An Adaptive Programming Framework for Web Applications

Po-Hao Chang, Wooyoung Kim and Gul Agha
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Ave., IL 61801, USA
{pchang2 | wooyoung | g-agha}@uiuc.edu

Abstract

Web applications service a gamut of users with a mix of static and dynamic resources: requests come from different devices running different Web agents (typically, Web browsers) with different capabilities. In a majority of such interactions, services are embedded in an HTML page and displayed on a Web browser. Consequently, Web application development has been tightly coupled with the platforms that a particular site intends to support; application developers have been asked to decide "where to run what" at design time to develop services using location-specific technologies and interweave them with static resources. Such difficulties stem from lack of adequate infrastructural support for platform-agnostic development and deployment framework. This paper presents the design and implementation of a framework for Web application development which hides the heterogeneity of the Internet and the underlying interaction model. It has been meticulously designed such that it is non-intrusive and requires little change to the existing Web infrastructure. The framework is built on top of a programming model that provides a uniform view across different platforms and thus hides the heterogeneity and distributivity of the Internet. Adaptive deployment involves placement specification and compiler translation. The uniformity gives rise to rapid prototyping and development, thereby reducing costs and improving productivity. The adaptiveness enables QoS aware service provisioning.

1. Introduction

After a decade of evolution the Web has become a "living organism," feeding on fresh data, producing new information and services, and shedding obsolete ones. We observe two important results of the evolution. First, Web servers increasingly serve as an information and service conduit for organizations. More and more raw data are generated from geographically distributed dynamic sources (e.g., network monitoring tools, a network of sensors, etc). Web applications process such raw data (i.e., *dynamic* contents) and combine them with static elements to compose an HTML/XML page on the way to users (i.e., dynamic page generation). Some applications sprinkle HTML pages with codelets to harness computational resources at client side for shorter response time and less communication traffic. Second, the simple client-server model of the Web gave way to a *deep* infrastructure where multiple intermediaries (e.g., edge servers [11, 6, 10], proxy caches [1, 4]) are transparently located in between clients and servers. Typically, client, intermediary and server platforms carry different technologies (e.g., VBScript, JavaScript). Given such a highly non-uniform execution environment it is clear that a monolithic approach to Web application development cannot fulfill equally well different quality-of-service (QoS) requirements of diverse users.

Instead of addressing the problem from the root, the Web community chose an easy solution of making Web application development platform-dependent. For example, a Web site built with PHP may want to service a request from Microsoft Internet Explorer 6.0 which supports VBScript over a broadband connection, and the response may be cached in Akamai's edge servers. We call this combination of platforms and technologies a *request context*. Given a request context developers are asked to answer the question of "where to run which codelet" at design time and then implement each codelet using the particular technology supported at the platform on which the codelet will be executed. In practice, most Web sites do not support all possible platforms and technologies. Rather, they settle with dominant technologies, alienating "non-conformist" users with the sign of 'this page is best viewed with Browser X' [14]. For those Web sites the reasoning is purely economical: to keep up with technology advances and to provide a certain level of QoS guarantee, they can afford to support only a few dominant request contexts. The difficulties stem from lack of appropriate infrastructural support.

We argue that platform-agnostic development and adap-

tive deployment may serve as a cost-effective alternative to today's Web application development practice. Platform-independent application development and deployment offers other benefits. First, it enables portable application development which reduces development and maintenance cost. Applications become immune to external environment changes, such as upgrade of interconnection networks and introduction of a new information appliance. Second, different users are guaranteed a certain QoS level. Those who access a site from a less capable platform are not overly penalized. Nevertheless, the sheer volume of the Web and its *laissez-faire* nature present unique challenges for platform-independent programming. A naive design may require substantial changes to the existing Web infrastructure. We all know well that Java's attempt to create a uniform execution environment on the Web has enjoyed only a limited success.

We present a framework for platform-agnostic development and adaptive deployment of Web applications [1] and describe its design and implementation. The framework consists of four components: a programming model, a scripting language, a placement specification language, and a retargetable translator. We designed the framework with great care such that it is non-intrusive and easily plugged in with little change to the existing Web infrastructure.

The platform independence comes from separating placement concerns from application logic [7, 8] and using an object-based programming model to describe the latter. Abstractions in the programming model create a uniform view (at the conceptual level) over different platforms and hide the heterogeneity and geographical distribution of the Internet below it. The uniformity gives rise to rapid prototyping and development, thereby reducing development costs and improving productivity. Adaptive deployment involves separate placement specification. The translator takes a platform-independent application implementation and generates a customized version according to a given placement specification. The explicit translation with the separate placement specification exposes platform specifics rather than hides them, enabling platform-specific optimizations and QoS aware service provisioning.

2. Motivation

Two most critical decisions that Web application developers have to make at design time are "where to run" and "when to load." The desire to find the best answers to the questions motivated our work reported in the paper. In a traditional distributed environment where certain uniformity is taken for granted (e.g., Java Virtual Machines), the decision making can be put off till deployment time with little adverse implication. By contrast, Web application developers should make right decisions early in the design phase since reverting them later would cost dearly. In this section we examine these issues in detail.

Location matters. Dynamic page generation is the most essential functionality of almost all Web applications. For one thing, it enables personalized browsing. Before the advent of DOM-enabled browsers the dynamic page generation has been done exclusively at server side. For a DOM-enabled browser, however, a Web server may harness the browser's dynamic content modification capability to delegate the responsibility to the browser by sending a page generation script along with raw data. Client-side page generation is preferred in terms of network bandwidth when a final HTML or XML document is much larger than the raw data in size. Also, it may help reduce server load when the page generation is time consuming. On the other hand, processing classified data and executing a sensitive processing logic should remain within the server boundary. Input validation is another example. User inputs can be validated locally at client side when simple rules are used. User inputs must be sent to the server, however, when they need to be validated against sensitive information. If a client is connected through a secure connection and can be trusted, some of such validation can be done at the client. As such, placement decision is a critical element in Web application development as it directly affects application performance. Once made, these placement decisions are costly to change since Web servers and Web agents usually carry quite different technologies. For example, they may support different scripting languages.

Timing matters. Consider the multi-level menu selection in a typical Web page. In a two-level book selection menu, the first level menu contains genres and the next level submenus contain individual books belonging to a particular genre. An optimization technique that many Web sites employ for the multi-level menu selection is to load all available menu entries with a Web page. A browser's displaying an appropriate sub-menu locally according to user's selection means better response time and less network traffic. However, if a user accesses the site over a slow or intermittent network connection and the number of choices is very large, the user may experience unnecessarily long loading time. Moreover, if the next level menu entries depend not only on the current selection, but also on dynamic data kept at the server (e.g., books in stock), it would be more desirable to request the server to send an appropriate sub-menu. Certainly, sending a page request explicitly does not come without cost; if a browser is not capable of doing incremental modification (which only a few state-of-the-art browsers have, at least for now), the server should regenerate the whole page with the new data for every new request

[1] We confine the discussion to Web contents. But our work can be naturally extended to more general Web services. Delivery of Web contents is a kind of Web service, after all.

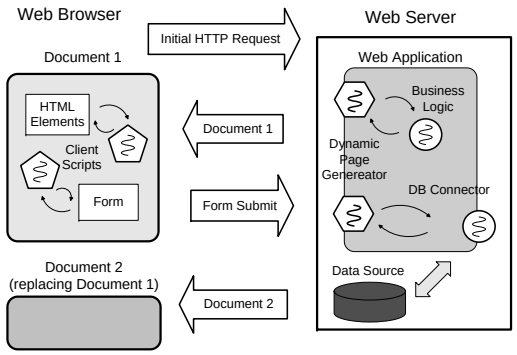


Figure 1. The Web interaction model. Without the incremental modification capability at the browser, the server needs to regenerate `Document2` and the browser needs to purge the display of `Document1` to render `Document2`, however similar the two are.

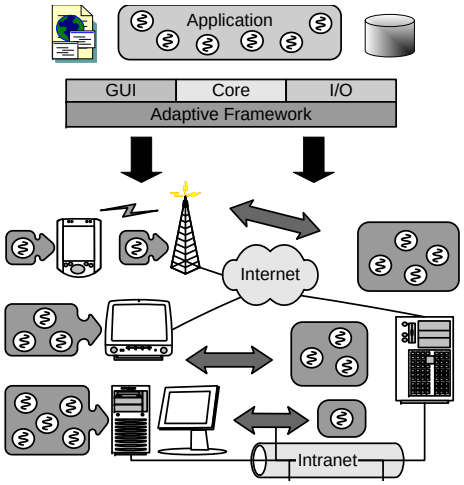


Figure 2. The framework. Programming at a higher level of abstraction enables adaptive deployment over heterogeneous platforms.

(Figure 1).

The analyses led us to the conclusions that the inadaptability of today's Web applications to different QoS requirements is attributed to the practice of platform-dependent application development and that supporting platform independence is the first step towards QoS-aware service provisioning.

3. Framework for Platform Independent Application Development

In what follows we present the design of a framework for platform-independent development of Web applications. The underpinning idea of the framework is to separate location and placement aspects from application logic and allow programmers to specify desired functionality on a uniform virtual execution environment built on an object-based programming model. The specified location-agnostic application is combined with a placement specification to generate a platform-specific instance before actual deployment (Figure 2). We start with some critical analyses on the current Web infrastructure and justify design choices we made in the development of the framework. Then, we give the programming model and briefly describe how applications can be deployed.

3.1. Critical Analyses

As the Web evolves, more and more dynamic contents are serviced by Web servers and browsers. For example, most Web servers customize documents per a request basis using dynamic page generation. An increasing number of browsers support interactive documents which respond to user's actions for better Web experience. These dynamic contents usually require complex interaction between client and server objects. And for QoS aware service provisioning, these components need to be relocated per a request basis. However, a few properties in the current Web infrastructure make arbitrary component relocation difficult.

- HTTP is simple and efficient for document retrieval. Yet its asymmetry and statelessness make it less attractive for location-agnostic development of distributed applications (in particular, Web applications). The one-way initiation makes server-to-client remote function invocation all but impossible. Many Web servers rely on ad hoc techniques, such as cookies and specially arranged hidden form elements, to maintain session states.
- The current Web access is based on a simple client-server model. This dichotomy has allowed each side to undergo independent evolution. The end results are mutually incompatible, diverse document object models in Web servers and browsers, and different technologies built upon these models.

These observations led us to use of a uniform object-based programming model which blurs demarcation between client and server objects and between local and remote objects, and obviates the notion of location.

3.2. Programming Model

The programming model is based on objects, in which objects create other objects dynamically and interact with

each other via message passing. Sharing the same programming model at a higher level not only enables use of a single programming language with consistent APIs, but also reconciles the incompatibilities between low-level implementation models at the two ends. The abstractions provided by the model hide location-specific distinctions among clients and servers and make HTTP requests and responses invisible to programmers. All entities involved in executing Web applications are represented as objects in the same address space (albeit virtual). In addition to user-defined objects the model defines a few system modules that implement essential services for application execution.

Core module serves as the skeleton of a Web application. It defines events for application execution including initialization and termination, and allocates a placeholder to each of entities shared among objects in the application. It can be thought of in analogy to *main()* in the C programming language.

GUI module abstractly represents visual effects rendered on a user's display. It contains an event model upon which all GUI components and window layouts are defined. The event model is defined similarly to the DOM model [5].

– *Controls* package basic HTML elements associated with event handlers.

– *Views* are containers of Controls. Programmers use Views to construct advanced controls from basic ones; for example, an annotated input box can be composed by a View containing an input box control and texts.

– *Windows* encapsulate browsers. Other objects interact with a browser using the corresponding Window object as a proxy.

I/O module includes server as well as client file system objects (for file upload if available), and database and network connection objects (to other Internet services).

3.3. Implementation Approach

A naive strategy to implement a platform-independent programming framework for Web application is to use specialized software and proprietary communication protocols. We believe the sheer scale of the Internet and the diversity of client platforms render such a strategy impractical. Note that only the Web server is in a service provider' total control. Also, note that the wide use of firewalls disallow use of non-standard communication protocols.

Instead, we have adopted an approach that involves source-to-source translation to cope with diverse execution environments. A developer writes a Web application using a platform-independent programming language as if it ran on a single platform. Before being deployed, the application undergoes source-to-source translation with a separate placement specification. The translation is necessarily platform- as well as location-dependent; depending on

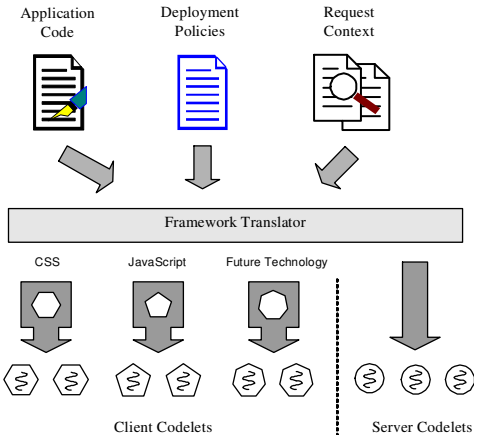


Figure 3. Source-to-source translation. Depending on the deployment policies and request context, the framework translator implements application objects using different technologies including server scripts, client scripts, and Cascading Style Sheets.

platforms involved and the placement specification given, different parts of the Web application are translated to target codelets in different implementation languages. For example, an advanced Web browser may receive XML documents with complex client scripts while a thin client will get basic HTML documents with most of computation being done at server side.

This translation-based approach offers better scalability and adaptability. Without rewriting applications service providers can have them take advantage of the latest technology as soon as it becomes available, and adapt to changes in infrastructural support by extending the translator. Even system migration becomes feasible: switching from J2EE to ASP.Net, for example, can be readily achieved by employing a different translator. The overhead incurred by the translation process can be managed tolerable since Web servers may opt to cache translation results and reuse them for other compatible request contexts. Also, it does not necessarily introduce any additional security risks because the baseline framework on which the approach is built has already been deployed and managed by existing platforms and technologies on the Web.

3.4. Deployment

Ideally, deployment choices of "where-to-run" and "when-to-load" should be automatically made by an underlying system employing aggressive optimization techniques. Indeed, it is one of our future research directions. The framework currently stops short of full automatic de-

ployment and supports only a static deployment approach involving explicit source-to-source translation of an application with a placement specification (Figure 3).

A placement specification is described by an object deployment sheet (ODS) which resembles Cascading Style Sheets (CSS) [3]. An ODS specifies how to place objects in a Web application by listing their meta properties, such as location, security requirement and supporting platform. Thus, a separate ODS would suffice for different deployment of a Web application. Locations may be specified at different granularity – at the *class* level, at the *member variable/method* level, and at the *object* level. Conflicting properties are resolved by having those specified by a more specific rule override less specific ones, as in CSS.

4. Implementation

In this section, we describe a prototype implementation of the framework. The translator and framework objects are designed for Microsoft Internet Explorer 6 (MSIE) and Active Server Pages (ASP). Since we use a subset of the W3C DOM Level 1 API and common features of Web servers in ASP, it is straightforward to port our implementation to other DOM Level 1 browsers and Web servers.

4.1. Framework Language

Any programming language can be used for the framework language. We decided to use one of scripting languages since they are popular for simplicity and flexibility. The two properties which contributed most to their success also make them attractive for Web application development. Being simple makes them easy to learn, an important factor for Web developers, especially those with less technical background. Being flexible means applications written in a scripting language are amenable to modification. Another reason to favor scripting languages is that they are ubiquitous in the Web. Almost all browsers and servers support at least one kind of scripting languages. We chose JavaScript (i.e., ECMAScript) as the framework language. The most critical factor for the choice was its near omnipresence; even Microsoft IE and ASP product lines support a version of JavaScript (i.e., JScript). Such omnipresence means less work for the translator.

4.2. Packaging Host Objects

Applications in scripting languages are often executed within host programs, such as Web servers and browsers. In our prototype implementation, the host programs are MSIE (at the client) and ASP (at the server). We call objects provided by these host programs *host objects*. MSIE and ASP already provide JavaScript interfaces to host objects in an application's I/O module as well as to Window objects in the GUI module. The Core module can be implemented using application and session objects in ASP. Thus, we describe the implementation of Control and View objects in the GUI module only.

A GUI Control object is implemented as a DOM Element node with associated event handlers. Because a DOM Element node can represent an HTML element, using the DOM API and scripting node styles enable us to control its visual presentation. Similarly, a GUI View object is realized with a DOM node containing other View and/or Control nodes. We distinguish GUI Control and View objects mainly because GUI Control objects are predefined while GUI View objects can be assembled with more basic components. The difference results in different implementations when they are deployed at client side because almost all browsers support basic HTML controls whereas simple browsers cannot handle composite DOM fragments by themselves.

Since GUI objects are implemented as DOM nodes, it is natural to deploy them at client side if the browser is DOM-enabled. For browsers which are not DOM-enabled, these visual components remain at the client but operations are performed at the server using the translator-generated DOM objects as proxies. The visual components are synchronized with DOM objects along HTTP communication paths.

4.3. Object Deployment

Since both MSIE and ASP support JScript, deploying a user-defined object at either end does not make any difference. Rather, the challenge is to implement bi-directional remote object access over HTTP connection. Accessing object properties and invoking object methods are implemented with remote function calls. Because of HTTP's asymmetry, we need to handle client-to-server and server-to-client remote function calls separately.

Client to Server It is easy to implement remote function calls in this direction. For SOAP-enabled browsers, packaging server-side objects into SOAP objects achieves the goal. Other advanced browsers allow client scripts to import XML documents for further manipulation. In such cases, a server object is implemented as a dynamic server page which returns a result in an XML document. The caller at the client retrieves the result from the returned XML document. For discussion on less capable browsers, refer to Section 4.5.

As an illustration, consider two functions $f()$ and $g()$. Suppose $f()$ is deployed at a server and $g()$ at a client.

```
function f(x) { return x+1; }
function g() { var c = f(123); }
```

The translator will generate client scripts for $g()$ as follows:

```
function g() {
  xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async=false
  xmlDoc.load("http://[server]/[app]/f.asp?x=123");
  var root = xmlDoc.documentElement;
  var c = root.childNodes.item(1).text;
}
```

For $f()$ the translator will prepare a specially designed active page $f.asp$ in the application directory. This active page fetches arguments and returns an XML document containing the result.

```
<?xml version="1.0"?>
<function>
<name>f</name>
<return>
<%
function f (x) { return x+1; }

var x = request.QueryString("x");
response.write(f(x));
%>
</return>
</function>
```

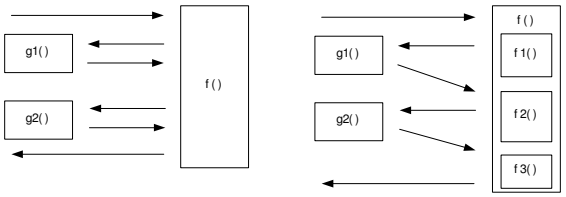
Server to Client This half is more challenging since there the HTTP protocol does not support any simple way for a server to push data to a client without the client's HTTP request. As before, we assume $f()$ and $g()$ are at a server and at a client, respectively. To implement a call from $f()$ to $g()$, we may exploit the observation that Web applications are event driven and in almost all cases these events are exclusively from client side [2]. Hence, we note that the call to $g()$ from $f()$ must be in the context of a call to $f()$ from the client.

First, we consider the case of the call to $g()$ being the last expression in $f()$, i.e., $f()$'s tail call. Here, $f()$ returns to the caller either what $g()$ returns or nothing. Our solution to the case is to let the server function (e.g., $f()$) return before invoking a client function (e.g., $g()$) and piggyback the call request in its response. We add a new XML element $<$*tailcall*$>$ whose content is the remaining part of the server function which should be executed in the client, i.e., $g()$ in this case. The resulting server script for $f()$ will look like:

```
<?xml version="1.0"?>
<function>
<name>f</name>
<return>
<%
function f (x) { return g(x+1); }

var x = request.QueryString("x");
%>
</return>
<tailcall>
g(<%=x+1%>);
</tailcall>
</function>
```

[2] Server events usually come with some form of concurrency and we do not consider this issue in the paper.



(a) before split (b) after split

Figure 4. Splitting a server-side function. $f()$ is split into $f1()$, $f2()$, and $f3()$. $f1()$ and $f2()$ contain a tail call to $g1()$ and $g2()$, respectively.

When the caller at the client receives the reply, it retrieves the server request from the reply, executes $g()$, and uses its result (if any) as $f()$'s return value. The client codelet will look like:

```
xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async=false
xmlDoc.load("http://[server]/app/f.asp?x=123");
var root = xmlDoc.documentElement;
var c = eval(root.childNodes.item(2).text);
```

If $g()$ is not $f()$'s tail call we split $f()$ into multiple sub-functions so that each server-to-client call is a tail call of one of the sub-functions (Figure 4). However, this conversion is presented just for completeness, and therefore, is not encouraged. In practical deployments, it is inefficient – handling an event at the client may result in a full cycle of request/response – and sometimes unnecessary. Only client side objects in the I/O module are required to reside at the client (i.e., it does not make sense to relocate them), for which operations can be executed collectively at the end of the event handler. We will discuss a more efficient solution for the other objects in Section 4.6.

4.4. Deployment Specification

In the current implementation an ODS may contain only location properties of objects with two available options – client and server. For the sake of simplicity we use the CSS syntax for ODS specification. The syntax is as follows:

```
class|class.member|#variable{location:server|client}
```

Suppose we have a class `foo` which has three methods `m1`, `m2` and `m3` and the following placement specification and codelet are given for `foo`. A name prefixed with a hash symbol in the placement specification corresponds to the variable with the same name in the codelet.

```
<placement specification>
foo {location:client}
foo.m3 {location:server}
#foo1 {location:server}
#foo2.m2 {location:server}
```

```
<codelet>
var foo1 = new foo();
var foo2 = new foo();
var foo3 = new foo();
```

Variable `foo3` will follow the placement specification of class `foo` as it has no other specification. By contrast, all three methods of variable `foo1` will be executed at the server as its placement is specifically specified in the specification. `foo2.m2` and `foo.m3` will be executed at the server (by `#foo2.m2` and `foo.m3`, respectively) while `foo.m1` will be executed at the client. As discussed in Section 3.4, properties defined by a more specific rule override conflicting properties defined by a less specific rule.

4.5. Support for Different Browsers

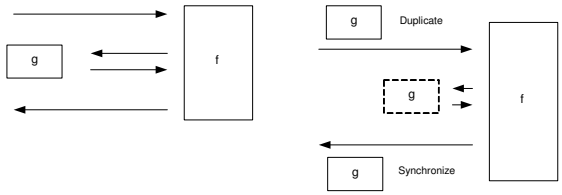
We cannot assume powerful client script support from simple browsers such as those found in PDAs and cell phones. A straightforward solution for these browsers is to forward events to the server (or other platforms, such as surrogates or gateways) and emulate a DOM-enabled browser at the server. The deployment can be obtained by assigning the server to all elements in an ODS. For fourth-generation browsers (i.e., DOM Level 0) which can change some properties of DOM nodes but cannot add, move, or delete these nodes by themselves, the translation can be done by assigning the server to these operations. For those that are neither SOAP-enabled nor capable of loading XML documents using client scripts, we simulate remote function calls by creating a hidden frame, loading the function page into the frame, and fetching the return value from it.

4.6. Optimization

Here we present a simple yet effective technique to reduce the number of round trips between a client and a server. As noted earlier, Web applications are event driven and most events are generated at client side. In case an event handler has to access the server, it would be cost-effective to process all required computation during a single request-response cycle. We would like that, when the control is transferred to the server, it is not returned to the client until no more server side execution is required. However, a problem arises when a program or a deployment interleaves client-to-server and server-to-client remote function calls. Our solution to the problem is to duplicate client objects at the server. When the event handler returns, these duplicates are synchronized with their corresponding counterpart at the client. Not all objects at the client need to be duplicated; only those that fall into the pattern of interleaving access will be duplicated.

4.7. Consistency Management

One caveat of the optimization is the fact that synchronizing object states and keeping them consistent could be very costly when they execute concurrently. In fact, Web applications are composed of otherwise independent client and server components, and some mechanisms are available to support their concurrent execution. Fortunately, most Web applications do not have true concurrency. For example, multiple events could be triggered concurrently in a Web page containing multiple GUI controls. However, current browser implementations effectively deliver an event at a time; no events are delivered while a browser is busy handling an event. Also the HTTP protocol is synchronous: browsers do nothing but waiting when an HTTP request is outstanding. If a new HTTP request is sent before the response of the last one returns, the new one cancels the pending return. These constraints considerably simplify consistency management of object states: the states of duplicated objects are synchronized along the execution paths through HTTP requests and responses.



(a) before optimization (b) after optimization

Figure 5. Optimization. To reduce communication between a client and a server some client-side objects are duplicated at the server and subsequently synchronized.

5. Related Work

ASP.NET [9] supports to some extent platform-independent development and customized deployment of Web applications. It packages HTML controls into server HTML controls and allows developers to design their own controls as server side controls. A copy of each client's view is kept at a server where all events for server controls are processed; event handlers directly operate on the copy without recreating a fresh one for a client. Specially designed server controls are employed to support customized deployment. Similar to our approach an ASP.NET server control detects the capabilities of a requesting client and morph itself into different code. However, the development cost is not negligible; it requires a good deal of careful programming of server controls. For instance, developers have to write the detection code, and then, manually arrange the deployment. For this reason many third party server controls emit the identical code for all browsers without exploiting the feature. To our best knowledge built-in Web controls in Microsoft Visual Studio.Net are the only ones that are optimized for Internet Explorer browsers by exploiting the

feature.

Java virtual machines on different physical platforms create an illusion of uniform execution platforms as a whole. However, the illusion does not hide all location-specific details. For example, Java [13] provides applets and servlets for Web programming. Without some specific arrangement it is difficult to make Java applets and servelts interact with each other. Unless a Java object does not access Java system objects it is very cumbersome to relocate the object between a Java applet and a servlet. Java is a powerful programming language for Web programming – it can provide multimedia, networking, and multi-threading support. However, it is less flexible than most scripting languages; it is more difficult to modify and extend existing Java code. Moreover, it lacks control over a browser. Most importantly it is still not omnipresent.

Inspired by Java's platform independence and popularity, various research work proposed Java extensions to support transparent distributed computing on the Web. JavaParty [12] introduces a new `remote` class modifier to annotate classes whose instances should be distributed across different hosts and to hide remote communication details from programmers. The JavaParty runtime performs object distribution and migration using a "distribution strategy," which the runtime may change to another one depending on program execution. A programmer may specify multiple distribution strategies, but they must be given prior to program execution. Java// [2] (read Java parallel) is a distributed programming language based on concurrent objects which supports the Java syntax, a high-level inter-object synchronization mechanism through "future" objects, and static specification of an object distribution strategy. It is implemented by using a class library. Each of these systems propose unique implementation techniques that we can exploit for implementing our framework. However, all of them are system-dependent and do not satisfactorily address the challenges posed by the intrinsic heterogeneity and dynamicity of the Web.

6. Conclusion and Future Work

We proposed a platform-agnostic programming framework for Web applications. The framework allows Web developers to focus on the functionality and behavior of applications and to avoid being distracted by differences in implementation technologies as well as the heterogeneous platforms. A translator in the framework takes a platform- and location-independent Web application and a placement specification to generate a platform specific application. Currently we are working on comprehensive implementation of the translator for various Web servers and browsers. We plan to extend our framework to support edge servers, P2P networks, and even Grids. Our current implementation uses static translation and caching to simulate adaptive behaviors of Web application. We believe the static translation ultimately give way to dynamic just-in-time translation with fully automatic adaptive deployment. As our future research work, we plan to explore feasibility of automatic application partitioning and adaptive re-deployment of Web applications with continuous service profiling.

References

[1] P. Cao and S. Irani. Cost-Aware WWW Proxy Caching Algorithms. In *Proc. of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, pages 193–206, Monterey, CA, 1997. http://www.cs.wisc.edu/~cao/papers/gd-size.html.

[2] D. Caromel, W. Klauser, and J. Vayssière. Towards seamless computing and metacomputing in java. *Concurrency: Practice and Experience*, 10(11–13):1043–1061, 1998. http://citeseer.nj.nec.com/article/caromel98towards.html.

[3] Cascading Style Shett. http://www.w3.org/style/css/, 2003.

[4] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A Hierarchical Internet Object Cache. In *Proc. of the 1996 USENIX Annual Technical Conference*, pages 153–164, 1996. http://www.usenix.org/publications/library/proceedings/sd96/danzig.html.

[5] Document Object Model (DOM). http://www.w3.org/dom, 2003.

[6] Edge Side Includes. http://www.esi.org/, 2003.

[7] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proc. of the European Conference on Object-Oriented Programming*, pages 220–242. Springer-Verlag, 1997. LNCS 1241.

[8] I. Kiselev. *Aspect-Oriented Programming with AspectJ*. Sams Publishing, 2002.

[9] Microsoft Corporation. ASP.NET. http://asp.net.

[10] P. Mohapatra and H. Chen. WebGraph: A Framework for Managing and Improving Performance of Dynamic Web Content. *IEEE Journal of Selected Areas in Communications*, 24(7):1414–1425, September 2002. Special Issue of Internet Proxy Servers. http://www.cs.ucdavis.edu/~prasant/pubs/journal/webgraph.ps.

[11] Oracle Corporation and Akamai Technologies Inc. Edge Side Includes (ESI) Overview, May 2001. http://www.akamai.com/en/resources/pdf/esioverview.pdf.

[12] M. Philippsen and M. Zenger. Javaparty – transparent remote objects in java. *Concurrency: Practice and Experience*, 9(11):1225–1242, 1997. http://citeseer.nj.nec.com/philippsen97javaparty.html.

[13] Sun Microsystems, Inc. The Java Technology. http://java.sun.com.

[14] Viewable with Any Browser. Web site, 2003. http://www.anybrowser.org/campaign/.