ICAP vs. SOAP: Which One is Better for Edge Services

Vikrant Mastoli, Valmik Desai, and Weisong Shi
{vikrang,valmik,weisong}@wayne.edu

Technical Report: **MIST-TR-03-002**

February 2003

Mobile and Internet SysTem Group (MIST)
Department of Computer Science
Wayne State University
Detroit, MI 48202
http://mist.cs.wayne.edu

ICAP vs. SOAP: Which One Is Better for Edge Services ?

Vikrant Mastoli, Valmik Desai, and Weisong Shi

Mobile and Internet SysTem Group
Department of Computer Science
Wayne State University
{*vikrant,valmik,weisong*}*@wayne.edu*

Abstract

The increase in web traffic leads to the deployment of network intermediaries like caching proxies and content delivery surrogates between the origin servers and the end-users. However, recent trends indicate an increasing demand for deploying content-oriented services along the data path between end users and content servers. These services range from dynamic content generation to support personalization services, security, content filtering and adaptation services. Deploying these services into the existing end-to-end architecture of the Internet requires the creation of an environment to host a variety of services. This suggests extending existing network infrastructure such as caching proxies for more than just their original intended purpose. Extending the network infrastructure implies equipping intermediary machines so that they are able to provide content-oriented services to end-users. This leads to the creation of an environment that allows execution of these services locally and remotely.

In this paper we designed, implemented and evaluated a Service Execution Environment (SEE) in the context of CONCA proxy cache, and compared the performance of Simple Object Access Protocol (SOAP) and IETF's Internet Content Adaptation Protocol (ICAP) when these are used as call-out protocols between the SEE and the service providers.

1 Introduction

The role of the Internet has undergone a transition from simply being a data repository to one providing access to a plethora of sophisticated network-accessible services such as e-mail, banking, on-line shopping and entertainment. Additionally, these services are increasingly being accessed by mobile consumers using end devices such as PDAs, Pocket/Handheld PCs, cellular phones and two-way pagers that connect to the internet using a variety of wireless networking options ranging from Bluetooth [15] to Wireless 3G [29]. The combination of these two trends holds out the possibility of providing a user with seamless, ubiquitous access to a service irrespective of the user's end device and location. Although compelling, achieving this goal requires coping with the inherent mismatch between the low-bandwidth, limited resource characteristics of wireless mobile devices and the high-bandwidth expectations of many content-rich services.

Current day applications and services cope with the above problems essentially by providing differentiated service for different networks/end-devices. For example, most popular news, e-mail, and stock trading services today present a different front-end for mobile users. Although adequate in some scenarios, this approach suffers from the limitation that mobile users are classified into a small number of classes and may not receive performance commensurate with the capabilities of the device or network they are using. More importantly, such an approach cannot adequately cope with dynamically changing environments where there is a big variation in available bandwidth (e.g., a user on a wireless LAN who is at different distances from an access point). More promising are several recently proposed infrastructures [8, 9, 14, 11, 13, 33], which allow the construction of *network-aware access paths* from application-specific component; these components cope with device and network mismatches by handling activities such as protocol conversion and content transcoding at sites best suited for them. Similarly, the OPES initiative [27] proposed by IETF Open Pluggable Edge Service Workgroup share the same goal to add third-party value-added services along

the data path between data consumers and data providers.

Although several such infrastructures have been proposed, they have so far not seen widespread use because of concerns about their deployability, performance, and scalability. Central to each of these concerns is the question of whether it is possible to construct paths that yield performance benefits over a range of (possibly dynamically changing) network conditions, and where to deploy these services, and how to integrate with the existing data flows, and who make the decision of service selection. Most of the previous work focus on the infrastructure support for the services, but neglect these questions. In this paper, we argue that the idea of extending existing caching proxies to support these services is promising, the service selection should be separated from the execution of services, and most importantly, the infrastructure should allow those value-added services talking with different protocols. Based on these arguments, we design, implemented, and evaluated a Java-based service execution environment to support service execution both locally and remotely. The major contribution of this paper is:

- A novel design of a Service Execution Environment (SEE), having several unique features, which includes (1) a secure interface between the service provider and SEE for the service provider to register the service; (2) no requirement of a language to configure the rules as it provides a simple web interface that allows authorized parties (clients, network providers, content providers) to select the services they desire based on the name of the services (high level type description). This simplifies the process of configuring the environment and decouples the rule generation and rule processing, thus allowing more flexibility when defining rules.

- To the best of our knowledge, our work is the first public experimental platform which supports multiple protocol bindings. As such, it is an ideal platform for other colleagues to test different call-out protocols [5].

- We implement three services and invoke them remotely by using the call-out protocols: Internet Content Adaptation Protocol (ICAP) [16] and Simple Object Access Protocol (SOAP) [30]. Then, we compare their performance from the perspective of performance, codeability, and scalability in this paper. We have performed a thorough analysis to effectively describe the overheads when using the service execution environment. The paper explicitly describes the performance overhead both at the proxy end as well as at the call-out server when using ICAP and SOAP as call-out protocols. This penalty includes the network overhead as well as the time taken at the ICAP/SOAP server to process the request. We found that ICAP outperforms SOAP by a little bit, but it requires more work on both at the caller and the callee, and it requires service provider to rewrite all the legacy codes. Therefore, we argue that SOAP protocol is a better choice as a call-out protocol.

The rest of the paper is structured as follows: Section 2 provides background information of the architecture of the CONCA. Section 3 provides the motivation behind the implementation of SEE and a description of the design of the SEE. Section 4 describes the implementation of SEE and the technology used to develop the prototype. Section 5 presents the results of the performance evaluation of SEE, SOAP and ICAP. Related Work and concluding remarks are listed in section 6 and Section 7.

2 Background

2.1 End-to-End is Not Enough

Since the evolution of Internet lot of the intelligence has been at the end systems. The rapid proliferation of Internet users and increasing web traffic have led to a lot of load on the origin servers and thereby led the content-providers to adopt techniques that disseminate the load on origin servers. The deployment of caching proxies and surrogates allow the distribution of content and transport the origin servers closer to the edge of the network. The rising demand for Internet services induces the idea of using existing caching proxies for more than simply accelerating the delivery of Web content. They seem to provide a viable location to deploy additional client services. This implies a change in the current Internet model where the client and the server are the two endpoints of communication and the introduction of "intelligent" networks where intermediaries could process certain requests and responses [6]. This suggests that the Internet will no longer be a mere data transfer network, more and more functionalities can be injected into the network along the data path, ranging from the network-layer, such as active networks [26], to application-layer,

such as CANS infrastructure [11].

2.2 CONCA Proxy Cache

CONCA (**CO**nsistent **N**omadic **C**ontent **A**ccess) [25] is a proposed edge architecture for the efficient caching and delivery of dynamic and personalized content to users who access the content by using diverse devices and connection technologies. CONCA attempts to exploit reuse at the granularity of individual objects making up a document, improving user experience by combining caching, prefetching, and transcoding operations as appropriate.

To achieve its goals, CONCA relies on additional information from both servers and users. All content supplied by servers in CONCA architecture is assumed to be associated with a "document template" which can be expressed by formatting languages such as XSL-FO [32] or edge-side include (ESI) [28]. Given this information, CONCA node can efficiently cache dynamic and personalized content by storing quasi-static document templates and reusing sharable objects among multiple users. Moreover, based on the preference information provided by users, a CONCA cache node delivers the same content to different users in a variety of formats using transcoding and reformating.

Figure 1 provides an overview of the CONCA architecture. CONCA uses a distributed client-side proxy cache architecture, similar to NSF's IRcache project [17] and other recently proposed projects [34, 19]. Such architectures, which are complementary to server-side solutions such as reverse proxy caches and content delivery networks [1, 7], attempt to reduce network traffic associated with a miss in the local cache; ideally, such schemes would service the miss from a "near" proxy as opposed to requesting the object from the original server. In CONCA, as we shall see below, distributed proxy caches are the key to providing scalability.

Each CONCA node consists of cache storage and three modules—*remote control unit*, *local control unit* and *resource management unit*—which manage the node's interaction with other cache nodes, users, and internal storage policies respectively. Each CONCA node leverages the document templates provided by server and user preferences provided by user to provide two broad kinds of support: (*a*) consistent caching of dynamic personalized content, even when some of the content needs to be transcoded prior to delivery to the client; and (*b*) support for nomadic users, by enabling efficient recreation of per-user cache state.

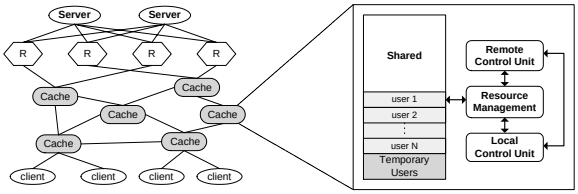


Figure 1: CONCA architecture: distributed proxy caches receive data from multiple server replica (R).

3 Service Execution Environment

3.1 Motivation

The Service Execution Environment(SEE) is a part of the ongoing development of CONCA project, an edge-computing platform. The architecture of CONCA is modeled on two recent trends: (a) Increasing amounts of dynamic and personalized content, and (b) A significant growth in "on-the-move" access using various mobile resource-constrained devices. The delivery of personalized content requires the deployment of an environment that generates content dynamically. An extension of this idea is the ability to support other value-added services. Typically third party vendors provide these services and hence the platform must allow dynamic addition and removal of services. It is proposed that this environment be deployed as a part of the CONCA node to aid in reducing the load on origin servers by allowing code to be downloaded and executed on the CONCA platform. Our execution environment is designed to deliver content that is tailored to the preferences of both the end-user as well as the content provider. This allows the end-user to enable services that allow personalization, as well as guarantee privacy and security for all communication. At the same time, it presents a business opportunity for ISP's and content provider's to provide these value-added services to their clients. These services could include access control to block inappropriate sites, virus scanning, anonymization services to protect privacy, language translation, addition of region-specific information, image resizing and image filtering to reduce the quality of images to shorten download time.

Current design proposals for an execution environment proposed the OPES workgroup [27] have limited ability to handle service registration and security issues. Further they require that the client, content-provider and the network provider configure complex rules in order to select the services. In this paper , we argue that the service selection should be separated from the execution of

services, and most importantly, the infrastructure should support different protocols for the environment to talk to the value-added services.

3.2 Objectives

Our service execution environment is designed with the following features in mind: secure, scalable, high-performance and ease-of-use. We demonstrate these requirements by describing how existing architecture [3, 27] do not prove to be as efficient as intended. First, OPES framework forces both content providers and content consumers to use IRML [4] to specify the rules to determine if a service should be invoked on certain content. Although it provides a standard interface, it is unrealistic to ask content providers or end users to write such sort of complex rules to employ some personalized services. Furthermore, because of the prevalence of web services, it is impossible to ask service providers to support one protocol only, i.e., Internet Content Adaptation Protocol (ICAP) proposed by OPES framework. Finally, they did not consider the secure interaction between service providers and proxy cache, and the mechanism for content integrity check from both client-side and server-side.

Our approach to address the above problems depends on the following five components: (1) A secure interface between service provider and the service execution environment that allows the service provider to register their services, and update their service information later. This is handled by the *service manager* module on the left side of Figure 2; (2) A simple Web interface that allows authorized parties (clients, network providers, content providers) to select the services they desire as well as choose providers for the selected services (some services may be provided by many providers). The decision of where the services are executed is left to service execution environment. When multiple services are chosen by one client, the composition of these services is done automatically after he or she finishes the configuration online by using the type-based service composition technique proposed in [11]. This ensures that they will be executed in correct order and further helps to improve the performance of SEE since this composition is done offline; (3) Supporting multiple protocol bindings, such as ICAP and W3C's SOAP to make the execution environment more flexible. (4) A protected local service execution environment for secure execution of injecting code. We plan to design a set of application programming interfaces (APIs) between proxy cache and protected execution environment, and use virtual memory monitors (VMMs) technique to provide process isolation and protection within the environment, which has been recently shown to be very useful in [31]; (5) Providing feedback-based content integrity mechanisms to allow both client and content provider to check the correctness of content after applying the content-oriented services. In this paper we plan to discuss the former three points.

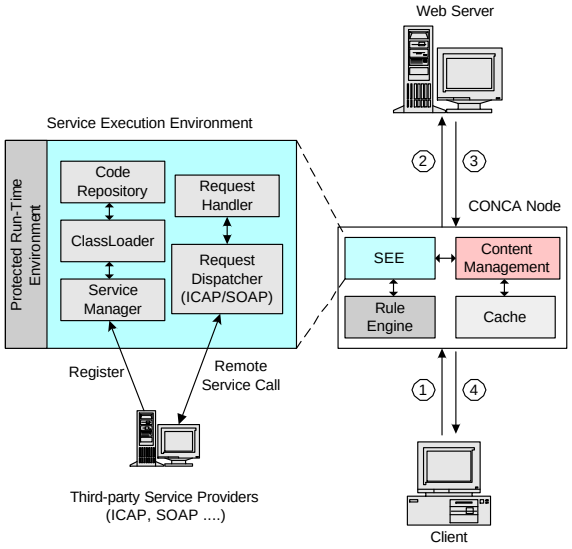
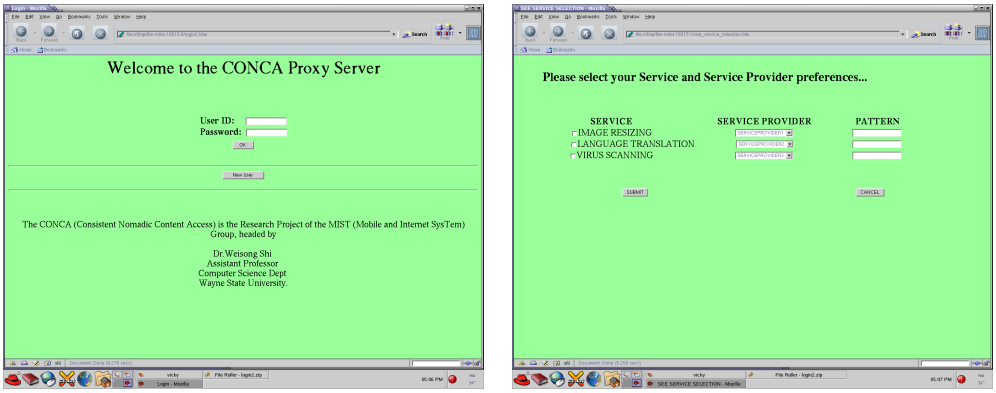


Figure 2: The architecture of service execution environment.

3.3 User Interface

The most important feature of our design is the simplicity with which the environment can be configured. We create a web login for user's that would authenticate them and then present a webpage outlining the current services available and the names of all service providers, as shown in Figure 3. We ensure that this webpage accurately reflects the most recent list of active services by creating it dynamically when the user logs in. This provides flexibility and allows users to select and configure how they prefer the content be delivered to them. But more importantly it allows them to decide who should provide them these services. This addresses an increasingly important issue; privacy, that dictates who should be allowed access to your web content and your details. The user can also specify the service and the rules for it through the web interface by using regular expressions. For example, if user A wants Image Resizing to be done over jpeg files then the expression can be "*.jpg, IR".



(a) (b)

Figure 3: A snapshot of (a) login session, and (b) service selection screen.

3.4 Service Manager

To handle different services with vastly differing parameters it is important that the task of managing services be assigned to a dedicated module. This is done by the service manager. It is used to handle the details of all the services that are currently deployed. It ensures that the back end always accurately reflects the services that are currently active. The `ServiceRegistry` handles the registration of services by the service provider through RMI. The entire service registration process between SEE and the service providers is done securely.

3.5 Processing Flow

When the client logs in, he is presented with a screen that reflects the services that are currently registered with the network/content provider. When the user selects preferences these are logged. We term the services selected by the user as "user-specific services". We also assume that the network provider may also wish to configure services that would be applied to all requests irrespective of user. We name these services "general services". Each request from a user would be serviced for all general services as well as user-specific services, if any.

For every request that comes in, a HTTP Object is created by SEE. This object represents all the information about the request at any point during processing and so forms the basis of communication between all the modules of SEE. As requests come in, the Rule Engine is invoked for each one of them. The Class Loader loads the rule file of each service present in the properties file of the user. The Rule class of a service determines whether the request satisfies the rules of the service or not, if it does then further processing is handed over to the Request Handler. The Request Handler handles the processing for different protocols. It passes the protocol specific requests created by the ICAP/SOAP client to the ICAP/SOAP server and retrieves the responses, which are then passed back to the Rule Engine through the Rule class. This is important since the decision to invoke further rules is based on the modified request/response obtained by the processing of earlier rules. Once all the rules are processed the final modified response is sent to the client. The left part of Figure 2 shows the overview of the six logical modules of SEE.

4 Implementation

Our prototype implementation of SEE is based on the architecture described in the earlier sections. We have deployed three services on the proposed architecture and used a basic proxy server that has no caching capability. We support two protocols that serve as call-out protocols, ICAP and SOAP. Also our testing environment involves locally as well as remotely deployed service modules. However, all our local services are invoked using a "local host" URL which means that although we have support for local service execution we treat them as remote services by using a URL to access them. The proxy server and all the architectural modules have been implemented in the Java programming language. The following subsections provide a detail explanation of the implementation of the modules.

4.1 Service Manager Implementation

The service manager maintains a data structure that records the details of all services and providers. This includes the following details: the service name, if it is

Image Resizer	Provider_A,www.a.com, ICAP, Remote
	Provider_B, localhost:8080, SOAP, Local, ImageResizer
	Provider_C, www.c.net, ICAP, Remote
Virus Scanning	Provider_A, www.a.com, ICAP, Remote, arguments(optional)
	Provider_C, www.c.net, ICAP, Remote
	Provider_D, www.d.com, SOAP, Remote, checkandremove()

Table 1: An example structure of service management in SEE.

a remote service then the URL where the service will be invoked, the protocol through which the request handler should communicate with the service, and a method name if the protocol is SOAP. These are stored such that they can be indexed by the service name. The data structure looks as shown in Table 1. To populate this data structure it is required that service providers register their services. This is implemented using Java RMI over SSL. This allows us to ensure secure communication between the service provider and the proxy server. Further more the issue of authentication is solved using asymmetric key cryptography. We assume that the proxy server and the service provider share a trust relationship and the service provider has been assigned a username and password offline. To register their service the providers call the `RegisterService()` method. The signature of the method is given below:

```
public boolean RegisterService(
    String username,
    byte[] EncryptedPassword,
    String servicename,
    String serviceURL,
    String protocol,
    String location,
    String method);
```

4.2 Rule Engine Implementation

The user preferences are recorded by writing the choices of each user into a properties file and using the cookie (login name) for that user to name the properties file. The recording of user preferences is done using Java servlets. The rule engine reads the properties file at run time. For each service selected SEE loads a Rule class through the class loader. If a request satisfies the rules that have been set by the service provider for the service then it is sent to the request handler for further processing. If the request doesn't satisfy the rules for any of the services present in the properties file then the response for the request is sent to the user without any modification. For the efficient loading of the rule classes and uniform processing of all requests we require that all rules implement the `Rule` Interface, which is:

```
public interface Rule {
  boolean check(HTTPObj hobject);
  Message getModifiedMessage(HTTPObj h);}
```

The `check` method indicates whether a given request satisfies a rule. It is the only method that would be executed for all requests and thus presents the real overhead of the Rule Engine. The `getModifiedResponse` method closely ties the rules and the services associated with those rule. It indicates that once a request satisfies a rule it can be retrieved using that rule thus eliminating the need for "rule processing points" specified in OPES.

4.3 Request Handler Implementation

The Request Handler gets requests (only those which satisfy the rules of a service) from the Rule Class. The ICAP and SOAP clients have been implemented in this module.

4.4 ICAP and SOAP Client Implementation

The ICAP client creates the ICAP request and sends it to ICAP server for service invocation on the contents encapsulated in the request. On receiving the response from the server it removes the ICAP headers and sends the processed content to the Rule class. It sends ICAP requests for two modes: REQMOD (Request Modication) and RESPMOD (Response Modification). The SOAP client has been implemented by using the Apache Axis 1.0 engine. It creates a call to the SOAP server using the end-point address of the machine on which the service is present and the contents to be send to the service for processing. On getting the processed contents it sends it to the Rule class.

4.5 ICAP and SOAP Server Implementation

The ICAP servers are run using three java files, `ServiceProvider_A`, `ServiceProvider_B` and `ServiceProvider_C` which corresponds to image resizing, language translation and virus scanning respectively. The service providers listen to incoming requests on port numbers 1344, 1345 and 1346. They parse the ICAP request, invoke the service on the content and then serve the ICAP client with an ICAP response containing the processed content.

The SOAP server is deployed using a `TOMCAT 3.2.4` server from Jakarta initiative [2]. The implementation of the service modules is in Java. A jws (Java web services) version of the service are deployed at the webapps/axis directory of the TOMCAT Server. The SOAP client creates a call to the SOAP server and then invokes it.

5 Performance Evaluation

5.1 Environment Setup

The performance evaluation was done by setting up an emulated environment consisting of three machines, including a web server, on a Ultra-Sparc2 200MHz with 512MB memory; The SEE, on a Pentium-IV (2.2GHz) Desktop with 512MB memory; The service providers, on a Pentium-IV (2.2GHz) Desktop with 512MB memory. In order to avoid the effects of a Web browser and the Internet traffic on our evaluation, all the requests to the SEE is generated by a client program running locally, implemented in Java. The client program records all the requests for a fixed web page, which consists of one html file (29KBytes) and five images (34KBytes each). The ICAP and the SOAP servers, and the SEE are based in the same LAN. This is to avoid the effects of the external traffic and to avoid network congestion during the performance evaluation. We have also placed our test page at a node on the same LAN to perform our experiment in a restricted environment.

Figure 4 lists the detailed timeline of a request and reply between a client, origin web server and service provider. Based on the this figure, the overhead that we are interested in are defined as follows:

T(Response) = $T_{16} - T_1$
T(Rule Engine) = $(T_{15} - T_{14})+(T_5 - T_4)+(T_3 - T_2)$
T(Origin Server) = $T_4 - T_3$
T(ICAP/SOAP Client) = $(T_7 - T_6)+(T_{13} - T_{12})$
T(ICAP/SOAP Server) = $(T_9 - T_8)+(T_{11} - T_{10})$
T(Service) = $T_{10} - T_9$
T(Network) = $(T_8 - T_7)+(T_{12} - T_{11})$

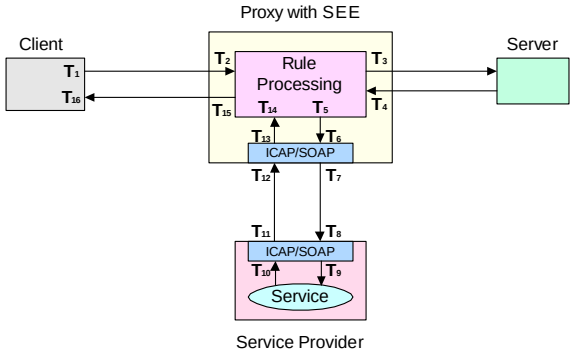


Figure 4: The detailed timeline of a request and reply.

5.2 Three Services

We are using three services, `ImageResizer`, `Language Translation` and `Virus Scanning`, to do the performance evaluation of ICAP, SOAP and our SEE. The `ImageResizer(IR)` processes the image bytes and reduces the image size by scaling its height and width. It has been implemented in Java. It is aimed at the users having limited bandwidth, those who have hand-held devices and to shorten the download time for large JPEG images. The `Language Translation(LT)`has been implemented by a wrapper, which uses the language translation service provided by free translation [10]. The `Virus Scanning (VS)` service has been implemented using the virus scanning service available on `www.openantivirus.com`. It checks for the presence of a virus on the requested document. The reason we choose these three services is the diversity of their input/output ratio. The input of `LT` is a URL but its response is almost of the same size as that of the original content, the input of `VS` is the original content but its response is either 'yes' or 'no', if the response is 'yes' then SEE generates a error page and sends it to the client and if its 'no' then the original content is sent to the client, and the input of `IR` is the original image but its response is a reduced form the original image. Our IR reduced the testing images we used for performance evaluation by 87%.

The execution time of these three services on the fixed web page is listed in Table 2. Note that the execution time of Language Translation is out of our control as it is dependent on the Internet speed and the free translation server.

IR	VS (html)	VS (image)	LT
345.0	177.3	133.0	**7858.5**

Table 2: Execution time of three services.

5.3 Overhead of SEE

We first evaluated the overhead of our Service Execution Environment. Figure 5 compares the overhead of SEE in five different cases. The readings are independent the call-out protocol being used to send the request. In the figure, No-SEE represents the case when SEE is not enabled, SEE-0 represents the case when there are no services in the user's properties file i.e the properties file of the user is empty, SEE-IR represents the case when IR service is invoked on the Request, SEE-VS represents the case when VS is invoked on the Request, and SEE-IR-VS represents the case when IR and VS services are invoked on the same request.

From the Figure 5 we can say that the overhead of SEE is acceptable when compared to the total response time for one client in Figure 7, it is less than 16% in case of the IR service and less than 18% in case of the VS service.

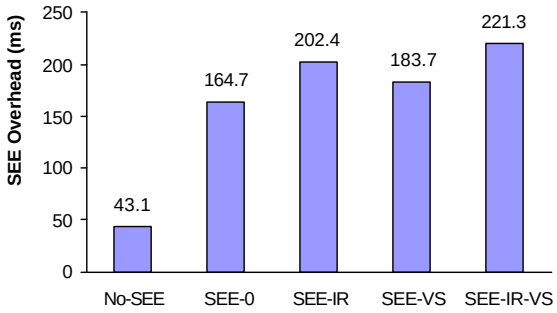


Figure 5: The execution overhead of Service Execution Environment.

5.4 Overhead of ICAP and SOAP Clients

The bars of each group (based on the three services) in Figure 6 shows the breakdown of the overhead of processing and communication within SEE, where T_r is the processing overhead of the Rule Engine, T_s is the request and reply time between the proxy cache and origin server, T_p represents the ICAP/SOAP Client overhead, and T_n is the ICAP/SOAP Server overhead. The left bars in each group is for ICAP and the right ones of SOAP. We found that the overhead of rule engine is independent of the call-out protocol and the service which reflects that the design of our SEE is stable. However we found a large overhead due to the ICAP/SOAP client and also a large variation in their overhead for different services.- On comparing the overheads of ICAP and SOAP clients we found that in most cases ICAP client out performs the SOAP client.

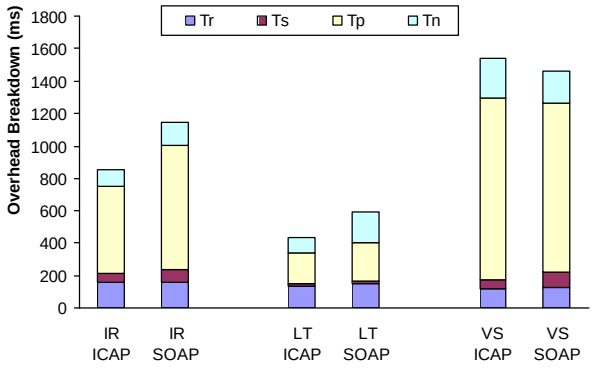


Figure 6: Breakdown of overhead of ICAP (left) and SOAP (right).

5.5 Benefit of ICAP and SOAP

After finding out the overhead of SEE and the overhead of ICAP and SOAP clients, we now provide the user-perceived latency in Table 3 when single and combination of services are invoked over ICAP and SOAP. The latency is defined as the difference between T_{16} and T_1. 'No' depicts the case when the properties file of the use is empty. We compared the latencies obtains for each service and combination of services with the 'No' case, and made the following conclusion: First, third party services may not always be good for us and second, if the service is running far away from the data flow, the performance may become 7 times slower. Here we argue that to obtain a good performance the service running on a remote machine should be as near as possible to the service execution environment.

After comparing the latencies obtained for ICAP and SOAP we find that there is not much difference in their performance but deploying a service over SOAP is more easier than doing it over ICAP, this is true especially in case of the legacy services. Based on the performance of ICAP and SOAP we believe that a service should be invoked remotely only if its a proprietary of someone or computing intensive, else its best to invoke it locally.

To compare the effort of implementing an ICAP or

Protocol	No	IR	VS (html)	VS (image)	IR+VS	LT	LT + VS
ICAP	387.7	990.2	665.6	772.2	1362.9	8180.8	8321.6
SOAP	387.8	1220.1	942.2	1046.6	1311.2	11062.2	7802.4

Table 3: Benefit of ICAP and SOAP

	IR		VS		LT	
Protocol	Client	Server	Client	Server	Client	Server
ICAP	175	331	175	284	175	224
SOAP	101	110	101	153	101	68

Table 4: The lines of codes for each service in the two protocols.

a SOAP client, we list the number of lines of code for ICAP and SOAP clients/servers in Table 4. We see that the implementation of a SOAP client/server requires less amount of coding work than the implementation of an ICAP client/server.

5.6 Scalability Analysis

Our last concern of the service execution environment is its scalability. We use a multithreaded client in Java, to create 1,2,4 or 8 clients and send requests to the SEE. The results of our evaluation are shown in Figure 7. Our observation from the figure is: First, the user-perceived latency is dominated by the service implementation and also on it being executed locally or remotely (as can be seen from the graph for LT) and, second the overhead of SEE is independent of the number of clients and the data to prove this will be provided in the final version of this paper.

5.7 Limitations

Currently, our service execution environment handles multiple services that are applied to the same data flow sequentially, therefore, the SEE overhead is proportional to the number of executed services, as shown in Figure 5. Obviously, this overhead can be optimized if the remote call-out protocol supports the combination of multiple requests or pipeline of multiple services. We argue that this is a necessary requirement for the call-out protocols and should be included in the OPES documents [5]. Secondly, the cooperation of multiple service execution environments is not discussed in this paper. However, in some circumstances the decision of executing some services in the downstream points along the data path, is dependent on the results of executing the services in the upstream points. Therefore, we plan to extend our execution environment to support the cooperation, such as information sharing, in the next step.

6 Related Work and Discussion

Our work in this paper was motivated by the two related research areas: open pluggable edge servies (OPES) [27] and distributed content adaptation [33, 12]. So, we discuss the related work in these two fields as follows.

In [27], IETF's Open Pluggable Edge Service working group proposes an environment to provide value-added services to the end-users, which motivates our work in this paper, but this paper focuses on the implementation of a service execution environment and the performance evaluation of ICAP and SOAP. Beck and Hofmann in [4] talked about a rule specification language for intermediary services i.e the IRML which can be used by the clients and/or content providers to configure the rules. But in this paper we argue that the service execution environment should provide a simple web interface that allows authorized parties (clients, network providers, content providers) to select the services they desire based on the name of the services (high level type description). This simplifies the process of configuring the environment and decouples the rule generation and rule processing, thus allowing more flexibility. In [3], Bell Lab has implemented a service execution environment prototype based on Apache Proxy Server and performed some preliminary analysis of the performance of their prototype. However, in their implementation the rule engine processes all the rules for each web transaction, our rule engine processes for each web transaction, the rules of only those services which are present in the properties file of the user made the web transaction, thereby optimizing the rule engine and decreasing its overhead.

There is a large amount of prior work on transcoding architectures [8, 13, 20, 22, 33], infrastructures for their effective deployment [11, 18, 21, 23, 24]. Those infrastructure allows the construction of *network-aware access paths* from application-specific component; these components cope with device and network mismatches by handling activities such as protocol conversion and content transcoding at sites best suited for them. Which share the same goal of the work in this paper. However,

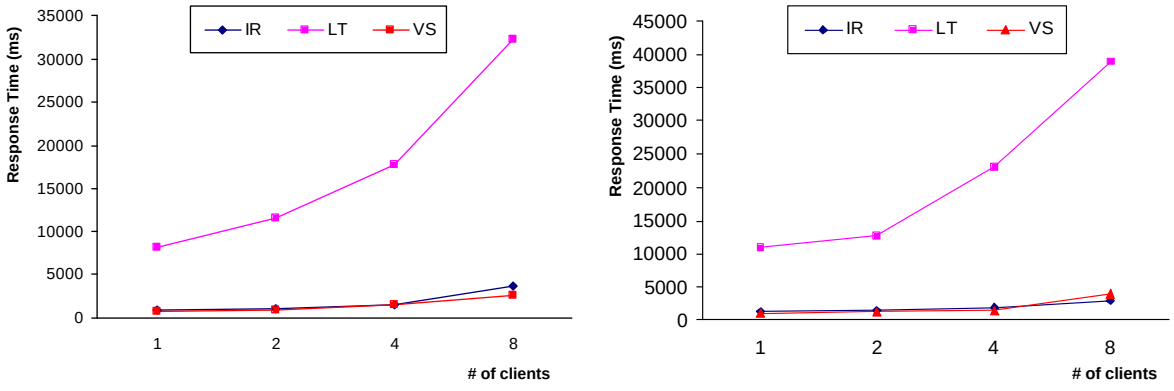


Figure 7: Scalability of the see execution environment (a) ICAP, (b) SOAP.

most of previous work in content adaptation are focus on infrastructure issues along the data path, and the adaptation functionality is conducted inside the proxy caches or edge. In this paper, we beleive that with the proliferation of Internet services, some of these services are computing intensive or proprietary code, which require to be executed in specific places. Therefore, our work focus on the execution environment that support different remote call-out protocols, which complements the previous work on content adaptation.

In summary, to the best of our knowledge, the performance evaluation presented in this paper is the first public results on comparing of ICAP and SOAP protocol for edge services. The service execution environment proposed in this project distinguishes itself from previous work by its ability to support multiple protocol bindings between service providers and proxy servers, scalable service management, efficient service composition support, as well as provides a simple interface to allow authorized clients to configure preferences.

7 Conclusions and Future Work

The paper proposes a novel service execution environment, which distinguishes itself from other work by its ability to support multiple protocol bindings (SOAP and ICAP) and the ease of rule specification. Further it allows secure registration of services, as well as provides a simple interface to allow authorized clients to configure preferences.

We have performed a thorough analysis to effectively describe the overheads when using the service execution environment. After comparing the latencies obtained for ICAP and SOAP we find that there is not much difference in their performance but deploying a service over SOAP is more easier than doing it over ICAP, this is true especially in case of the legacy services. Based on the performance of ICAP and SOAP we believe that a service should be invoked remotely only if its a proprietary of someone or computing intensive, else its best to invoke it locally.

Our future work includes integrating SEE into our ongoing CONCA proxy caches, extending our work to optimize the execution of multiple services within one execution environment, providing support for distributed service composition among multiple SEEs. The code of SEE will be public available soon at `http://mist.cs.wayne.edu`.

References

[1] Akamai Technologies Inc., `http://www.akamai.com/`.

[2] Apache Jakarta Project, `http://jakarta.apache.org`.

[3] A. Beck and M. Hofmann. Enabling the internet to deliver content-oriented services. *Proc. of the 6th International Workshop on Web Caching and Content Distribution (WCW'01)*, June 2001, `http://www.cs.bu.edu/techreports/2001-017-wcw01-proceedings/107_beck.pdf`.

[4] A. Beck and M. Hofmann. IRML: A rule specification language for intermediary services, work in progress, Nov. 2001, `http://www.ietf.org/internet-drafts/draft-beck-opes-irml-02.txt`.

[5] A. Beck, M. Hofmann, H. Orman, R. Penno, and A. Terzis. Requirements for OPES callout protocols, work in progress, Nov. 2001, `http://www.ietf.org/internet-drafts/draft-ietf-opes-protocol-reqs-03.txt`.

[6] M. Blumenthal and D. Clark. Rethinking the design of the internet: The end to end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 2002.

[7] Digital Island Corp., http://www.digitalisland.com/.

[8] A. Fox, S. Gribble, Y. Chawathe, and E. A. Brewer. Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Prespectives. *IEEE Personal Communication*, Aug. 1998, http://www.cs.washington.edu/homes/gribble/papers/adapt.ps.zip.

[9] A. Fox, S. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based Scalable Network Services. *Proc. of the 16th ACM Symp. on Operating Systems Principles*, Oct. 1997.

[10] Free Translation Inc., http://www.freetranslation.com.

[11] X. Fu, W. Shi, A. Akkerman, and V. Karamcheti. CANS: Composable, Adaptive Network Services Infrastructure. *Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS'01)*, pp. 135-146, Mar. 2001.

[12] X. Fu, W. Shi, and V. Karamcheti. Automatic deployment of transcoding components for ubiquitous, network-aware access to internet services. Tech. Rep. TR2001-814, Computer Science Department, New York University, Mar. 2001.

[13] S. D. Gribble and et al. The Ninja Architecture for Robust Internet-Scale Systems and Services. *Journal of Computer Networks* 35(4), Mar. 2001, http://www.cs.washington.edu/homes/gribble/papers/ninja.ps.gz.

[14] S. D. Gribble, M. Welsh, E.A.Brewer, and D. Culler. The MultiSpace: An Evolutionary Platform for Infrastructual Services. *Proc. of the 1999 Usenix Annual Technical Conf.*, June 1999.

[15] J. Haartsen. BLUETOOTH– The universal radio interface for ad hoc, wireless connectivitity. *Ericsson Review*, 1998.

[16] ICAP Protocol Group. ICAP: the internet content adaptation protocol, work in progress, Feb. 2001, http://www.i-cap.org/icap/media/draft-elson-opes-icap-01.txt.

[17] IRCache Project. A distributed testbed for national information provisioning, http://www.ircache.net/Cache/.

[18] E. Kiciman and A. Fox. Using Dynamic Mediation to Intergrate COTS Entities in a Ubiquitous Computing Environment. *Proc. of the 2nd Handheld and Ubiquitous Computing Conference (HUC'00)*, Mar. 2000, http://www.stanford.edu/~emrek/pubs/paths.huc2k.pdf.

[19] J. Kubiatowicz and et. al. OceanStore: An architecture for global-scale persistent storage. *Proc. of the ASPLOS'00*, Nov. 2000.

[20] R. Mohan, J. R. Simth, and C. Li. Adapting Multimedia Internet Content for Universal Access. *IEEE Transactions on Multimedia* 1(1):104–114, Mar. 1999.

[21] A. Nakao, L. Peterson, and A. Bavier. Constructing End-to-End Paths for Playing Media Objects. *Proc. of the OpenArch'2001*, Mar. 2001, http://www.cs.princeton.edu/nsg/papers/e2e.ps.

[22] B. D. Noble. *Mobile Data Access*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, May 1998, http://mobility.eecs.umich.edu/papers/diss.pdf.

[23] B. Raman, R. Katz, and A. D. Joseph. Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network. *Proc. of the Workshop on Mobile Computing Systems and Applications (WMSCA'00)*, Dec. 2000, http://www.cs.berkeley.edu/~bhaskar/iceberg/univ-inbox.pdf.

[24] P. Reiher, R. Guy, M. Yavis, and A. Rudenko. Automated Planning for Open Architectures. *Proc. of OpenArch'2000*, Mar. 2000, http://www.lasr.cs.ucla.edu/yarvis/Conductor/papers/Planning_OpenArch_s%hort.ps.

[25] W. Shi and V. Karamcheti. CONCA: An architecture for consistent nomadic content access. *Workshop on Cache, Coherence, and Consistency(WC3'01)*, June 2001.

[26] D. Tennenhouse and D. Wetherall. Towards an Active Network Architecture. *Computer Communications Review* 26(2), Apr. 1996, http://www.tns.lcs.mit.edu/publications/ccr96.html.

[27] G. Tomlinson, R. Chen, and M. Hofmann. A model for open pluggable edge services, work in progress, Nov. 2001, http://www.ietf.org/internet-drafts/draft-tomlinson-opes-model-00.txt.

[28] M. Tsimelzon, B. Weihl, and L. Jacobs. ESI language sepcification 1.0, 2000, http://www.esi.org.

[29] U. Varshney and R. Vetter. Emerging Mobile and Wireless Networks. *Communications of the ACM* pp. 73–81, June 2000.

[30] W3C Consortium. Simple object access protocol (SOAP) 1.1, 2000, http://www.w3.org/TR/SOAP/.

[31] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and performance in the denali isolation kernel. *Proc. of the Fifth USENIX Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[32] W3C XSL Working Group, http://www.w3.org/Style/XSL/.

[33] M. Yavis, A. Wang, A. Rudenko, P. Reiher, and G. J. Popek. Conductor: Distributed Adaptation for complex Networks. *Proc. of the Seventh Workshop on Hot Topics in Operating Systems*, Mar. 1999, http://lasr.cs.ucla.edu/reiher/papers/yarvis.ps.

[34] L. Zhang, S. Michel, K. Nguyen, A. Rosenstein, S. Floyd, and V. Jacobson. Adaptive web caching: Towards a new global caching architecture. *Proc. of the 3rd International WWW Caching Workshop*, June 1998, http://wwwcache.ja.net/events/workshop/25/3w3.html.