behaviour or characteristics change over time (i.e., are non-deterministic). Such irregular systems break most existing analysis and visualization techniques by contradicting assumptions made by tool designers.

There are many different object-oriented environments, each having their own characteristics. To generalize and talk about object-oriented system analysis, we will assume only that the environment is message-based and admits, at least conceptually, late binding of variables and functions. In our search for applicable technologies, we have looked at domains with similar behaviours. Concurrent systems is one of the most apt. Much of the work done in that community relates to message/event tracing, analysis and visualization. The developed technology can be directly applied to simple object-oriented systems but is not readily available for use with complex system for the same reasons it fails to support complex concurrent applications – the magnitude and depth of interactions overwhelm these straightforward attempts at analysis and visualization. The tools are not powerful enough to deal with the quantity and complexity of data available.

Irregularity limits our ability to filter and correlate events of interest. Call graphs, time-lines, object relationship diagrams, object state meshes and the like all convey too low a level of data about too few entities. This statement highlights two of today's problems; poor analysis techniques (low data level) and limited visualization tools (to few entities – lack of scalability) which are inter-related. We have reasonably good low-level tools for investigating stack states etc. but in a complex system we deal with millions of objects. It is inconceivable to approach even a limited subset of these problems at the level of drawing individual objects and message sends. Clearly we need abstractions.

2 Vibes

The burden of behaviour analysis currently falls on the user. Tools gather and present data but do not sufficiently analyze it. To rectify this, we propose a system called Vibes. Vibes is a data gathering and analysis framework underlying a set of tools designed to assist the user in identifying and understanding interesting behaviours. Analyzed behaviour is *rendered* and input to the user. The gathering, analysis and rendering must be scalable in depth (level of data handled) and capacity (amount of data handled) to enable the identification of areas of interest and their detailed investigation.

We will discuss only the analysis and visualization components of Vibes here. Readers should assume the existence of a dynamic, reflective, adaptive instrumentation system. The analysis framework is fundamentally a dataflow graph. Observed data (events) are input at various points in the graph and processed by the functions at the graph nodes. Processed data is passed on to the next node for further processing. This idea is not new. In fact, the more general concurrency monitoring tools (e.g., Pablo [Pablo]), employs something similar, as do systems for scientific data visualization (e.g., Explorer [SGI] and AVS [Stardent]). The difference in Vibes is that the graph can be arbitrarily complex (e.g., coarse or fine grained) and can handle any kind of data and processing. Pablo's analysis nodes typically contain very simple functions (e.g, summation and thresholding) and are static in the types of data they can process. Explorer nodes are more sophisticated but are not capable of handling arbitrary data (i.e., objects) and are somewhat coarse-grained.

Output is taken from the analysis graph at a point of interest. Data for output can consist of a single event, a stream of events or some arbitrarily complex abstract object representing the analyzed behaviour. Outputs typically form the basis for producing input to users (rendering) but can also be used to feed control information back into the monitoring system or the system being observed. An important part of rendering is *mapping*. Mapping done as an extension of the analysis graph. It is the process of converting the analysis output data into a form useful for creating the user-perceptible input. Simple mappings may transform values to colors or tones while more complex ones may provide position, trajectory or multi-view transformations. Mappings can model abstract shapes and structures and are used to coordinate multiple sequential renderings.

3 Using Vibes

Give such a framework, what sort of analysis and output can we expect to produce? On the simplest level, Vibes can produce the typical object relationship and control flow displays as well as standard graphs (e.g., scatter plots and time-lines). As analyzed systems get larger, there is a higher volume of data and the job of analysis becomes more difficult. Abstraction and filtering begin to play significant roles. Abstraction is the process of taking sequences of events and known system state and building higher-level representations which have more semantic meaning. For example, a particular type of service request may involve the exchange of many messages to negotiate for, request and get some service. The recorded events would register this sequence but would not include the abstract *service request* event. Though analysis, we can recognize this pattern and replace its occurrences with a single abstract event. This reduces data volume and raises the semantic content of the trace data.

Even so, the bulk of any given set of raw behaviour data is useless to the user. It is not related to the behaviours in which they are interested. The hard part of analysis is determining what is interesting and isolating it without loss or distortion. Parameterized instantaneous filtering (e.g., thresholding) is easily implemented and powerful however, does not scale well with respect to data level. We need the ability to filter on higher-level, abstract (symbolic) values related to both system and event state. Another important analysis tool related to both abstraction and filtering is correlation. Much of our misunderstanding of system behaviour can be traced back to unexpected relationships between objects. Correlation techniques can identify such relationships.

On reviewing the above list of desirable analysis techniques we see that many are implicit to the signal processing domain. Signal processing is well known for its ability to handle multiple, complex, interrelated, time-variant, high bandwidth channels or tracks. This is ideal. Vibes integrates Digital Signal Processing (DSP) techniques into its analysis framework. Streams of data flowing over the graph edges can be treated as signals. This makes available several new analysis and rendering techniques. Filtering can be done on a frequency basis isolating periodic sequences of events from frequently occurring "random" events (i.e., noise). Isolated sequences can be further filtered based on the frequency of some event types and then correlated with other event sequences to see if they are interdependent. Adaptive signal processing techniques allow complex sequences of events to be tracked as they change frequency over time (e.g., as system performance degrades) aiding in the detection of critical degradation points.

Treating the data as signals leads to obvious, though not direct, analogies to sound and light – both frequency based senses. We are exploring the possible output transformations in both domains. There have been some experiments with sound mappings which, while rudimentary, (e.g., a message send by object A is mapped to some tone on the left speaker, etc.) have demonstrated promise [Pablo, Francioni et al 91]. Sound locality mapping is another important aspect especially when combined with 3D graphics and immersive technologies. These areas are of great interest to us.

A note on output mappings. We must take care to think beyond the present day concepts of output, both visual and sonic. While graphs and plots are very important tools for quantitative data, they are not so good for higher-level data which is not easily quantified. Mappings from analyzed data to output need not be direct. For example, experiments with mapping time-sliced stack information to L-System [Prusinkiewicz 90] grammar inputs have lead to interestingly regular graphics for recursive operations like quick sort [McGuire 91]. Such a technique highlights the recursiveness, or high-level behaviour rather than the more quantitative data such as stack depth.

4 Conclusions

Though object-oriented programming is much vaunted for its intuitiveness and similarly to the real world, the fact is that many implementation behaviours bear little or no resemblance to any naturally occurring phenomena. Consequently, we have little intuition as to what those behaviours (should) look/feel/sound like. Given this, it is unreasonable expect rendering techniques to be immediately intuitive. It is not unreasonable that users should *learn* what previously unseen behaviours look like! For example, the waterfall display used in sonar systems is not immediately obvious but is none the less very powerful. As a research direction, we should seek to find new rendering techniques which highlight behavioural features and reduce the user's analysis load. It is a somewhat organic concept but we should build tools which allow users to *get a feel* for the system before they dive into the depths of its low-level interactions.

Our work on Vibes is being applied to massively concurrent object-oriented systems. In this domain parallelism contributes greatly to the complexity and seeming (though often not actual) independence of object behaviour. We have not even begun to explore the full power of DSP in this domain but it is clearly a promising direction. The data as signals view puts time back into the analysis equation in a significant way.

References

[Pablo] Daniel A. Reed, *An overview of the Pablo performance analysis environment*, Department of Computer Science, University of Illinois, 1992, ftp://bugle.cs.uiuc.edu/

[Prusinkiewicz 90] Prusinkiewicz, P. and Lindenmayer, A. *The algorithmic beauty of plants*, Springer-Verlag 1990.

[Francioni et al 91] Francioni, J., Albright, L., and Jackson, J. *The sounds of parallel programs*, in the Proceedings of Sixth Distributed Memory Computing Conference 1991, IEEE Computer Society.

[McGuire 91] Kevin McGuire, Personal communication.

[SGI] Silicon Graphics Inc. *Explorer User's Guide*, 1992.

[Stardent] Stardent Computer Inc. *Application Visualization System, User's Guide*, 1989.