A Scalable Framework for the Support of Advanced Edge Services

Michele Colajanni[1], Raffaella Grieco[2], Delfina Malandrino[2], Francesca Mazzoni[1], and Vittorio Scarano[2]

[1] Dipartimento di Ingegneria dell'Informazione,
Università di Modena e Reggio Emilia, 41100 Modena, Italy.
{colajanni, mazzoni.francesca}@unimore.it
[2] Dipartimento di Informatica ed Applicazioni "R.M. Capocelli",
Università di Salerno, 84081 Baronissi (Salerno), Italy.
{rafgri, delmal, vitsca}@dia.unisa.it

Abstract. The Ubiquitous Web requires novel programming paradigms and distributed architectures for the support of advanced services to a multitude of user devices and profiles.
In this paper we describe a Scalable Intermediary Software Infrastructure (SISI) that aims at efficiently providing content adaptation and combinations of other complex functionalities at edge servers on the WWW. SISI adopts different user profiles to achieve automatic adaptation of the content according to the capabilities of the target devices and users.
We demonstrate SISI efficiency by comparing its performance against another framework for content adaptation at edge servers.

1 Introduction

The World Wide Web, thanks to its simplicity, visibility, scalability and ubiquity, is considered the inexhaustible universe of information available through any networked computer or device and is also considered as the ideal testbed to perform distributed computation and to develop complex distributed applications.

In the *Pervasive and Ubiquitous Computing* era the trend is to access Web content and multimedia applications by taking into account four types of important requirements: *anytime-anywhere* access to *any data* through *any device* and by using *any access network.* Nowadays, the existing and emerging wireless technologies of the 3^{rd} generation (GSM, GPRS and UMTS), involve a growing proliferation of new rich, multimedia and interactive applications that are available on the Web. On the other hand, these appealing services are requested from a growing variety of terminal devices, such as Desktop PC, pagers, personal digital assistants (PDAs), hand-held computers, Webphones, TV browsers, laptops, set top boxes, smart watches, car navigation systems, etc. The capabilities of these devices widely range according to different hardware and software properties. In particular, for mobile devices relevant constraints concern storage, display capabilities (such as screen size and color depth), wireless network connections, limited bandwidth, processing power and power consumption. These constraints involve several challenges for the delivery and presentation of complex personalized

applications towards these devices, especially if there is the necessity of guaranteeing specified levels of service.

The most simple solution (still used by many Web portals) is to follow the *"one-size-fits-all"* philosophy. The idea is to provide content that is specifically designed for Desktop PC, without taking care of the troubles that, for example, could affect mobile users.

The effective presentation of Web content requires additional efforts and in particular new computation patterns. Providing a tailored content in an efficient way for different client devices, by addressing the mismatch between rich multimedia content and limited client capabilities, is becoming increasingly important because of the rapid and continuous evolving of the pervasive and ubiquitous devices.

One of the current research trend in distributed systems is how to extend the traditional client/server computational paradigm in order to allow the provisioning of *intelligent* and *advanced* services. One of the most important motivation is to let heterogeneous devices access WWW information sources through a variety of emerging 3G wireless technologies. This computational paradigm introduces new actors within the WWW scene, the intermediaries [1, 2] that is, software entities that act on the HTTP data flow exchanged between client and server by allowing content adaptation and other complex functionalities, such as geographical localization, group navigation and awareness for social navigation [3, 4], translation services [5], adaptive compression and format transcoding [6–8], etc.

The major research in this area is how to extend the capabilities of the Web to provide content adaptation and other complex functionalities to support personalization, customization, mobility and ubiquity. The idea of using an intermediate server to deploy adapted contents is not novel: several popular proxy systems exist, which include RabbIT [9], Muffin [10], WebCleaner [11], FilterProxy [12] and Privoxy [13]. These systems provide functionalities such as compressing text, removing and/or reducing images, removing cookies, killing Gif animations, removing advertisement, java applets and javascripts code, banners, pop-ups, and finally, protecting privacy and controlling access.

The main objective of this paper is to present SISI, a flexible and distributed intermediary infrastructure that enables universal access to the Web content. This framework has been designed with the goal of guaranteeing an efficient and scalable delivery of personalized services. SISI adds to the existing frameworks two main novelties:

- *per user* profiles. Many existing proxies only allow just one system profile, which is applied to all requests, coming from any user. That is, all the requests involve the same adaptation services. SISI, instead, allows each user to define one (or more) personal profiles, in such a way that the requests of a user may involve the application of some services, and those of another user may involve the application of completely different services.
- high scalability, because these services are computationally onerous and the large majority of existing frameworks does not support more than few units of contemporary requests per second.

The rest of the paper is organized as following: in Section 2 we present SISI, an intermediary software infrastructure, whose main objective is to provide advanced edge

services by efficiently tackling the dynamics nature of the Web. To this end, we present the integration of a per-user profile mechanism into the SISI framework to dynamically allow different Web content presentations according to users's preferences. In Section 3 we present SISI advanced functionalities. Section 4 describes the workload model that we used to benchmark the SISI framework, whose results are shown in Section 5; finally, some remarks and comments will conclude the paper in Section 6.

2 Scalable Intermediary Software Infrastructure (SISI)

In this section we provide a description of the SISI architecture. This framework is based on top of existing open-source, mainstream applications, such as Apache Web server [14] and mod_perl [15]. The motivations are threefold: first of all, it will make our work widely usable because of the popularity of these products. Then, our results will be released as open source (by using some widely accepted open-source license) so that it will be available for improvements and personalizations to the community. Last but not least, Apache is a high quality choice, since it represents one of the most successful open-source projects (if not the most successful) that delivers a stable, efficient and manageable software product to the community of Web users.

2.1 SISI Overview

The main guidelines and objectives of the SISI project are described in [16]. The idea is to create a new framework that aims at facilitating the deployment of efficient adaptation services running on intermediate edge servers.

SISI framework uses a simple approach to assemble and configure complex and distributed applications from simple basic components. The idea is to provide functionalities that allow programmers to implement services without taking care of the details of the infrastructure that will host these services.

SISI provides a modular architecture that allows an easy definition of new functionalities implemented as building blocks in Perl. These building blocks, packaged into *Plugins*, produce transformations on the information stream as it flows through them. Moreover, they can be combined in order to provide complex functionalities (i.e. a translation service followed by a compression service). Thus, multiple Plugins can be composed into SISI edge services, and their composition is based on preferences specified by end users. Technically, SISI services are implemented as Apache handlers by using mod_perl.

An handler is a subroutine, implemented by using mod_perl, whose goal is to manipulate HTTP requests/responses. Since Apache has twelve different phases in its HTTP Request Life-cycle, it provides different *hooks* to have the full control on each phase. mod_perl provides a Perl interface for these hooks. In such a way Perl modules will be able to modify the Apache behavior (for example, a PerlResponseHandler configures an Apache Response object). Handlers can be classified according to the offered functionality, in different categories that is, Server life cycle, Protocols, Filters and HTTP Protocol. We used the last two to implement our handlers under Apache.

A detailed description of the functionalities of SISI modules is presented in [16].

3 SISI Advanced Functionalities

SISI is a modular architecture composed of basic building blocks applications that can be easily assembled to cooperate and provide complex functionalities.

SISI programmability is a crucial characteristics since it allows an easy implementation and assembling of edge services that can enhance the quality and the user perception of the navigation. Often the introduction of new services into existing networks is an expensive and time-consuming process. To simplify this task, SISI provides mechanisms to obtain a general-purpose programmable environment. Programming under existing intermediaries could involve difficulties in term of efficiency, generality or integration. For this reason SISI offers an execution environment, in which a compositional framework provides the basic components for developing new services, and a programming model is used to make this execution environment highly *programmable*. The SISI programming model provides APIs and software libraries (Perl language) for programming and deploying new services into the intermediary infrastructure.

Another important aspect of the SISI framework is the user and device profiling management. In particular, the administrator manages users' accounts, by adding or removing users from the system, resolving incorrect situations (for example, forgotten passwords), providing information about the allowed services for a given user. When a new user is added to the system, a default profile is automatically generated and he/she can modify the profile the first time he/she enters the system.

SISI approach in user profiling management is to explicit ask the user what he/she needs and use this information with a rule-based approach to personalize the content. In particular, users have to fill-out forms to create new profiles and to modify or delete existing ones. For example, when a user connects with a PDA, he/she could want his/her device to display only black and white images or not to be given images at all to save bandwidth. Through this services configuration, SISI is able to affect the adaptation of a given delivery context, and to change the user experience accordingly. Moreover, SISI allows a simple configuration mechanism to add, remove, enable or disable functionalities and a more complete configuration mechanism where service parameters can be specified by asking the user to fill-out Web-based forms.

In this context another important characteristics is the *hot-swap* of services composition i.e. the capability to load and execute different services according to different profiles at run-time without recompiling and restarting the software infrastructure.

SISI supports the deployment and un-deployment of advanced services, by making these tasks automatic and accessible through local and remote locations. Application deployment is an important system functionality that provides clients with an anytime-anywhere access to services and applications. By making the deployment an automatic task (i.e. wizard) it is possible to add new functionalities into the intermediary system without taking care of the complexity of the system itself.

Finally, SISI supports logging and auditing functionalities. The intermediary entity provides mechanisms to record security-related events (logging) by producing an audit

trail that allows the reconstruction and examination (auditing) of a sequence of events. The process of capturing user activities and other events on the system, storing this information and producing system reports is important to understand and recover from security attacks. Logging is also important to provide billing support, since services can be offered with different price models (flat-rate, per-request, per-byte billing options).

4 Workload Models and Testbed

Since studies on real traces show great differences, the difficulty of defining a "typical" workload model is a well known issue. Traces were collected form a real dynamic Web site, but they were modified in order to artificially stress the content adaptation process, so to emulate a worst case scenario. The number of HTML pages has been raised, so that adaptation occurs more frequently, many GIF images were substituted with animated GIFs in order to stress GIF de-animation process.

Requests are referred to a mix of content types consisting of images (49%), HTML documents (27%), others (24%). HTML resources typically contain embedded objects ranging from a minimum of 0 to a maximum of 25, with a mean value of 10. Embedded objects are images (GIFs and JPGs), CSS or SWF files. Animated GIF images are about 6% of the whole workload. These percentages tend, once again, to stress the content adaptation process. HTML pages also contain links to other pages, ranging from a minimum of 0 to a maximum of 25, with a mean value of 5.

The workload is characterized by small inter-arrival times for client requests.

To avoid possible non predictable network effects, the experiments were conducted in a LAN environment.

In order to test the SISI framework we set up a testbed composed of three nodes connected through a switched fast Ethernet LAN. One node, equipped with a Pentium 4 1.8GHz and 512MB RAM, running Gentoo Linux with kernel 2.6.11, ran an Apache Web server deploying the origin resources. Another node, equipped with a Pentium 4 2.4GHz and 1GB RAM, running Fedora Core 3 Linux with kernel 2.6.9, ran the application proxy, being it SISI rather than Muffin, deploying adapted contents. Finally a third node, equipped with a Pentium 4 1.8GHz and 1.5GB RAM, running Gentoo Linux with kernel 2.6.11, ran *httperf* [17] by D. Mosberger, which is is a tool for measuring Web server performance. It provides a flexible facility for generating various HTTP workloads and for measuring server performance.

5 Performance Evaluation

The SISI architecture is implemented on top of the Apache Web server software [14]. The prototype was extensively tested to verify its scalability and to compare its performance with Muffin [10], which is a Web HTTP proxy that provides content adaptation functionalities such as removing cookies, killing GIF animations, removing advertisements, adding, removing, or modifying any HTML tag, etc. It is a Java-based programmable and configurable proxy: new services or filters can be developed through a set of provided APIs and can be added at run time, using the provided graphical interface.

The first set of experiments focuses on the different response times the user gets when connecting to a proxy, instead of directly connecting to the origin Web Server. A second set of experiments aims at comparing SISI with another intermediary: Muffin. It is worth noting that SISI supports user authentication, while Muffin does not. Thus, we can expect that Muffin is advantaged with respect to SISI because of this lack. A third set of experiments, finally, aims at verifying SISI scalability by applying different services at the same time.

In all the experiments, the response time is referred to a whole page, including its embedded objects. That is, the response time represents the necessary time to completely download both the (possibly adapted) page content and all of its (possibly adapted) embedded objects.

5.1 Intermediary Overhead

In this section we evaluate the impact on the user response time given by the overhead of contacting a proxy adaptation server instead of the origin Web server. We configured SISI and Muffin so that they only forward resources to users, without applying any content adaptation. This is done in order to understand how much the user response time is affected by the use of an intermediary. We configured *httperf* to request 3000 pages, with the respective embedded objects. At a slow rate of 5 pages per second, the 90 percentile of user response time is 44 ms, 87 ms and 120 ms for the origin Web server, Muffin and SISI, respectively. The user response time contacting an intermediary is slightly less than two to three times the one obtained by contacting the origin Web server. It is worth noting that contacting an intermediary implies an increase of one hop, thus there is a slight network delay due to the additional three way handshaking, necessary to open the connection between the intermediary and the origin Web server.

Furthermore, SISI authenticates the user, thus there is an additional computational step, in order to satisfy the request, which possibly justifies a further increase in the user response time, with respect to Muffin.

To sum up, with a very slow rate and without any adaptation process, Muffin outperforms SISI, but the ranking will definitely change, as soon as we apply some content adaptation and/or increase the request rate, as we will see in the next subsections.

5.2 Performance Comparison of Two Intermediaries

In this section we aim to compare Muffin and SISI performance. It is worth noting that we could not find two services exactly performing the same adaptation process, but we may assume that Muffin Painter service is very similar to SISI RemoveLink service, from a computational point of view. That is, both of them parse the HTML content, search for some tags and rewrite them. Actually, Muffin Painter service can remove and/or modify colors and background images found in HTML documents, while SISI RemoveLink searches for `a href=""` tags and replaces them with plain text. Our claim is that, even though they do not perform the same tasks, the two services are computationally comparable.

We set up *httperf* to request 3000 pages with the respective embedded objects, at varying rates. Figure 1 and Table 1 show the 90 percentile of user response time when

Muffin or SISI, respectively, are contacted as intermediaries and apply the adaptation service. An X sign in Table 1 (as well as in the following) means the intermediary is overloaded.

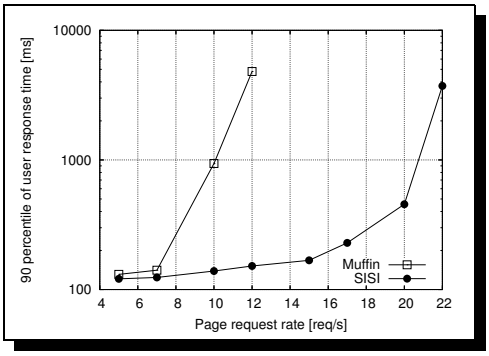

Fig. 1. 90 percentile of user response time as a function of the page request rate

Table 1. 90 percentile of user response time as a function of the page request rate

Page Request Rate [pages/s]	5	7	10	12	15	17	20	22
Muffin Painter service	131	141	938	X	X	X	X	X
SISI RemoveLink service	121	124	139	152	168	229	455	X

For Muffin we only report three page request rates because the system is overloaded with higher rates. SISI, instead, shows a better stability, with a 90 percentile of the user response time below half a second with a rate of 20 pages per second. Muffin shows a very poor stability: increasing the page rate from 5 to 7 brings to a 7.6% increment of the user response time, while passing from 7 to 10 pages per second, brings an increment of 565.25%. This is a clear sign that Muffin is overloaded. SISI performs much better: the increase is about 10% up to 15 pages per second, passing from 15 to 17 brings an increment of about 36%. When the page request rate further increases from 17 to 20 the user response time is nearly doubled, but still under half a second. Finally with a rate of 22 pages per second there is an increment of about 718%, which evidences an overloaded intermediary. To sum up, Muffin can sustain a rate of 7, while SISI up to 20 pages per seconds, nearly tripling the sustainable load.

5.3 SISI Scalability

GifDeanimate service. In this section we aim to evaluate SISI scalability. To this purpose, we choose SISI GifDeanimate service, which parses a Graphics Interchange Format (GIF) image into its component parts. The GifDeanimate service produces a de-

animated version of the original image, showing only its first frame. Our main objective is to save bandwidth in the delivery of Web pages with a lot of animated embedded images and to spare CPU cycles at the client device.[3]

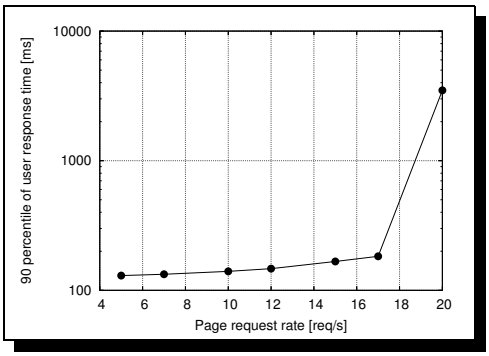

Fig. 2. SISI: 90 percentile of user response time with GifDeanimate service.

In this case also, we set up *httperf* to request 3000 pages with the respective embedded objects, at varying rates. Figure 2 and the third row of Table 2 show the 90 percentile of user response time when SISI GifDeanimate service is applied to the requests.

We can roughly draw the same observations as for the previous RemoveLink service, with a limited increase in the user response time up to a rate of 17 pages per second. The fact that the system gets overloaded earlier (with a rate of 20 pages per second, instead of 22) is a sign that GifDeanimate is a heavier service than RemoveLink.

Composition of services. Finally, we want to test SISI scalability when more services are applied one after the other to the same request. To this goal, we set up SISI to perform both RemoveLink and GifDeanimate services on each request. In this experiment we have nearly one third of all the requests being adapted.

In this case also, we set up *httperf* to request 3000 pages with the respective embedded objects, at varying rates. Figure 3 and Table 2 show the 90 percentile of user response time when SISI GifDeanimate and Remove Link services are applied to the requests.

From Table 2 we can notice that the influence of the RemoveLink service is very limited on the whole user response time. If we compare the third and the fourth row of Table2 for the first two rates the difference is very low. Increasing the rate brings to slightly bigger differences up to 17 pages per second, when the system is getting overloaded.

[3] Muffin seems to have a similar service to SISI GifDeanimate, which is called AnimationKiller, but Muffin adaptation is far away from the adaptation performed by SISI, which keeps only the first frame of the animated GIF. This is why we do not compare SISI with Muffin results.

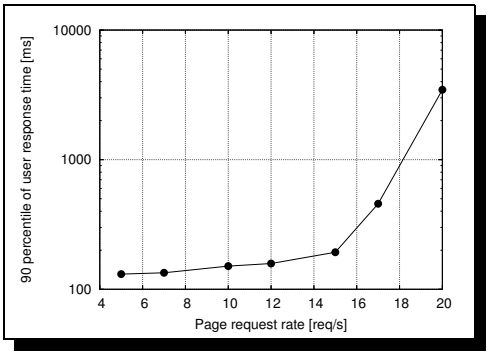

Fig. 3. SISI: 90 percentile of user response time with GifDeanimate and RemoveLink services.

Table 2. SISI: 90 percentile of user response time with RemoveLink and GifDeanimate services.

Page Request Rate [pages/s]	5	7	10	12	15	17	20
RemoveLink	121	124	139	152	168	229	455
GifDeanimate	130	133	140	147	167	183	X
Both services	131	134	151	158	193	457	X

6 Conclusions and Future Work

In this paper we presented a Scalable Intermediary Software Infrastructure (SISI), whose main goal is to create a framework that aims at facilitating the deployment of adaptation services running on intermediate edge server on the WWW. It provides a modular architecture that allows an easy definition of new functionalities implemented as building blocks in Perl.

Users can define one or more personal profiles, in such a way that the requests of a user may involve the application of some services, and those of another user may involve the application of completely different services.

The prototype was extensively tested. In particular our experiments demonstrate that the response times the user gets when connecting to a proxy, instead of directly connecting to the origin Web Server are increased from two to three times by proxies which do not or do use user authentication, respectively. When comparing SISI with another intermediary (Muffin), we found that SISI provides much better performance. In particular, it can nearly triple the sustainable load. Finally, SISI is able to efficiently deliver adapted content on a per-user basis in a scalable way.

Our goal in the very next future is to deploy SISI in a distributed network environment and to port it to different operating systems.

Acknowledgments

This research work was financially supported by the Italian FIRB 2001 project number RBNE01WEJT “WEB–MiNDS” (Wide-scalE, Broadband MIddleware for Network Distributed Services). `http://web-minds.consorzio-cini.it/`

References

1. Barrett, R., Maglio, P.P.: Intermediaries: An approach to manipulating information streams. IBM Systems Journal **38** (1999) 629–641
2. Luotonen, A., Altis, K.: World-Wide Web proxies. Computer Networks and ISDN Systems **27** (1994) 147–154
3. Barrett, R., Maglio, P.P.: Adaptive Communities and Web Places. In: Proceedings of 2^{th} Workshop on Adaptive Hypertext and Hypermedia, HYPERTEXT 98., Pittsburgh (USA), ACM Press (1998)
4. Calabrò, M.G., Malandrino, D., Scarano, V.: Group Recording of Web Navigation. In: Proceedings of the HYPERTEXT'03, ACM Press (2003)
5. Almaden Research Center, I.: Web Based Intermediaries (WBI) (2004) http://www.almaden.ibm.com/cs/wbi/.
6. Ardon, S., Gunningberg, P., LandFeldt, B., Y. Ismailov, M.P., Seneviratne, A.: MARCH: a distributed content adaptation architecture. Intl. Jour. of Comm. Systems, Special Issue: Wireless Access to the Global Internet: Mobile Radio Networks and Satellite Systems. **16** (2003)
7. Hori, M., Kondoh, G., Ono, K., Hirose, S., Singhal, S.: Annotation-Based Web Content Transcoding. In: Proceedings of the 9^{th} International World Wide Web Conference, Amsterdam (The Netherland), ACM Press (2000)
8. WebSphere: IBM Websphere Transcoding Publisher (2005) http://www-3.ibm.com/software/webservers/transcoding.
9. Olofsson, R.: RabbIT proxy (2005) http://rabbit-proxy.sourceforge.net/.
10. Boyns, M.: "Muffin - a filtering proxy server for the World Wide Web". (2000) http://muffin.doit.org.
11. WebCleaner: A filtering HTTP proxy. (2005) http://webcleaner.sourceforge.net.
12. McElrath, B.: FilterProxy in perl (2002) http://filterproxy.sourceforge.net/.
13. Burgiss, H., Oesterhelt, A., Schmidt, D.: Privoxy Web Proxy. (2004) http://www.privoxy.org/.
14. The Apache Software Foundation: Apache (2005) http://www.apache.org.
15. The Apache Software Foundation: mod_perl (2005) http://perl.apache.org.
16. Grieco, R., Malandrino, D., Mazzoni, F., Scarano, V., Varriale, F.: An intermediary software infrastructure for edge services. In: Proceedings of the 1^{st} Int. Workshop on Services and Infrastructure for the Ubiquitous and Mobile Internet (SIUMI'05) in conjunction with the 25^{th} International Conference on Distributed Computing Systems (ICDCS'05). (2005)
17. Mosberger, D., Jin, T.: httperf - a tool for measuring web server performance. SIGMETRICS Performance Evaluation Review **26** (1998) 31–37 http://www.hpl.hp.com/research/linux/httperf.