A Composable Framework for Secure Multi-Modal Access to Internet Services from Post-PC Devices

Steven J. Ross, Jason L. Hill, Michael Y. Chen, Anthony D. Joseph, David E. Culler, and Eric A. Brewer

Computer Science Department, University of California - Berkeley

E-mail: {stevross, jhill, mikechen, adj, culler, brewer}@cs.berkeley.edu

Abstract

The Post-PC revolution is bringing information access to a wide-range of devices beyond the desktop, such as public kiosks, and mobile devices like cellular telephones, PDAs, and voice based vehicle telematics. However, existing deployed Internet services are geared toward the secure rich interface of private desktop computers. We propose the use of an infrastructure-based secure proxy architecture to bridge the gap between the capabilities of Post-PC devices and the requirements of Internet services. By combining generic content and security transformation functions with service-specific rules, the architecture decouples device capabilities from service requirements and simplifies the addition of new devices and services. Security and protocol specifics are abstracted into re-usable components. Additionally, the architecture offers the novel ability to deal with untrusted public Internet access points by providing fine-grain control over the content and functionality exposed to the end device, as well as support for using trusted and untrusted devices in tandem. Adding support for a deployed Internet service requires a few hundred lines of scraping scripts. Similarly, adding support for a new device requires a few hundred lines of stylesheets for the device format. The average latency added by proxy transformations is around three seconds in our unoptimized Java implementation.

1 Introduction

Internet services for electronic communication and commerce have permeated our society. People routinely access and exchange private and sensitive information when they use Internet services, such as electronic mail, the World-Wide Web, electronic banking, on-line shopping, E-commerce, and stock trading. At the same time, we are moving into a "Post-PC" era, where people use many different types of devices to access these Internet services, including pay-per-use public Internet kiosks, laptops, PDAs, cellular telephones, and two-way pagers, each with different characteristics and capabilities. Three critical requirements for users of Internet services in these Post-PC environments are *secure access to information, dynamic content adaptation,* and the *fusion of multiple devices*. The *fusion of multiple devices* represents an extension of the split trust model [5] where multiple devices work in tandem. Meeting these requirements will preserve the privacy and integrity of users' transactions and enable the use of new and interesting services and access devices.

Secure, wide-area access to information is of paramount importance on the Internet, but existing solutions do not address Post-PC requirements (e.g., traffic can be intercepted and user identities can be forged). Protection against such attacks is provided by strong authentication and end-to-end encryption. A user's *trusted* application establishes a secure connection to the service of interest. For example, when accessing a secure web-based banking service, the user's web browser establishes a secure connection to the web server by using HTTP over Secure Socket Layer (SSL) [24].

The existing security model makes two basic assumptions: first, that both the user's access device and the software running on it can be trusted not to intercept or send private information elsewhere; second, the access device has the computational resources to secure the connection (e.g., to perform public key SSL operations) and to display the service's content (e.g., to render complex color graphics). While these assumptions are reasonable for users of private desktop computers, they do not hold for Post-PC devices, where users access services from both untrusted devices (such as public kiosks), and devices with limited resources (such as PDAs).

Untrusted kiosks are problematic for secure Internet services, as all service data is available in unencrypted form to the kiosk. Possible attacks may be as straightforward as recording all keystrokes (e.g., Personal Identification Numbers, passwords, and login information), or as subtle as recording users' personal information for later fraudulent use (e.g., account numbers). Additionally, a kiosk can "hijack" a secure connection and perform active attacks (e.g., make bank transfers or send forged e-mails). Untrusted endpoints should not be allowed to see a user's personal information; *instead, the content value of such data must be reduced.* Likewise, access to sensitive service functionality from these endpoints must also be guarded.

PDAs are also problematic because they are generally low-power, computationally-limited devices with limited memory and networking capabilities. To perform the industry-standard SSL handshake phase on one such device, a 3Com Palm Pilot V [35] requires several seconds. This latency imposes an intolerable delay for connection setup, which is particularly undesirable if network connectivity is intermittent. An SSL implementation that uses elliptic curve cryptography [9] is feasible on a Palm Pilot V, but few Internet services support that option. Moreover, even if a PDA is capable of performing SSL, one may still opt for a protocol that is faster and more power-efficient. Finally, entering data to fill out forms using a pen-based interface is tedious at best and even more cumbersome using number pads on devices such as cellular telephones.

Demand for continuous access to Internet content is increasing and Internet services are becoming available in new environments such as automobiles [28] and kiosks in airplane seats [27]. These environments demand multimodal access to content either through traditional HTML, PDA thin client browsers, WML enabled phones, and voice.

We propose the use of a trusted infrastructure-based proxy service to provide secure multi-modal access to Internet service content from any device. An advantage of this proxy-based approach is the layer of indirection it provides which allows transparent support for widely deployed systems owned and operated by others. For untrusted terminals, sensitive user information is removed from the information stream going to the terminal. For example, real names and account numbers can be screened out. The proxy also mediates access to the service by filtering control data in order to limit allowable actions (e.g., disallowing all stock trades on untrusted kiosks). For access from computationally-limited devices, the proxy service transcodes the security protocol into one which is more efficiently executed on the device. Additionally, the proxy can distill the service content into a format more suitable for the device [16]. For example, a HTML service can be rendered as WML for a WAP enable phone, or even as voice.

A drawback of the proxy approach is that it requires users to place a significant amount of trust in the proxy infrastructure. The components in the infrastructure require access to all data flowing through them to perform filtering and adaptation transformations. Additionally, users are required to store private information and preferences in the infrastructure to enable these operations. This trust can be mediated by colocating the proxy with a particular Internet service in which the user already has established trust of sensitive information. For example, an Internet e-mail or stock-trading service can provide its own proxy service for access from mobile devices and public terminals. Alternately, the proxies can be treated as orthogonal to the services for which they enable access; a user need only establish an account with a secure multi-modal proxy provider, much as they do with an Internet service provider today.

In addition to enabling basic access, our security proxy can be used to combine the capabilities of both untrusted public terminals and mobile personal devices. For example, the limited GUI of a PDA can be supplemented by the richer, larger display of a public kiosk, while the PDA is used only for sensitive operations. The security proxy splits trust between the PDA and the public terminal by fusing the devices together to provide one logical channel with secure access to the end service. Consider the case of users accessing their stock trading accounts from public access terminals. Instead of relying on the terminal to protect their secure information, the users could direct private or sensitive information (e.g., portfolio value and account information) to their portable device, while using the GUI capabilities of the public terminal to initiate requests and display generic stock information (e.g., stock price fluctuations and historical graphs). The users initiate trading operations through the untrusted public terminals and then confirm them using their trusted portable devices. The connections to the mobile devices can be provided by the environment (e.g., kiosks with infrared network connections for PDAs) or by the devices (e.g., receiving a call on a cell-phone, or a message on a two-way pager). The proxy allows data to be encrypted with a protocol more appropriate for the capabilities of computationally limited and power constrained small devices such as elliptical curve or shared key techniques. This capability bridges the gap to existing services that do not currently provide this option for smaller devices.

We present a secure proxy architecture that:

- Protects users from untrusted access points.
- Enables users to use untrusted access points in tandem with trusted mobile devices for security-enhanced service interactions.
- Allows users to use the access device of *their* choice regardless of the device's computational abilities.
- Simplifies the tailoring a service to multiple device formats to simple content authoring of style sheets and scraping scripts.

Our secure proxy incorporates a component-oriented, rule-based architecture which minimizes the amount of service-specific code that must be written. In particular, service developers (and users) can specify security rules that make the appropriate modifications to the data and control actions flowing in both directions between the user and the service. Adding support for new services only requires authoring a XML representation of the semantic content and a WebL script to extract the content from the service. Adding support for new devices is done simply through XSL stylesheets that render the content to the devices' format.

2 Architecture

In this section, we present an overview of our architecture for an infrastructure-based proxy for secure multimodal access to Internet service content, followed by a detailed design discussion of each component.

2.1 Design overview

Our secure proxy for multi-modal access to Internet services consists of a few building block components and the canonical path between them. The proxy provides a secure level of indirection that can be used to modify content flowing between clients and services. It is a component-based system for creating customizable building blocks that control and modify users' connections to secure services. These components can easily be composed together to bridge the gap between traditional services and Post-PC devices that have different trust levels and to simplify the process of adding support for new services and devices. This model provides a partitioning of computation that allows the components to be placed in a scalable, fault-tolerant, and highly-available execution environment such as [20] that meets the critical requirements of Internet-scale services.

Our architecture allows the user to take on different *trust profiles* based upon the location, device, capabilities, or trust in an access device. For example, these profiles include connecting from home devices (fully trusted), work computers (partially trusted), public terminals (not trusted), PDAs (trusted, but not powerful), or with a untrusted device in tandem with a trusted one (split-trust).

By dividing the functionality of our architecture into structural and semantic transformations, it is relatively easy to add new services and devices. *Format transformations* adapt content to device capabilities (e.g., changing the resolution, color depth, or encoding format of an image), while *semantic transformations* protect users' data and prevent various actions from being performed from untrusted end devices. Additionally, the use of rule-based components to add service-specific semantics to generic transformations simplifies the incorporation of new services.

The architecture consists of the following components:

- Security Adaptors (SA): These components bridge the gap between the secure access protocol required by a service (server-side) and access devices (client-side).

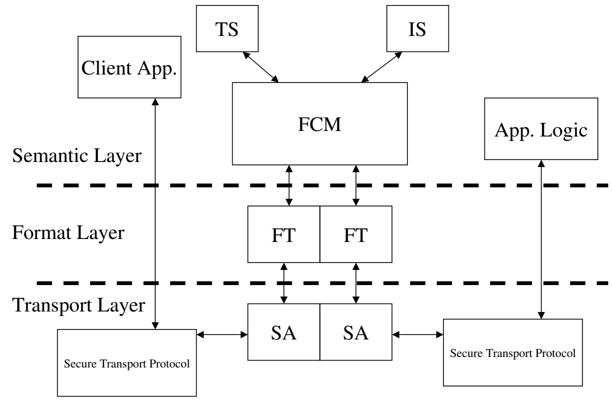

Figure 1. Secure proxy architecture

- Format Transcoders (FT): These components perform structural transformations to convert data flowing to and from services or devices into a representation format that the FCM component can operate upon.

- Filter and Control Modifier (FCM): A rule-driven engine that performs semantic transformations to reduce the content value of data and restrict control actions.

- Identity Service (IS): The IS is a secure central repository that stores persistent state on behalf of users. The state includes identities for accessing services and rules for displaying and modifying service content based upon the users' role and mode of access. Additionally, the IS stores identifying information for Internet services. This information consist of scripts for scraping the content from the service, and style sheets for rendering the content to device formats.

- Transient Store (TS): The TS stores temporary state, such as current session identifiers, mappings for obfuscated or hidden data, Uniform Resource Locators (URLs), and client request information.

Figure 1 illustrates the architecture of our proxy. Each component unifies a collection of devices or protocols into one format, isolating service-specific details from the logical operations required to transform data. This abstraction capability becomes increasingly important as the number of devices and services grows. A new device or service can be enabled by simply creating a few device-specific components, easing composability versus modern vertical web-based applications that require service authors to individually tailor services to each end device. Our proxy simplifies the delivery of content to multiple devices by: reducing it to the task of authoring style sheets to render content, and delivering the content through building block components that are implemented once to support a specified security or Internet protocol. Without these abstractions, authoring is an $N \times M$ task, where the introduction of a new device type requires changes to every service to add support for the device and vice versa (e.g., web-clipping [1]).

2.2 High-level Walk through

In this section, we illustrate the components of the framework by exploring the scenario of a user at an airport who wants to buy stock using an Internet-based stock trading service from an untrusted kiosk. The user would like to use the kiosk for filtered graphical data display (i.e., sensitive account information should not be displayed), and would like to confirm stock purchases using her trusted PDA. The kiosk provides Internet access to the PDA over an infrared connection.

We use this example as motivation for the component design decisions that are required to raise the bar high enough such that users feel comfortable with the security associated with their transactions. We choose stock trading as it most intuitively stresses each of the components in the architecture. We feel that the architecture applies equally well to other web based sites for electronic commerce, i.e., purchasing books or on-line banking. The architecture can also provide secure access to other Internet services such as directory services (LDAP), or e-mail (IMAP). In addition, we present further design decisions that support these scenarios.

At a high level, the flow and operation of the components are as follows: the user first authenticates herself to the system using incoming Security Adaptors both from the untrusted kiosk and the trusted PDA. The outgoing Security Adaptor establishes an SSL session to the stock trading service, both authenticating the service to the user and authenticating the user to the service.

Data sent from the stock trading service to the user flows through the proxy infrastructure as follows. The stock trading service dynamically generates a page and sends it over an SSL connection into an infrastructure SSL Security Adaptor (1). The SSL Security Adaptor passes the web page's HTML to the server-side Format Transcoder (2), which looks up the scraping script from the Identity Service(3) and extracts the semantic content and returns it as XML. The XML representation is passed on to the FCM (4). The FCM looks up the rules for the page based on the user's current profile (5), in this case the split-trust profile. The FCM applies the content filtering rules to the data in XML format, storing any obscured state and mappings in the Transient Store (6). Once the FCM has processed all the rules associated with the current page, it passes the modified XML to the client-side Format Transcoder (7), which looks up the XSL style sheet from the Identity Service(8) and renders the content as HTML for display on the kiosk (or WML for display on a PDA). Finally, the rendered content is passed to the client side SSL Security Adaptor (9), which sends it securely to the kiosk (10) over SSL (or another secure channel for the PDA).

Data sent from the user to the stock trading service flows through the proxy components as follows. The user submits a HTTP GET or POST request from a client device (1). The incoming Security Adaptor takes the data from the established secure connection and passes it on to the incoming client side Format Transcoder (2), which transforms the request into an XML representation and then passes it to the FCM (3). The FCM retrieves the rules associated with the user's current role from the Identity Service (4) and uses them to process control requests. During rule application, mappings are retrieved from the Transient Store (5). The modified XML representation of the request is passed to the out-going server-side Format Transcoder (6), which creates an HTTP GET or POST request from the XML representation. The request is passed to the server-side Security Adaptor (7), which sends it securely to the service (8) over SSL.

The path for a request that must be authorized by the users trusted device is as follows. Steps 1 through 5 of generic request processing from the kiosk or client device are the same as previously described. Additionally, the user uses the PDA's browser to request the authorization information page (6). The FCM generates XML that is rendered appropriately by the FT's for authorization on the PDA (7). In this example, the page displays verification data (stock name, price, etc.) and form buttons for authorizing and declining the request. The user then verifies that the information matches that entered on the untrusted endpoint. If it is acceptable, the user sends a message (8) authorizing the request and receives the response (9) on the PDA. Finally, the FCM releases the request, which flows through the Format Transcoder, Security Adaptor, and to the stock trading service as previously described (10, 11, 12).

The following sections describe each of the framework components in detail.

2.3 Security Adaptors (SA)

Security Adaptors allow devices capable of performing one security protocol to access services that require a different protocol. Consider a mobile user with a low-power device capable of shared-key authentication and Blowfish [36] encryption, but not capable of performing the complex computations required by SSL [24]. However, an existing internet service may only support authenticated access over SSL. In this case, the infrastucture provides a client-side Security Adaptor that uses a shared-key protocol, obtaining a user's key from the Identity Service and using the key to perform challenge-response authentication and to encrypt all communication with the client. The server side Security Adaptor communicates with the end service over SSL. Security Adaptors must be trusted as they have access to unencrypted user and service data and authenticate users, services, and devices to the infrastructure.

When connecting from an untrusted public key encryption-capable device, users authenticate themselves to the infrastructure using a pseudonym identity that identifies the trust profile and access device, and a one-time use password mechanism (e.g., S/KEY or SecurID [31, 19, 13]). This information does not need to be hidden from the untrusted endpoint as it can only be used once.

Additionally, the client-side Security Adaptor can provide persistent connections, even if the service does not provide them, enabling a client to establish a secure connection to the infrastructure once, while the infrastructure (potentially) establishes multiple connections to the service. If the connection setup time or round trip latency is high, this technique will yield a substantial performance improvement. Also, the infrastructure can establish multiple connections to different services to aggregate information (e.g., to perform a meta-search of multiple web sites).

2.4 Format Transcoders (FT)

Format Transcoders (FTs) transform data between external service and device formats and the FCM's XML-based language, abstracting service- and device-specific information from the FCM. There are two types of Format Transcoders: server-side FTs and client-side FTs.

2.4.1 Server-Side Format Transcoders

The server-side Format Transcoder receives a service's content and transcodes it into an XML [8] representation of the semantic data. We choose XML as it is the industry standard technology for semantic mark up. Additionally, the XSL [15] presentation standard, which separates content from presentation, allows us to render the XML representation to any device format. Our architecture allows content authors to choose the XML representation of their choice, enabling authors to provide multi-modal access to services without waiting for service providers to agree on a standard XML DTD for their content.

The proxy extracts a service's semantic representation using "screen scraping"[1]. Scraping is a service-specific operation, so each site requires its own content extraction scripts. The server-side Format Trascoder is a generic execution environment for these scripts.

By abstracting the logic for scraping service content into scripts, we create a single place where all service-specific code is located and enable the FT component to be generically reused across multiple services. By semantically tagging data the FCM can apply rules, based on the user's current trust profile, to these tags and appropriately modify the data.

The author of the server-side Format Transcoder scraping script must identify the semantic content of displayed data and the actions that can be performed by a service. For web-based services, this is a substantial task that requires manual analysis. Tools such as WebL [12] help to automate this task. Products such as Epicentric portal server [14] and Cohera [10] include web-scraping building blocks.

When operating on data flowing towards a service, the format transformation merely transforms the XML representations of control actions (e.g., HTTP POST or GET requests for web pages, or RMI for Java based services) into the services' target format. The code to perform these operations is reusable across multiple services.

2.4.2 Client-Side Format Transcoders

Client-side Format Transcoders perform transformations on data flowing to and from the client into and out of the FCM's XML representation. As data travels to a service, HTTP POST and GET requests, RMI calls, etc. are transformed by the client-side Format Transcoder into an XML representation that can be operated on by the FCM to apply control filtering rules.

Consider, for example, the buy request that a user would execute in the stock trading scenario. The original HTTP POST request is translated into a representation where each form item is given a semantic XML tag. Additionally, the intermediate XML format allows HTTP based clients to access RMI based services and vice versa.

For data traveling to the client, the client-side Format Transcoder uses XSL style sheets to render the the FCM-modified XML for the particular end device. Using XSL gives content authors a standard tool to tailor the presentation to the device format.For the stock trading example, the format for the kiosk is HTML, while the format for the PDA could be WML, a thin client browser format, or another custom format.

2.5 Filter and Control Modifier (FCM)

The FCM implements application-level logic for adapting data flowing between devices and services to the desired security parameters by applying the rules associated with users' current trust profiles to perform content and control filtering on the XML representation of the semantic content. The FCM also applies rules for split-trust control functionality, where a trusted portable device is used to authorize and authenticate actions from an untrusted device.

The following sections describe the types of rules used by the FCM and their actions. Figures 2, 3 and 4 summarize these rules.

[1] Extracting information from a web page's HTML code.

rule name	action	example
allow-content	Explicitly state that content is allowed to be displayed	allow-content <AccountValue>
hide	Replace content with obscuring text	hide <UserName>
obfuscate -well-known	Replace with well-known obscuring text	obfuscate-well-known <Account-Number> "Checking Acc. Num."
obfuscate -mapping	Replace with replacement text generated from set-of-mapping-data, and store mapping in Transient Store	obfuscate-mapping <StockSymbol> US-State-Abbreviations-Set
obfuscate -form	Replace all form field names with randomly generated mappings, and store mapping in Transient Store	obfuscate-form <StockOrderCgi> e.g. before and after are: <select name="buysell"> <select name=34382>
obfuscate -cookies	Replace cookies with randomly generated cookies storing mapping in Transient Store	obfuscate-cookies <DatekOnLine>
verify	Send content to trusted device for verification	verify <StockQuote>

Figure 2. Content filtering rules used by the FCM

2.5.1 Using content filters to obfuscate data

Infrastructure content filters alter or remove sensitive service content before it reaches the untrusted device, effectively decreasing the data's privacy or security level.

The FCM applies the user's rules by matching each XML tag to the appropriate rule, and performing the specified action. In order for a data item on the page to be displayed, the security level at which it can be rendered must be explicitly specified by the user's trust profile. This requirement follows the principle of least privilege.

There are six types of rules for data value reduction: *allow-content*, *hide*, *obfuscate-well-known*, *obfuscate-mapping*, *obfuscate-form*, and *obfuscate-cookie*. The *allow-content* rule explicitly states that a content item is allowed to be rendered on the page in its current format. This rule prevents service format changes from causing secure information to be accidently released. The *hide* rule replaces sensitive information in the content with an uncorrelated number of "|" characters, within a reasonable limit of the original size of the data. The *obfuscate-well-known* rule replaces well-known sensitive information with its description rather than its value. For example, rather than displaying the user's home address "777 Main Street", the text "HOME ADDRESS" is displayed. This information is meaningful to the end user, but leaks no useful information to the kiosk.

The *obfuscate-mapping* rule produces a random table of mappings, stores the table in the Transient Store, and replaces sensitive data with an obscured name from the table. If a trusted device is being used in tandem with an untrusted one, the FCM sends the mapping table to the user's trusted device, so the user can make sense of the obfuscated data. In the stock trading example, to prevent the kiosk from obtaining the list of stocks contained in a portfolio, the FCM applies the *obfuscate-mapping* rule to the portfolio data. For example, the result could be a table that maps stock names to the names of states (e.g., "IBM" maps to "Kentucky", "AOL" to "Michigan", etc.).

rule name	action	example
replace -well-known	Replace well-known place-holder value with real value stored in user profile rule	replace-well-known <value> *MyVisa* 1234 e.g., request before and after are: <value>*MyVisa*</value> <value>1234</value>
replace -mapping	Replace obfuscated value looking up mapping from Transient Store	replace-mapping <StockSymbol> e.g., request before and after are: <StockSymbol>MI</StockSymbol> <StockSymbol>AOL</StockSymbol>
replace -form	Replace obfuscated request data with real data from Transient Store	replace-form <StockOrderCgiName>
replace -cookies	Replace obfuscated cookies	replace-cookies <DatekOnLine>

Figure 3. Content rewriting rules used by the FCM

A rule that is similar to the *obfuscate-mapping* rule is the *verify* rule, which leverages the trusted mobile device to verify content displayed on the untrusted endpoint. The *verify* rule can be used to make sure that the untrusted device does not modify useful public information in a malicious manner. For example, the price of an item on an on-line shopping web site is of dubious value if it can be modified by an untrusted endpoint. The *verify* rule uses the portable trusted device to display small amounts of data, thereby allowing the user to determine whether the untrusted kiosk has tampered with the data. The *verify* rule is particularly attractive for mobile devices with cumbersome interfaces as it allows the majority of the interaction to occur on the rich interface of the untrusted public device.

The *obfuscate-form* rule removes sensitive information encoded in form fields and hyper-links by generating temporary obfuscated names for form action names, form input fields, and hidden fields for state. Mappings between obfuscated values and actual data are held in the Transient Store. The rule also modifies hyper-links and Java Script in HTML pages containing sensitive session or user data. These transformations are necessary to prevent the leakage of sensitive data to kiosks, as well as to prevent kiosks from being able to post form requests directly to the end service. The *obfuscate-cookie* rule performs a similar transformation for any cookies that may be sent to the untrusted endpoint.

2.5.2 Using content rewriting to reduce the entry of sensitive information

Just as we do not trust endpoints to receive sensitive data, we do not want users to enter sensitive information into untrusted kiosks. Thus, the FCM uses replacement rules to exchange placeholder information with sensitive information that is stored only in the trusted infrastructure. There are four rules for content rewriting: *replace-well-known*, *replace-mapping*, *replace-form*, and *replace-cookie*.

The *replace-well-known* rule replaces well-known identifiers with the actual data, such as "CREDIT CARD", with the user's credit card number, and "HOME ADDRESS"

rule name	action	example
allow-control	Explicitly allow control requests for POSTs/GETs	allow-control `<action>` "http://finance.yahoo.com/q"
authorize	Request and wait for out-of-band authorization of control requests from trusted mobile device	authorize `<action>` "http://mail.school.edu/send"

Figure 4. Control filtering rules used by the FCM

with the user's real home address. This type of form replacement is useful for supporting e-commerce purchase requests that need account numbers and shipping addresses that can't be entered on untrusted clients. This functionality is also useful for trusted clients as it simplifies the process of filling out forms.

The *replace-mapping* rules replace obfuscated values with mapped values using information stored in the Transient Store by *obfuscate-mapping* rules (e.g., a user selling a stock enters a U.S. state abbreviation rather than the stock symbol, and the FCM performs the reverse mapping).

The *replace-form* rule is used to replace obfuscated form action names, field names, and hidden text in HTTP GET/POST requests using mappings previously stored in the Transient Store. The *replace-cookies* rule works similarly for cookies.

2.5.3 Using control filtering to protect service functionality

Many services provide information with varying degrees of sensitivity (e.g., looking up a stock quote is much less sensitive than trading a stock), however, they provide all or nothing access to service functionality, as they assume that the endpoint is trusted. The proxy infrastructure supports access from untrusted endpoints by providing fine-grain access control over service functionality. The FCM provides this capability using control filtering rules that control data and commands flowing from the client to the server.

The FCM uses two rules for control filtering: *allow-control* and *authorize*. The first rule states whether or not a request to submit a form or view a page can be transmitted to the end service. In the stock trading example, the control functions on the service's forms and pages (even in the split-trust case) must be explicitly allowed by the profile.

The *authorize* rule provides fine-grain control over sensitive actions, such as buying stock from an untrusted endpoint by using an a trusted device to explicitly authorize the request. In the stock trading example, a request to trade stocks is intercepted and held by the FCM until authorization is received from the user's trusted companion device (e.g., a PDA, cellular telephone, or two-way pager). If a user wants actions to be authorized through a device, they need to first register the authorization device with the infrastructure. Then, when an action is performed that requires confirmation, the FCM will intercept and hold it, and contact the user via the specified device. The trade action is then sent to the device and confirmation is requested. Once it is authorized, the FCM releases the request.

2.6 Identity Service (IS)

The Identity Service is the secure repository of persistent data associated with users' *trust profiles* and their security preferences for access to services. The Identity Service stores the following types of information:

- Identities and credentials (e.g., usernames, passwords, encryption keys, etc.) for accessing secure services.
- Rules and preferences for content and control filtering.
- Personal information (e.g., addresses and account numbers) for automated form filling from untrusted devices.
- Pseudonym identities and one-time passwords.
- Service information such as content scraping scripts and XSL style-sheets.

The primary motivation for the Identity Service derives from the need to hide private data from untrusted endpoints, where there is no way for a user to securely transmit private information. The Identity Service provides an alternative where users' passwords and other private information are stored in the infrastructure and merged with any transmission after the data has left the untrusted device and entered the trusted infrastructure. One could imagine a world in which identity services become a common piece of the public Internet infrastructure. However, until existing Internet services migrate to adopt this functionality our proxy can transparently provide it.

The Identity Service provides additional benefits for users accessing secure services from trusted endpoints. The replication of security credentials on several devices, some of which may easily be stolen (e.g., small PDAs), increases the probability that credentials may be compromised. It is also unlikely that most users will be savvy enough to understand the security capabilities of several devices, operating systems, and file systems. Similarly, changes to a credential require that it be updated on all devices. The IS solves these problems by providing a single source for such data.

Obviously, the security model of the Identity Service must be carefully considered and it must be well-protected against attacks. We argue that storing the data in one professionally-administered, physically-secured Identity Service is more secure than having it managed across several physically insecure devices by naive end users.

Users can control access to their private information in the Identity Service as each of the data entries has an associated Access Control List (ACL).

2.7 Transient Store (TS)

The Transient Store is used to store soft state associated with the current session of the user, such as mappings of obfuscated to original URLs, pending requests for trusted device authorization, or session information. Data can be stored using index keys that are generated by applying a secure hash function (e.g., SHA1 [37]). Secure access can be provided by using either a large key space such that a key can never be guessed or by performing access control.

3 Implementation

The design we propose in this paper is a generalization of the ideas we gained from an initial proof of concept implementation of the system. The proof of concept implementation provided access to the Datek Online Stock Trading service for users of untrusted kiosks and computationally limited PDAs. The lessons learned in the proof of concept implementation were valuable and led to the design decisions for the current implementation, based upon the Ninja vSpace Platform.

The vSpace platform is version two of the Ninja [25] infrastructure for building scalable, highly available, fault tolerant, cluster based applications. Among the most notable features of vSpace are asynchronous RPC, distributed data structures, and an event driven programming model. Additionally, the vSpace execution environment provides support for load-balancing, and fail-over.

Our proxy service is implemented as a pipeline of ten vSpace Workers composed through the Task/Completion asynchronous Ninja RPC mechanism The vSpace platform is essential for high concurrency performance of the composable pipeline. In thread based RMI environments, blocking interfaces break down as threads are tied up on the callers. For example, when object A remotely calls object B that remotely calls object C the thread on A has to wait until the call to C finishes. In the asynchronous task based vSpace environment [40], the thread becomes freed immediately after object A remotely calls object B. The vSpace platform supports high concurrency with fewer threads and hence less computational overhead do to thread overhead and context switching.

3.1 vSpace Worker Architecture

In the current vSpace model, each vSpace worker is given one thread of control. Parallelism is achieved by asynchronous processing of events by this thread. Each worker consists of a queue that receives incoming events that the worker registers for reception with the environment on startup. Events are placed on the queue by the infrastructure platform usually as tasks from the RPC mechanism, or lower level events from the asynchronous I/O libraries. Ninja RPC tasks are message objects containing data and methods for accessing and operating on the data. The worker thread removes events from the queue for processing.

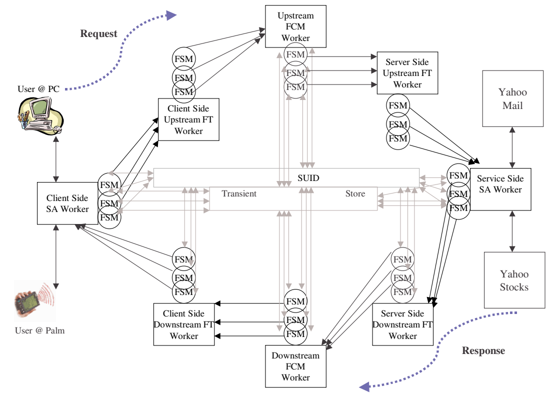

Figure 5. Transformation pipeline

For processing events that require long blocking calls such as reading from the network, RPC (dispatching tasks, and waiting for completions), or disk IO, the worker must explicitly manage state. A useful abstraction for managing this concurrency consists of creating a new finite state machine (FSM) object that manages the state associated with the computation required.

The FSM object is the equivalent of a thread context in the multi-threaded programming model. It consists of local variables for the current state of processing, and implements the UpcallHandlerInterface to receive the results of blocking operations. When this object completes the necessary processing steps it dispatches its results to the next worker in the system as a task using the Ninja asynchronous RPC mechanism.

All of the workers in our system follow this general model of processing. The overall composition of workers leads to a transformation pipeline for communication between the client device and end service as depicted in figure 5. The messages to the Identity Service and Transient Store are greyed out in the figure to emphasize the transformation pipeline from the client to the Internet service.

3.2 Current worker implementations

In the current implementation, Client Side Security Adaptors (CSSA) listen on well known ports using asynchronous TCP server sockets[2] modified to support security protocols. Currently, SSL is supported, using Sun's JSSE library, as well as a shared key protocol based on Blowfish. SSSA session objects handle HTTP redirections directly to avoid costly round trips through the potentially thin network to the client device. SSSA Session objects also support cookies, storing them in the Transient Store.

The Format Transcoder implementation consists of four vSpace workers supporting HTTP based services. The Client Side Upstream Format Transcoder (CSUFT) uses a simple parser to transform HTTP requests into an XML representation. The Service Side Upstream Format Transcoder (SSUFT) uses Sun's SAXP XML parser to transform the XML representation into the appropriate HTTP request for accessing the end service. The Service Side Downstream Format Transcoder (SSDFT) uses WebL [12] to scrape the content from HTML pages and return it in an semantic XML representation. A drawback in this implementation is that a new WebL process is exec'd for each page. The Client Side Downstream Format Transcoder (CSDFT) uses Apache's Xalan [3] XSL renderer.

The Filter and Control Modifier workers use Sun's SAXP XML parser and apply rules in one pass. Currently, the following subset of rules is implemented: hide, allow-content, allow-control, and replace-well-known. Rules are specified per profile and can be set by either a text file or through the JSP based UI.

The Identity Service (SUID) is implemented as an API layer on top of a distributed hash-table [21, 26] and works across a cluster. The SUID Worker provides a Ninja RPC interface for the specific tasks required by the proxy service to manage user profile information, service transcoding rules, and UI functionality. Since the state transitions in the SUID are relatively simple, rather than creating a new FSM object for each request, it manages requests internally. The Transient Store is implemented in a similar fashion.

3.3 Discussion

Our initial design changed little during the implementation phase. The most notable change is using the WebL scripting language to create a generic Server Side Downstream FT component, a capability that requires the storage of service-specific information, such as WebL scraping scripts and XSL style sheets, in the Identity Service. A future change would be to separate out the XSL rendering and web scraping functionality into individual workers, rather than having the functionality embedded in the FT worker state machines. The further decomposition will create useful building blocks for other Ninja services. It will also make it easy to replace the workers with alternative web scraping and XSL tools.

[2] provided by the vSpace platform

Symbol	Last Trade		Chg	Volume	Shrs	Value	Value Change		Paid	Gain		More Info
$$CASH					75,916.25	$75,916.25	-	0.00%	-	-	-	N/A
MSFT	Sep 22	**63 1/4**	-15/16	42,238,300	100	$6,325.00	-$93.75	-1.46%	62.3125	$83.75	+1.34%	Chart , News , Msgs , Profile Research , Insider , Options
AOL	Sep 22	**55 1/4**	+1 3/4	10,005,800	100	$5,525.00	$175.00	+3.27%	54.3125	$83.75	+1.54%	Chart , News , Msgs , Profile Research , Insider , Options
MSFT	Sep 22	**63 1/4**	-15/16	42,238,300	100	$6,325.00	-$93.75	-1.46%	62.3125	$83.75	+1.34%	Chart , News , Msgs , Profile Research , Insider , Options
INKT	Sep 22	**125**	-15/16	1,292,200	50	$6,250.00	-$46.88	-0.74%	123	$90.00	+1.46%	Chart , News , Msgs , Profile Research , Insider , Options
5 symbols					**Totals(USD):**	**$100,341.25**	-$59.38	-0.06%		**$341.25**	+1.42%	

Figure 6. Yahoo Contest holdings

Another interesting lesson learned in the current implementation is the need to provide pluggable protocol parsing in the security adaptors. We found the CSSAWapSession to be a useful place to do pre-processing of parsing of headers for protocols and gathering data packets.

Overall, the current implementation consists of approximately 26,000 lines of commented Java source. The management user interface is 2,100 lines of JSP code. The scripts written to provide access to representative internet services Yahoo Contest and Yahoo Mail are 1,400 and 1,600 lines of XSL and WebL code respectively.

4 Analysis

In this section, we evaluate the effectiveness of our proxy-based approach to secure multi-modal access to Internet content from post-pc devices. The evaluation criteria are: ease of adding support for new services, ease of supporting new client formats, and overall performance of the system. The analysis is based on experience with a sample service we wrote that provides secure access to Yahoo Contest [22], a stock trading simulator.

4.1 Adding new services

Adding support for a new service merely requires writing a site map, XML representations and WebL scripts for each of the services pages, and a default rule set. Our implementation of proxied multi-modal access to Yahoo Contest service consists of five content pages: holdings, orderConfirmation, orderForm, orderVerification, and quotes. We also use a shared page that we include in each of these pages. The site map consists of the service name and unique page identifiers for each page.

A service author creates XML representations and WebL scripts for each page and stores the scripts in the SUID. In this example, the holdings page requires the most scraping and consists of a 250 line WebL script. Figure 6 displays the original Yahoo Contest holdings page. The following is an example XML representation for the data on the holdings page.

```
<StockSymbol>        MSFT          </StockSymbol>
<TimeOfPrice>        Sep 22        </TimeOfPrice>
<Price>              63 1/4        </Price>
<Change>             -25/16        </Change>
<Volume>             42,238,300    </Volume>
```

<table>
<tr><th>symbol</th><th colspan="2">last trade</th><th>change</th><th>volume</th><th>shares</th><th>value</th><th colspan="2">value change</th><th>paid</th><th colspan="2">gain</th><th>more info</th></tr>
<tr><td colspan="5">$$CASH</td><td>||||||||</td><td>||||||||</td><td>-</td><td>0.00%</td><td>|</td><td>-</td><td>-</td><td>N/A</td></tr>
<tr><td>MSFT</td><td>Sep 22</td><td>63 1/4</td><td>-15/16</td><td>42,238,300</td><td>|||</td><td>||||||||</td><td>-$93.75</td><td>-1.46%</td><td>||||||</td><td>$83.75</td><td>+1.34%</td><td>Chart , News , Msgs , Profile Research , Insider , Options</td></tr>
<tr><td>AOL</td><td>Sep 22</td><td>55 1/4</td><td>+1 3/4</td><td>10,005,800</td><td>|||</td><td>||||||||</td><td>$175.00</td><td>+3.27%</td><td>||||||</td><td>$83.75</td><td>+1.54%</td><td>Chart , News , Msgs , Profile Research , Insider , Options</td></tr>
<tr><td>MSFT</td><td>Sep 22</td><td>63 1/4</td><td>-15/16</td><td>42,238,300</td><td>|||</td><td>||||||||</td><td>-$93.75</td><td>-1.46%</td><td>||||||</td><td>$83.75</td><td>+1.34%</td><td>Chart , News , Msgs , Profile Research , Insider , Options</td></tr>
<tr><td>INKT</td><td>Sep 22</td><td>125</td><td>-15/16</td><td>1,292,200</td><td>||</td><td>||||||||</td><td>-$46.88</td><td>-0.74%</td><td>|||</td><td>$90.00</td><td>+1.46%</td><td>Chart , News , Msgs , Profile Research , Insider , Options</td></tr>
<tr><td>5 symbols Totals(USD):</td><td>||||||||</td><td>||||||||</td><td>-$59.38</td><td>-0.06%</td><td></td><td>$341.25</td><td>+1.42%</td><td colspan="5"></td></tr>
</table>

Figure 7. Yahoo Contest holdings filtered

```
<Shares>                 100              </Shares>
<Value>                  6,325.00         </Value>
<ValueChange>            -$93.75          </ValueChange>
<PercentValueChange>     -1.46%           </PercentValueChange>
<Paid>                   62.3125          </Paid>
<Gain>                   $83.75           </Gain>
<PercentGain>            +1.54%           </PercentGain>
<MoreInfo>               html omitted     </MoreInfo>
```

Writing the initial Yahoo Contest scraping scripts took approximately three weeks of part time effort by an undergraduate student unfamiliar with WebL. After gaining familiarity with WebL, authoring service scraping scripts becomes a simple task that can be done in a few days. An example WebL script can be found in the appendices.

The following example illustrates the default rule set for the tags found on the holdings page of Yahoo Contest. Only the rules for hiding data are shown as the rest of the rules for this page are allow-content.

```
hide            <Username>
hide            <Shares>
hide            <Value>
hide            <Paid>
hide            <TotalValue>
hide            <TotalPaid>
```

Additional rules are needed for the Upstream FCM to determine which control actions to allow by default in a given profile. In this case, the actions map to URLs in the service.

```
allow-control <action> "http://finance.yahoo.com/p"
allow-control <action> "http://contest.finance.../t2"
allow-control <action> "http://contest.finance.../t3"
allow-control <action> "http://finance.yahoo.com/q"
```

4.2 Adding support for new device formats

Adding support for a new client device requires writing an XSL style sheet to render the content for that device. Security adaptors need to be implemented, if they do not already exist for the desired security protocol. However, since security adaptation and content adaptation functionality is separated in our design, security adaptors can be re-used and only need to be written once per new client format.

Figure 7 shows the modified version of the holdings page after the security transformation rules have been applied. Figure 8 shows the corresponding WML version of the holdings page. The figures illustrate multiple device fusion for service access, as sensitive information hidden from the kiosk is displayed on the trusted PDA.

Two XSL style sheets are used to render the holdings page. The first generates an HTML representation for use by either a secure home machine, or a web-based kiosk. The second generates a WML representation that can be viewed on mobile devices with WML browsers. The HTML style sheet was easy to author based on the site's original HTML presentation. The WML style sheet is a simple adaptation of the HTML style sheet. For the holdings page, the HTML and WML style sheets are 370 and 100 lines respectively, plus a shared file of 100 lines. Example XSL style sheets to render the holding page in HTML and WML can be found in the appendices.

As illustrated in the examples, adding support for an additional device format only requires writing a short XSL style sheet to output the content in a suitable format for that device. This powerful capability allows content authors using the proxy to tailor the content for device screen layout and other factors. Additionally, the semantically tagged representation enables powerful transformations such as meaningful text-to-speech synthesis of the XML content.

However, these benefits come at a cost, with a trade-off versus the automated approach taken by other proxies such as ProxiNet for delivering content to PDAs. The ProxiNet approach does not require any style sheets and performs generic transformations of HTML into its custom thin client browser format. Thus, instant access to any web-site is provided to PDAs that run its browser. However, generic approaches do not offer the ability for content authors to customize pages for small devices, or do transformations such as voice rendering. To enable access to more services, we could provide the functionality of the ProxiNet approach by bypassing the semantic transformation layer. However, this would disable the fine-grain access control, content-filtering, and multiple device fusion capabilities.

4.3 Performance

Performance was measured in two configurations. The first consisted of all the workers running on a single node. The second consisted of the workers distributed across the cluster, each node running one worker.

Both of these experiments were performed on the UC-Berkeley Millennium cluster [6]. Each cluster node has two 500 MHz Intel Pentium III processors with 512KB cache, 512 MB of RAM, two 9 GB disks, and a 100 Mb/s Ethernet connection. The nodes run Red Hat Linux version 6.0 release 2.2.5. The Java environment is Sun's JDK 1.2.2 production release for Linux running with green threads and the Inprise just in time compiler[3].

For the experiments, a client used the proxy to request the holdings and quotes page 20 times each. The cold start run exhibited extremely high latencies (more than an order of magnitude higher than the average times after warmup). These results can be attributed to Java class loading and initial JIT compilation. As a result, the average times only

[3] The current Java environment only supports a single processor.

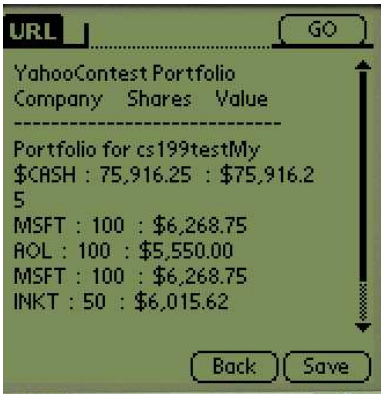

Figure 8. Yahoo Contest holdings WML

worker	Average Time(ms)	FSM Total Percent	Roundtrip Total Percent
cssa	72.94	3.03%	1.97%
csuft	280.53	11.64%	7.59%
upfcm	17.23	0.71%	0.47%
ssuft	255.55	10.60%	6.91%
sssa	465.92	19.33%	12.60%
ssdft	896.27	37.19%	24.24%
sdfcm	29.27	1.21%	0.79%
csdft	392.48	16.28%	10.61%
other	1287.52	n/a	34.82%

Figure 9. All Workers on a Single Node Performance Breakdown

include warm runs. During the experiments, the vSpace workers were run with minimal logging.

The single node configuration returned pages in an average of 3.75 seconds. The holdings and quotes pages round trip times (including authentication and redirections to the Yahoo service) averaged 4.28 s and 3.23 s respectively .

Some of the long latency is caused by inefficiencies in the current implementation. The workers exec new processes for WebL (331.3 ms) and Xalan (99.94 ms). Inefficiencies in the size and serialization of the format transcoder collection also contributes to this time due to the slow read from disk and repeated serializations. The disk writes also affected the storage components for temporary state as the DDS uses two phase commit. A large portion, `34.82%`, of the round trip time is not accounted for by these measurements. Factors that contribute to this time are task dispatch Java serialization time, queue wait time, and debug output. An optimized implementation should be able to make a substantial improvement in the latency of the proxy.

The distributed configuration performed in a similar manner. Pages were returned in 3.87 s on average. The round trip times for the holdings and quotes pages were 4.18 s and 3.55 s respectively. We expected the distributed version to outperform the single node as we observed the Java based proxy to be memory and CPU intensive during development on smaller workstations. However, the resources for

worker	Average Time(ms)	FSM Total Percent	Roundtrip Total Percent
cssa	76.51	2.76%	1.94%
csuft	314.32	11.36%	7.96%
upfcm	33.20	1.20%	0.84%
ssuft	396.84	14.34%	10.05%
sssa	497.00	17.96%	12.59%
ssdft	962.66	34.79%	24.39%
sdfcm	47.91	1.73%	1.21%
csdft	438.67	15.85%	11.11%
other	1179.96	n/a	29.89%

Figure 10. Distributed One Worker Per Node Performance Breakdown

the single millennium node proved adequate for this benchmark. The proxy was clearly I/O bound in this environment.

The distributed configuration displayed considerable tolerance to faults. Workers sometimes caused the JVM running the vSpace to crash. When the JVM was restarted, the system recovered gracefully with a higher latency for the initial class loading and compiling of the restarted worker. Future versions of vSpace, which will include automatic restart of workers, can take advantage of the properties of this configuration.

At the time of this writing, the current implementation is not stable enough to perform scalability experiments. Scalability results will be obtained after further optimization and cleanup of our proxy service, and the vSpace platform. We hypothesize the distributed configuration will have a higher throughput than the single node configuration.

4.4 Discussion

Overall, our proxy approach allows considerable customization capabilities and easy scaling to multiple devices and services. The development effort required is minimal: the simple authoring of WebL scripts for content scraping and XSL style sheets for device specific rendering. This development work can be done by an independent service provider running the proxy, and specializing in this work, rather than waiting for end services to provide support for multiple devices. Alternatively, content providers can provide content in an XML representation to bypass web-scraping and enable the content-filtering and digital wallet capabilities provided by the proxy.

In our experience, the time to author components for new protocols and style sheets for new devices is minimal. The CSSAWapSession and WML pages for YahooContest were written in a few days on top of the existing implementation. This included the time to learn WML and discover the differences between the WML browser protocol and HTTP. Additionally, the time to develop scraping scripts and a XML representation for an existing Internet service is also minimal. Support for the Yahoo Mail service was added in a few days. The user interface for customization of rules for user profiles should make it easy for users to choose a content and control filtering level they feel comfortable with from their mode of access.

The current implementation was developed on a prototype version of the Ninja vSpace platform. The majority of the development effort to get the current version running consisted of understanding the vSpace programming model and flushing out bugs in the prototype platform. Without time for optimization, the initial performance numbers of a 3 second latency are promising. Our experience with fault recovery in the distributed configuration also shows promise. Now that the platform is stabilizing, we should be able to significantly improve our performance, stability, and scalability using the understanding we gained in building the current implementation.

An open issue is the psychological acceptability of our approach. Mobile commerce is just starting to hit the public. Will mobile users be satisfied with the cumbersome interfaces of small devices or trusting of public terminals in the environment? One may argue that people aren't as concerned enough with privacy (or aware enough) as they are willing to use shopper affinity cards, revealing detailed purchase histories to supermarkets in exchange for saving a few dollars. However, we feel these same users will be sensitive and concerned when access to credit accounts and other personal information required to complete a commerce transaction are at stake. We also feel that these sensitive users will find the act of connecting to the proxy service and authenticating with one-time passwords an acceptable part of their mobile commerce routine.

5 Related work

Several related projects have focused on data filtering at different levels of granularity for increased security. Wiederhold and Billelo [41] examine the problem of protecting the release of data from databases for collaboration purposes, where the internal databases are not organized according to external access criteria. The authors note that, in these environments, traditional authentication and authorization tools and secure transmission fail to protect the release of inappropriate data. They propose a solution based upon a rule-driven security-mediator that performs result-checking to filter both queries and, more importantly, their results. Their rules for filtering query results contain traditional role-based access control, site blocking, and more advanced features, such as the replacement of query result components with non-identifying (obfuscated) terms.

Our architecture's approach to content and control filtering is based upon having specific semantic knowledge of the service data and control actions that are being filtered or modified. However, such an approach does not work for protecting unstructured information, such as the content of e-mail messages. Instead, we believe that we could apply techniques from medical research that focus on generalizing, substituting, and removing information from medical data to protect privacy without obscuring "important" details of the data [38].

Related to our idea of multiple device fusion is the notion of *splitting trust* between PCs and mobile devices to increase security [5]. In the split trust paradigm, an application's security crucial parts execute on a small trusted device, while other parts execute on a more powerful, but untrusted device. The authors explore the split trust paradigm in the context of e-mail using a Palm Pilot as a PKCS11 smart card for Netscape Communicator. Our architecture employs a similar technique in the network using out-of-band channels between the trusted and untrusted devices.

Another related project is the work by the World Wide Web Consortium on the Platform for Privacy Preferences (P3P) [18] and the associated preference exchange language – APPEL [29]. P3P provides a standard means for service providers to disclose their practices regarding the collection and dissemination of personal information and allows users to make informed decisions regarding the use of this information. This research is primarily focused on protecting the information sent from users to services, varying from click-stream data to users' personal information, such as names and addresses. We explore the inverse of this problem: allowing users to express their preferences about the type of data to be displayed and the functionality to be allowed as a function of the users' access device. Also related to this research area are trust-management systems, such as KeyNote [7], that serve as engines for applications to determine whether or not a potentially dangerous operation conforms to a user's security policy.

Our infrastructure-based proxy model is also related to research on proxies to more efficiently use network resources, reduce cost, and increase security focusing on filtering to optimize performance along the last link in mobile environments [2, 43, 44]. Similar to these works, we stress the ease of deploying an intermediary instead of changing the protocols at end points and place considerable trust in the proxy.

Related systems have examined the problem of managing different types of identity data; for example, Apple's KeyChain [4] stores all username-password pairs locally on the client PC. Lucent's web proxy, ProxyMate [30], takes this capability a step further by generating stronger user ID's and passwords transparently to the user, and using them to login to the actual site. Novell's DigitalMe [33] and Yodlee [42] store passwords on a central server[4]. Microsoft's Passport [32] service as well as other e-wallet ser-

[4]These systems provide convenience, but do not enhance security from untrusted endpoints, as the master password still needs to be entered and thus, can be captured or hijacked by the untrusted endpoint. In contrast, our architecture's Identity Service provides one-time passwords and fine-grain control over security.

vices hold personally identifying information to simplify form filling in e-commerce transactions. This functionality is similar to our architecture; however, we make such information always available by keeping it in the infrastructure, rather than on a user's end device. CommerceNet's IdentitySafe [11] provides users with pseudo identities for purchasing from online merchants and only releases personally identifiable information to sites on a need to know basis.

Our application of security protocol adaptation is derived from the ideas presented in [23], which suggested that security transformations for small devices would be desirable and presented proxy-based access to Kerberized services. Their design places the bulk of Kerberos authentication work on an infrastructure-based proxy, providing indirect authentication of clients and simplifying the requirements of small devices.

Finally, to format content for a range of devices with limited display capabilities, we could leverage related research, such as TACC [16, 17]. By abstracting device functionality into the client-side Format Transcoder, we simplify the process of adding new types of devices. In addition, the Wireless Application Protocol (WAP) [39] presents a standard format for small devices. Other related work in this area includes Oracle's Portal-To-Go which promises any service to any device access [34].

6 Conclusion

In this paper, we present a novel infrastructure-based architecture for providing secure multi-modal access to wide-area services. The architecture uses rule-driven content and security transformation functions to enable access from a wide variety of end devices by decoupling device capabilities from service requirements. This approach greatly simplifies and reduces the amount of work required to support a new device or service.

Providing access to a new service merely requires writing a WebL script to scrape the content into an XML representation. Our example stock service consists of 6 scripts totaling 1000 lines. After gaining familiarity with WebL, support for the Yahoo Mail service was added in a few days. Yahoo Mail support consists of 5 scripts totaling 610 lines.

Support for new device formats is easily provided by writing XSL style sheets. For HTML, the most complex format, there are 1100 lines of code total in all of the style sheets for Yahoo Contest. Yahoo Mail requires 360 lines respectively. Adding support for the WML browser and style sheets for WML pages only required a couple of days effort. The custom WML version of the holdings page was adapted from the HTML version and is only 100 lines compared to the 370 used for HTML tables. Although some development is required, this approach allows considerable customization capabilities. Furthermore, it provides the ability to support interesting formats such as custom WML and voice access which have yet to be seen from generic HTML transformation proxies.

The current implementation returns pages in an average time of approximately 3-4 seconds. The implementation has not yet been optimized and there are many opportunities for improvement. Implementing the system as a vSpace service required a steep learning curve initially. However, the cluster wide management of persistent state provided by the DDS, and auto-discovery of workers are very valuable building blocks. Experimenting with vSpace v1 workers in the distributed configuration showed promising fault tolerance results. We are looking forward to the added features provided by vSpace v2.

We provide a generic "any device to any service" model that is in stark contrast to the traditional approaches of vertical service and device integration and implementation. Our hypothesis is that such a generic model provides rapid support for new devices and services, a critical requirement for the Post-PC era.

Furthermore, the architecture also supports a new model of secure interaction from untrusted public Internet access points found in mobile environments. To our knowledge, our architecture is the first to address this interaction model. By providing a generic control and content rewriting capability, the architecture provides users with precise control over the exposure of information, both as a function of their access devices and as a function of the users' adopted roles. Additionally, the architecture supports the fusion of multiple devices, using trusted portable devices for secure authorization of sensitive requests. Further non-security related applications of multiple device fusion could allow voice access to a service via the car, displaying graphical information on a passenger's PDA.

References

[1] 3Com. Web Clipping Applications Tutorials. `http://www.palm.com/devzone/webclipping`, January 2000.

[2] E. Amir, S. McCanne, and H. Zhang. An Application Level Video Gateway. In *Proc. ACM Multimedia 95*, San Francisco, CA, 1995.

[3] Apache. Apache XML Project. `http://www.apache.org`, March 2000.

[4] Apple. Keychain. `http://www.apple.com/macos/feature4.htm`, January 2000.

[5] D. Balfanz and E. Felten. Hand-held computers can be better smart cards. In *Proc. of the Eighth USENIX Security Symposium*, Berkeley, CA, August 1999.

[6] UC Berkeley. Millennium. `http://www.millennium.berkeley.edu/`, 2000.

[7] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. RFC 2704: The KeyNote Trust-Management System

Version 2. http://www.crypto.com/papers/rfc2704.txt, September 1999.

[8] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/REC-xml, 2000.

[9] Certicom. Elliptic Curve Cryptography for Palm VII. http://www.certicom.com/press/98/dec0298.htm, December 1998.

[10] Cohera. Cohera. http://www.cohera.com, 2000.

[11] CommerceNet. IdentitySafe. http://www.commerce.net/project/hotsheet.html, January 2000.

[12] Compaq. Webl User Manual. http://www.compaq.com/WebL, March 2000.

[13] Security Dynamics. SecurID. http://www.rsasecurity.com, January 2000.

[14] Epicentric. Epicentric Portal Server 3.0 Datasheet. http://www.epicentric.com, 2000.

[15] A. Adler et. al. Extensible Stylesheet Language (XSL). http://www.w3.org/TR/xsl/, 2000.

[16] A. Fox et. al. Adapting to client variability via on-demand dynamic distillation. In *Proc. of the 7th ACM Inter. Conference on Architectural support for Programming Languages and Operating Systems*, Cambridge, Massachusetts, October 1996.

[17] A. Fox et. al. Scalable network services. In *Proc. of the 16th ACM Symp. on Operating Systems Principals (SOSP-16)*, St. Malo, France, October 1997.

[18] L. Cranor et. al. Platform for Privacy Preferences (P3P1.0) Specification. http://www.w3.org/TR/P3P, 2000.

[19] N. Maller et. al. A One-Time Password System. http://www.ietf.org/rfc/rfc2289.txt, February 1998.

[20] S. Gribble et. al. The MultiSpace: an Evolutionary Platform for Infrastructural Services. In *Proc. of the 1999 Usenix Technical Conference*, 1999.

[21] S. Gribble et. al. Scalable, distributed data structures for internet service construction. In *Proc. of the Fourth Symp. on Operating Systems Design and Implementation*, OSDI, October 2000.

[22] Yahoo Finance. Yahoo Finance Investment Challenge. http://contest.finance.yahoo.com/t1?u/, 2000.

[23] A. Fox and S. Gribble. Security on the Move: Indirect Authentication using Kerberos. In *Proc. of the 2nd ACM Inter. Conference on Mobile Computing and Networking*, Rye, New York, November 1996.

[24] A. Frier, P. Karlton, and P. Kocher. The SSL 3.0 Protocol. http://www.netscape.com/eng/ssl3/ssl-toc.html, March 1996.

[25] S. Gribble, M. Welsh, R. von Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R.H. Katz, Z.M. Mao, S. Ross, and B. Zhao. The Ninja Architecture for Robust Internet-Scale Systems and Services. In *Special Issue of IEEE Computer Networks on Pervasive Computing*, 2000.

[26] Steven D. Gribble. *A Design Framework and a Scalable Storage Platform to Simplify Internet Service Construction*. PhD thesis, University of California at Berkeley, September 2000.

[27] InfoWorld. Boeing to put Net in the air. http://www.infoworld.com/articles/hn/xml/00/04/27/000427enboeing.xml, April 2000.

[28] InfoWorld. E-cars take to the streets; wireless connections link road warriors to the Net. http://www.infoworld.com/articles/hn/xml/00/03/13/000313hnauto.xml, March 2000.

[29] M. Langheinrich. A P3P Preference Exchange Language (APPEL) Working Draft. http://www.w3.org/TR/WD-P3P-preferences, 1998.

[30] Lucent. Proxymate. http://www.proxymate.com/, January 2000.

[31] N. Maller. The S/KEY One-Time Password System. http://www.ietf.org/rfc/rfc1760.txt, February 1995.

[32] Microsoft. Passport. http://www.passport.com, January 2000.

[33] Novell. Digitalme. http://www.digitalme.com/, January 2000.

[34] Oracle. Oracle Portal-to-Go Any Service to Any Device. http://www.oracle.com/mobile/panbwp.pdf, October 1999.

[35] Palm.com. Palm V PDA. http://www.palm.com/products/vseries.html, 1999.

[36] B. Schneier. Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). In *Fast Software Encryption, Cambridge Security Workshop Proceedings*, December 1993.

[37] B. Schneier. *Applied Cryptography*. John Wiley and Sons, Inc., 1996.

[38] L. Sweeney. Guaranteeing anonymity when sharing medical data, the Datafly system. In *Proc. of the American Medical Informatics Assoc. Symposium.*, Washington, DC, August 1998.

[39] WAP. WAP Forum Specifications. http://www.wapforum.org/what/technical.htm, January 2000.

[40] M. Welsh, S. Gribble, E. Brewer, and D. Culler. A Design Framework for Highly Concurrent Systems. UC Berkeley CS Technical Report No. UCB/CSD-00-1108, May 2000.

[41] G. Wiederhold and M. Bilello. Protecting Inappropriate Release of Data from Realistic Databases. In *Proc. of Data and Expert Systems (DEXA) Security Workshop*, August 1998.

[42] Yodlee. Yodlee. http://www.yodlee.com/, January 2000.

[43] B. Zenel and D. Duchamp. General purpose proxies: solved and unsolved problems. In *Proc. of the Sixth Workshop on Hot Topics in Operating Systems*, Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 1997.

[44] B. Zenel and D. Duchamp. A general purpose proxy filtering mechanism applied to the mobile environment. In *Proc. of*

the 3rd ACM/IEEE Conference on Mobile Computing and Networking, New York, NY USA, 1997.

7 Appendix A - Example WebL scraping script

The following is a WebL script for scraping the Yahoo Contest holdings page. It includes additional functionality for detecting whether the user has logged in. If the user has not logged in, it generates an XML version of the login request to send to the FCM.

```
// shared includes Files for us...
#include "shared.w"

// this script reads input from a file called <filename>.html and outputs
// to <filename>.xml

// our scripts take two arguments, the desired filename and the sessionid...
// note that it does not take the complete filename... it takes in
// the prefix of the filename... so if you want to read from
// asdf.html and write to asdf.xml, set filename to 'asdf'
var filename = ARGS[1];
var sessionid = ARGS[2];

var page;

// load page from file. abort if we can't open...
if(Trap(page = Files_LoadFromFile(filename + ".html","text/html")) != nil) then
  AbortParse(filename, "error: couldn't open input html file");
end;

// build up xml in this string
var output;

// figure out if we're dealing with the holdings page or the login page...
// determine this by trying to find the words "Please sign in"

// get the title
if(Size(Pat(page, "Please sign in")) != 0) then

// define tags
var PostTag = "post";
var ActionTag = "action";
var ProtocolTag = "protocol";
var PostArgumentsTag = "post-arguments";
var PostargTag = "postarg";
var NameTag = "name";
var ValueTag = "value";
var AcceptTag = "Accept";
var UserAgentTag = "UserAgent";

// we are going to build up the xml in this string...
output = "<?xml version=\"1.0\"?>" + BeginTag(PostTag);

// the login form is the first form on the page...
var form = Elem(page, "form");

// try to grab the first form on the page. this should be okay...
// yahoo puts forms in ads sometimes, but i don't think the
// login page has ads.
if(Trap(form = form[0]) != nil) then
  AbortParse(filename, "parsing error: couldn't find login form");
end;

// fill in the action
output = output + BeginTag(ActionTag);
if(Trap(output = output + XmlQuote(form.action)) != nil) then
  AbortParse(filename, "parsing error: can't parse login form");
end;

output = output + EndTag(ActionTag);

// fill in the protocol
output = output + BeginTag(ProtocolTag);
output = output + XmlQuote("HTTP/1.0");
output = output + EndTag(ProtocolTag);

// fill in the arguments
output = output + BeginTag(PostArgumentsTag);

  // go through all the input fields in the form, and create postarg
  // tags for them...
  var inputs = Elem(form, "input");
  every input in inputs do
    // only parse this input field if it has a name [ie not the submit button]
    if(Trap(input.name) == nil) then
      output = output + BeginTag(PostargTag);
        output = output + BeginTag(NameTag);
        output = output + XmlQuote(input.name);
        output = output + EndTag(NameTag);
        output = output + BeginTag(ValueTag);
        // if this input has a value, fill it in...
if(Trap(input.value) == nil) then
     output = output + XmlQuote(input.value);
        // if this input has no value and the name of the field is "login",
        // put in a placeholder for the fcm to replace...
        elsif(input.name == "login") then
     output = output + XmlQuote("*USERNAME*");
        // if this input has no value and the name of the field is "passwd",
        // put in a placeholder for the fcm to replace...
        elsif(input.name == "passwd") then
          output = output + XmlQuote("*PASSWORD*");
end;
        output = output + EndTag(ValueTag);
      output = output + EndTag(PostargTag);
    end;
  end;
output = output + EndTag(PostArgumentsTag);

// fill in the accepted mime types
output = output + BeginTag(AcceptTag);
output = output + XmlQuote("text/html");
output = output + EndTag(AcceptTag);

// fill in the userAgent
output = output + BeginTag(UserAgentTag);
output = output + XmlQuote("postpcproxy");
output = output + EndTag(UserAgentTag);
output = output + EndTag(PostTag);

// end parsing of login page... if we didn't find "Please sign in", it's
// the holdings page, so run the code below...
else

// xml tags for a normal holding:
var holdinginfo = ["StockSymbol",
                   "TimeOfPrice",
                   "Price",
                   "Change",
                   "Volume",
                   "Shares",
                   "Value",
                   "ValueChange",
                   "PercentValueChange",
                   "Paid",
                   "Gain",
                   "PercentGain",
                   "MoreInfo"];

// xml tags for cash: [cash doesn't have last trade, change, or volume, so it's
// special
var cashinfo = ["StockSymbol",
                "Shares",
                "Value",
                "ValueChange",
                "PercentValueChange",
                "Paid",
                "Gain",
                "PercentGain",
                "MoreInfo"];

// xml tags for the totals:
var totalinfo = ["TotalValue",
                 "TotalValueChange",
                 "PercentTotalValueChange",
                 "TotalPaid",
                 "TotalGain",
                 "PercentTotalGain",
                 "MoreInfo"];

// define tags...
var HoldingsTag = "Holdings";
var UsernameTag = "Username";
var PortfolioTag = "Portfolio";
var PerformanceQuoteTag = "PerformanceQuote";
var PerformanceTotalsTag = "PerformanceTotals";
// the quote for cash has to be parsed specially, but we tag it like
// any other performance quote..
var CashTag = "PerformanceQuote";

// we are going to build up the xml in this string...
output = "<?xml version=\"1.0\"?>" + BeginTag(HoldingsTag);

// parse the links
if (Trap(output = output + ParseLinks(page, sessionid)) != nil) then
  AbortParse(filename, "parsing error: couldn't parse links!");
end;

// parse the form to get quotes
if(Trap(output = output + ParseGetQuotes(page, sessionid)) != nil) then
  AbortParse(filename, "parsing error: couldn't parse getquotes form!");
end;

// look for "Welcome, _______"
var greeting = Pat(page, `Welcome, ([^ \n]+)`);

// use the first match we get...
if(Trap(greeting = greeting[0]) != nil) then
  AbortParse(filename, "parsing error: couldn't find greeting!");
end;

// username is at index 1 in the greeting... index 0 is the entire matched
// string ["Welcome, <username>"]
if(Trap(output = output + BeginTag(UsernameTag) + greeting[1] + End-
Tag(UsernameTag)) != nil) then
  AbortParse(filename, "parsing error: couldn't find username!");
end;
```

```
// look for the portfolio
output = output + BeginTag(PortfolioTag);

// look for a table heading that contains the word "Symbol"
var headings = Elem(page, "th");
var symbolheading = headings intersect Pat(page, "Symbol");

// find the table that contains a heading for Symbol
var table = Elem(page, "table") contain symbolheading;

// use the first table we find with a heading for Symbol
if(Trap(table = table[0]) != nil) then
  AbortParse(filename, "parsing error: couldn’t find portfolio table");
end;

// get a list of the rows in this table... remove nested tables...
var rows = Elem(table, "tr") without Elem(table, "table");
every row in rows do

  // this is a list of the columns in this row
  var columns = Elem(row, "td");

  // the info we should be using...
  var info;

  // the tag we should use for the quote...
  var quotetag = "";

  // if we have the same number of columns as there are in the
  // cash line, then we have the cash line... etc...
  if Size(columns) == Size(cashinfo) then
    quotetag = PerformanceQuoteTag;
    info = cashinfo;
  elsif Size(columns) == Size(totalinfo) then
    quotetag = PerformanceTotalsTag;
    info = totalinfo;
  elsif Size(columns) == Size(holdinginfo) then
    quotetag = CashTag;
    info = holdinginfo;
  else
    quotetag = "";
  end;

  if ! (quotetag == "") then
    // tag this quote
    output = output + BeginTag(quotetag);

    // keep count of which column we’re on
    var currentcolumn = 0;
    every column in columns do
      // tag the data
      output = output + BeginTag(info[currentcolumn]) +
               Text(column) + EndTag(info[currentcolumn]);
      currentcolumn = currentcolumn + 1;
    end;
    output = output + EndTag(quotetag);
  end;
end;

output = output + EndTag(PortfolioTag);
output = output + EndTag(HoldingsTag);

// end parsing of holdings page...
end;

// regardless of whether it’s a login page or a holdings page, the
// same procedure can be used for dumping it to disk.

Files_SaveToFile(filename + ".xml",output);
```

8 Appendix B - Example XSL Style Sheets

The following are example XSL style sheets for rendering the Yahoo Contest holdings page content in HTML and in WML.

8.1 Holdings Page HTML Style Sheet

```
<!-- this stylesheet transforms the holdings page to HTML -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<!-- load the shared stuff -->
#include "shared.x"

<!-- heirarchy for holdings:
Holdings
  Links
    YahooTradeHref
    YahooHoldingsHref
  GetQuotes
    Action
  Username
  Portfolio
    PerformanceQuote
      StockSymbol
      TimeOfPrice
      Price
      Change
      Volume
      Shares
      Value
      ValueChange
      PercentValueChange
      Paid
      Gain
      PercentGain
      MoreInfo
    PerformanceTotals
      NumSymbols
      TotalValue
      TotalValueChange
      PercentTotalValueChange
      TotalPaid
      TotalGain
      PercentTotalGain
-->

<xsl:template match="Holdings">
  <html>
    <head>
      <title>
      your yahoo holdings
      </title>
    </head>
    <body>
      <xsl:apply-templates select="Links"/>
      <xsl:apply-templates select="GetQuotes"/>
      <P>
        <b>
        Welcome, <xsl:apply-templates select="Username"/>
        </b>
      </P>
      <xsl:apply-templates select="Portfolio"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="Username">
  <i>
  <xsl:apply-templates/>
  </i>
</xsl:template>

<xsl:template match="Portfolio">
  <P>
  Portfolio for <xsl:apply-templates select="/Holdings/Username"/>
  </P>
  <table border="1" cellpadding="1" cellspacing="0">
    <tr bgcolor="#dcdcdc">
      <th nowrap="true">
      symbol
      </th>
      <th nowrap="true" colspan="2">
      last trade
      </th>
      <th nowrap="true">
      change
      </th>
      <th nowrap="true">
      volume
      </th>
      <th nowrap="true">
      shares
      </th>
      <th nowrap="true">
      value
      </th>
      <th nowrap="true" colspan="2">
      value change
      </th>
      <th nowrap="true">
      paid
      </th>
      <th nowrap="true" colspan="2">
      gain
      </th>
      <th nowrap="true">
      more info
      </th>
    </tr>
    <xsl:apply-templates select="PerformanceQuote"/>
    <xsl:apply-templates select="PerformanceTotals"/>
  </table>
</xsl:template>

<xsl:template match="PerformanceQuote">
  <tr align="right">
  <xsl:apply-templates/>
  </tr>
</xsl:template>

<xsl:template match="PerformanceTotals">
  <tr align="right">
    <th align="right" colspan="6">
    Totals(USD):
    </th>
    <xsl:apply-templates/>
  </tr>
</xsl:template>
```

```
<!-- if the stocksymbol is cash, it needs to span 5 columns -->
<xsl:template match="StockSymbol">
  <xsl:choose>
    <xsl:when test="contains(text(), '$$CASH')">
      <td nowrap="true" align="left" colspan="5">
      <xsl:apply-templates/>
      </td>
    </xsl:when>
    <xsl:otherwise>
      <td nowrap="true" align="left">
      <xsl:apply-templates/>
      </td>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="TimeOfPrice">
  <td nowrap="true" align="center">
  <xsl:apply-templates/>
  </td>
</xsl:template>

<xsl:template match="Price">
  <td nowrap="true">
    <b>
    <xsl:apply-templates/>
    </b>
  </td>
</xsl:template>

<!-- if things are negative, color them red. else color them green -->
<xsl:template match="Change">
  <xsl:choose>
    <xsl:when test="contains(text(),'-')">
      <td nowrap="true">
        <font color="red">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:when>
    <xsl:otherwise>
      <td nowrap="true">
        <font color="green">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="Volume">
  <td nowrap="true">
  <xsl:apply-templates/>
  </td>
</xsl:template>

<xsl:template match="Shares">
  <td nowrap="true">
  <xsl:apply-templates/>
  </td>
</xsl:template>

<xsl:template match="Value">
  <td nowrap="true">
  <xsl:apply-templates/>
  </td>
</xsl:template>

<xsl:template match="ValueChange">
  <xsl:choose>
    <xsl:when test="contains(text(),'-')">
    <td nowrap="true">
      <font color="red">
      <xsl:apply-templates/>
      </font>
    </td>
    </xsl:when>
    <xsl:otherwise>
      <td nowrap="true">
        <font color="green">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="PercentValueChange">
  <xsl:choose>
    <xsl:when test="contains(text(),'-')">
    <td nowrap="true">
      <font color="red">
      <xsl:apply-templates/>
      </font>
    </td>
    </xsl:when>
    <xsl:otherwise>
      <td nowrap="true">
        <font color="green">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="Paid">
  <td nowrap="true">
  <xsl:apply-templates/>
  </td>
</xsl:template>

<xsl:template match="Gain">
  <xsl:choose>
    <xsl:when test="contains(text(),'-')">
    <td nowrap="true">
      <font color="red">
      <xsl:apply-templates/>
      </font>
    </td>
    </xsl:when>
    <xsl:otherwise>
      <td nowrap="true">
        <font color="green">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="PercentGain">
  <xsl:choose>
    <xsl:when test="contains(text(),'-')">
      <td nowrap="true">
        <font color="red">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:when>
    <xsl:otherwise>
      <td nowrap="true">
        <font color="green">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="MoreInfo">
  <td nowrap="true" align="center">
    <small>
    <xsl:apply-templates/>
    </small>
  </td>
</xsl:template>

<xsl:template match="TotalValue">
  <td nowrap="true">
    <b>
    <xsl:apply-templates/>
    </b>
  </td>
</xsl:template>

<xsl:template match="TotalValueChange">
  <xsl:choose>
    <xsl:when test="contains(text(),'-')">
      <td nowrap="true">
        <font color="red">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:when>
    <xsl:otherwise>
      <td nowrap="true">
        <font color="green">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="PercentTotalValueChange">
  <xsl:choose>
    <xsl:when test="contains(text(),'-')">
      <td nowrap="true">
        <font color="red">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:when>
    <xsl:otherwise>
      <td nowrap="true">
        <font color="green">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="TotalPaid">
  <td align="right">
  <xsl:apply-templates/>
  </td>
</xsl:template>

<xsl:template match="TotalGain">
  <xsl:choose>
```

```
    <xsl:when test="contains(text(),'-')">
      <td nowrap="true">
        <font color="red">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:when>
    <xsl:otherwise>
      <td nowrap="true">
        <font color="green">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="PercentTotalGain">
  <xsl:choose>
    <xsl:when test="contains(text(),'-')">
      <td nowrap="true">
        <font color="red">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:when>
    <xsl:otherwise>
      <td nowrap="true">
        <font color="green">
        <xsl:apply-templates/>
        </font>
      </td>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

8.2 Holdings Page WML Style Sheet

```
<?xml version="1.0"?>
<!-- this stylesheet transforms the holdings page to WML -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output omit-xml-declaration="yes"/>

<!-- load the shared stuff -->
#include "shared.x"
#include "error.x"

<!-- heirarchy for holdings:
Holdings
  Links
    YahooTradeHref
    YahooHoldingsHref
  GetQuotes
    Action
  Username
  Portfolio
    PerformanceQuote
      StockSymbol
      TimeOfPrice
      Price
      Change
      Volume
      Shares
      Value
      ValueChange
      PercentValueChange
      Paid
      Gain
      PercentGain
      MoreInfo
    PerformanceTotals
      NumSymbols
      TotalValue
      TotalValueChange
      PercentTotalValueChange
      TotalPaid
      TotalGain
      PercentTotalGain
-->

<xsl:template match="Holdings">

<wml>
  <card id="Holdings">
        YahooContest Portfolio
        <p></p>
<p> Company     Shares     Value</p>
<p>------------------------------</p>
 <xsl:apply-templates select="Portfolio"/>
        <p></p>
  </card>
</wml>

</xsl:template>

<xsl:template match="Portfolio">
  <p>Portfolio for <xsl:apply-templates select="/Holdings/Username"/> </p>
  <xsl:apply-templates select="PerformanceQuote"/>
</xsl:template>

<xsl:template match="PerformanceQuote">
<p><xsl:apply-templates select="StockSymbol"/>   <xsl:apply-templates se-
lect="Shares"/>    <xsl:apply-templates select="Value"/> </p>
</xsl:template>

<!-- if the stocksymbol is cash, it needs to span 5 columns -->
<xsl:template match="StockSymbol">
  <xsl:choose>
    <xsl:when test="contains(text(), '$$CASH')">
      <xsl:apply-templates/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="Shares">
 :  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="Value">
  :  $<xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

9 Biographies

9.1 Steven James Ross

Steven James Ross received a B.S.E. degree in Computer Engineering from the University of Michigan Ann Arbor in 1996. After working for Netscape Communications Corporation, he received an NSF Fellowship and a M.S. degree in Computer Science from the University of California Berkeley in 2000. Steve is currently working at the OnStar Advanced Technology Lab. His research interests include telematics, mobile computing, internet systems, and post-pc devices.

9.2 Jason L. Hill

Jason Hill received a B.S. degree in Electrical Engineering and Computer Science from the University of California, Berkeley in 1998. He is currently a Ph.D. student at University of California, Berkeley. As a member of the TinyOS research team, his principal field of research is Sensor Networks including both hardware and software.

9.3 Mike Yen-Yang Chen

Mike Yen-Yang Chen received a B.S. degree in Computer Science from the University of British Columbia in Canada in 1997. He is currently a Ph.D. student at the University of California, Berkeley. His research interests include highly-scalable, cluster-based Internet services, availability benchmarks, and search engines. He is the principal architect of the Ninja vSpace.

9.4 Dr. Anthony D. Joseph

Dr. Joseph received his B.S., S.M., and Ph.D. degrees in computer science from MIT. He joined the UC Berkeley faculty in 1998, where he is working on the integration of mobile telephony systems, wireless packet radio, and wired/wireless IP-based networks. His principal field of interest is computer systems: mobile systems, operating systems, networking, and computer architectures for clustered and distributed systems.

9.5 Dr. David Culler

David E. Culler is a Professor of Computer Science at the University of California and University Director of the Intel Berkeley Extreme Interconnected Systems Lab. He has been on the faculty at Berkeley since 1989 and has served as Vice Chair for Computing and Networking. He received his Ph.D. from MIT in 1989. He was awarded the NSF Presidential Young Investigator in 1990 and the Presidential Faculty Fellowship in 1992. His research addresses parallel computer architecture, parallel programming languages, and high performance communication structures. He is well known for his work on Networks of Workstations (NOW), Active Messages, Split-C, the Threaded Abstract Machine (TAM), and dataflow systems. He has published widely in leading conferences and journals, obtained three patents, and recently completed a graduate text called Parallel Computer Architecture: A Hardware/Software Approach (Morgan-Kaufmann, publisher). He has served as General Chair and Program Chair for Hot Interconnects, Program Chair for the ACM Symposium on Parallel Algorithms and Architectures, Technical Papers Chair for SC2001 and Co-editor for special issues of IEEE Transactions on Parallel and Distributed Computing and IEEE Micro.

9.6 Dr. Eric A. Brewer

Dr. Eric A. Brewer is an Associate Professor of Computer Science at U.C. Berkeley, and the Founder and Chief Scientist of Inktomi Corporation. He received his PhD in CS from MIT in 1994. Interests include mobile and wireless computing (the InfoPad and Barwan projects); scalable servers (the NOW and Inktomi projects); and application- and system-level security (the ISAAC project). He is a World Economic Forum ”Global Leader for Tomorrow”, one of InfoWorld’s top ten innovators (in information systems), and was named the ”most influential” architect on the evolution of the Internet by the Industry Standard. Technology Review, Vanity Fair, Upside and Forbes have all listed him in their leaders of the Internet lists, including a cover photo with Forbes. Last year he founded the Federal Search Foundation, which helped to create the first government wide portal, FirstGov.gov, for which he won the Federal CIO Council’s Azimuth Award for 2001.