Position Summary: Architectures For Adaptation Systems

Eyal de Lara[†], Dan S. Wallach[‡], and Willy Zwaenepoel[‡]
Departments of Electrical and Computer Engineering[†] and *Computer Science*[‡]
Rice University
{delara,dwallach,willy}@cs.rice.edu

1 Introduction

Modern systems need support for adaptation, typically responding to changes in system resources such as available network bandwidth. If an adaptation system is implemented strictly at the system layer, data adaptations can be added within the network or file system. This makes the adaptation system portable across applications, but sacrifices opportunities to change an application's behavior. It's not possible, for example, to first return a low-quality version of an image and later upgrade it should excess network capacity be available. On the flip side, the adaptation logic could be built into each and every application, with the system providing information to the applications in order to help them adapt their behavior. This becomes impractical because many applications will never be written to perform adaptation, and an application writer may not be able to foresee all possible adaptations that may be desirable.

We argue that adaptation systems should be centralized, where they can make global observations about system usage and resource availability. We further argue that applications should *not* be written to perform adaptation. Instead, applications should support an interface where the adaptation system can dynamically modify an application's behavior as it runs.

How would such an interface work? Largely, we would like applications to make visible their *document object model* (DOM) – the hierarchy of documents, containing pages or slides, containing images or text, etc. Likewise, we would like a standard way to know what portions of a document are on the user's screen. Finally, it's quite helpful when the file formats are standardized, such that the system can see and manipulate the components within a file.

In order to support adaptation while documents are being edited, we would like a standard way to learn which components are "dirty" and to compute diffs between those dirty components and their original contents. Likewise, it would be helpful for applications to support conflict detection and resolution between components.

2 Experience

We observe that many "component-based" applications already support interfaces for external programs to manipulate their components as the application is running. Taking advantage of this, we developed a system called Puppeteer [1], as it "pulls the strings" of an application.

Puppeteer currently supports Microsoft Word, PowerPoint, and Internet Explorer, as well as their StarOffice equivalents. In terms of implementation complexity, Puppeteer has roughly 8000 lines of Java code shared across applications. The Internet Explorer drivers are 2700 lines and the PowerPoint drivers are 1800 lines of code.

Our current system supports adaptation for read-only files. We achieve significant improvements in user-perceived latency at a modest cost in system overhead.

3 Future Work

Building on the base Puppeteer system, we are working on a number of extensions. We are investigating a "thin client" version of Puppeteer to minimize the client memory footprint – an important consideration on PDAs. We are designing a special-purpose language to express adaptation policies at a high-level. We are investigating alternative network transmission protocols and hierarchical scheduling of network transmissions to better reflect the priorities of the adaptation policy. We are also working on extensions to Puppeteer to support writes, dealing with issues like cache coherence and conflict resolution. So far, the Puppeteer architecture has proven flexible enough to accommodate such a wide variety of extensions without sacrificing its portability or core architecture.

References

[1] E. de Lara, D. S. Wallach, and W. Zwaenepoel. Puppeteer: Component-based adaptation for mobile computing. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, California, Mar. 2001.