Pers Ubiquit Comput (2010) 14:301–320
DOI 10.1007/s00779-009-0232-9

ORIGINAL ARTICLE

MIMOSA: context-aware adaptation for ubiquitous web access

Delfina Malandrino · Francesca Mazzoni · Daniele Riboni · Claudio Bettini · Michele Colajanni · Vittorio Scarano

Received: 26 October 2008 / Accepted: 9 March 2009 / Published online: 17 April 2009

Abstract The ubiquitous computing scenario is characterized by heterogeneity of devices used to access services, and by frequent changes in the user's context. Hence, adaptation according to the user's context and the used devices is necessary to allow mobile users to efficiently exploit Internet-based services. In this paper, we present a distributed framework, named MIMOSA, that couples a middleware for context-awareness with an intermediary-based architecture for content adaptation. MIMOSA provides an effective and efficient solution for the adaptation of Internet services on the basis of a comprehensive notion of context, by means of techniques for aggregating context data from distributed sources, deriving complex contextual situations from raw sensor data, evaluating adaptation policies, and solving possible conflicts. The middleware allows programmers to modularly build complex adaptive services starting from simple ones, and includes tools for assisting the user in declaring her preferences, as well as mechanisms for detecting incorrect system behaviors due to a wrong choice of adaptation policies. The effectiveness and efficiency of MIMOSA are shown through the development of a prototype adaptive service, and by extensive experimental evaluations.

Keywords Context-awareness · Adaptation · Transcoding

D. Malandrino (✉) · V. Scarano
Dipartimento di Informatica ed Applicazioni "R.M. Capocelli", Università di Salerno, 84084 Fisciano, Salerno, Italy
e-mail: delmal@dia.unisa.it

V. Scarano
e-mail: vitsca@dia.unisa.it

F. Mazzoni · M. Colajanni
Dipartimento di Ingegneria dell'Informazione, Università di Modena e Reggio Emilia, 41100 Modena, Italy
e-mail: mazzoni.francesca@unimore.it

M. Colajanni
e-mail: colajanni@unimore.it

D. Riboni · C. Bettini
Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, 20135 Milano, Italy
e-mail: riboni@dico.unimi.it

C. Bettini
e-mail: bettini@dico.unimi.it

1 Introduction

Until a few years ago, access to the Web was restricted to users sitting in front of their desktop computer with a wired Internet connection. However, the advent of handheld devices and high-bandwidth wireless networks has radically changed the situation. In fact, nowadays users can access the Web in virtually any context, using a myriad of devices (such as PDAs, smartphones, pagers, and car navigation systems) having different capabilities and input/output modalities. As a consequence, users on the move can now take advantage of Internet-based services for getting information that are useful for accomplishing their tasks, such as searching interesting locations, checking travel timetables, and getting maps and directions. For this reason, a hot research topic in the human–computer interaction area is the investigation of intelligent techniques for tailoring the Web experience to the context of mobile users (see, e.g., [2, 26, 30, 33, 44, 45]).

Unfortunately, this radical change in the Web usage has not been appropriately supported by service providers. As a matter of fact, even if in the last years several techniques and delivery platforms have been proposed for supporting tailoring of Web contents to different devices and usage conditions (e.g., mobility), experience tells us that the vast majority of Internet services does not support any form of adaptation. Hence, being formatted for presentation in wide, high resolution screens, most Internet services are practically not accessible from small-screen handheld devices. Moreover, their user interfaces—that are intended to be accessed through a mouse—are not well suited for devices having different input modalities, such as stylus, softkeys, or directional pad. Performing Web content adaptation on the client side can be unfeasible on many resource-constrained handheld devices, and it is in general costly in terms of network consumption; hence, a pragmatical approach to this problem consists in the use of intermediary adaptation components (called edge servers) to transcode contents on the basis of the context. The idea is, therefore, to implement and deploy any computing transformation and adaptation algorithms at edge servers, programmed to perform aggressive computation and storage on behalf of clients. In particular, the context data that must be considered for effectively adapting Internet services to the user situation is wide and provided by different distributed sources, and includes, among the others, device capabilities, network characteristics, user current activity and preferences, as well as information about possible user disabilities.

Recently, several systems for Web content transcoding (e.g., [9, 24, 35, 41, 42, 46]), as well as several architectures for context-awareness (e.g., [10, 16, 20, 22, 29, 43]) have been proposed; however, an integrated solution for Web adaptation based on a comprehensive notion of context has not yet been devised. In particular, existing solutions consider only a restricted set of context data (typically, the capabilities of the used device), and lack mechanisms for retrieving and dynamically deriving a comprehensive set of context data. We claim that a wider set of context data must be considered to effectively perform adaptation in ubiquitous computing scenarios; e.g., since handheld devices equipped with sensors such as accelerometers are becoming more and more common (e.g., Apple's *iPhone* and OpenMoko's *Neo 1973*), techniques for deriving the user's situation (e.g., activity) on the basis of sensor data must be integrated into the adaptation process.

In this paper, we propose a novel framework, named MIMOSA, that couples a middleware for context-awareness [6] with an intermediary architecture for Web transcoding [11] in order to provide a comprehensive solution to the addressed problem. The main contributions of our work are:

- A framework for ubiquitous Web adaptation based on a comprehensive notion of context;
- Tools for assisting the user in declaring her preferences, as well as mechanisms for detecting incorrect system behavior due to a wrong choice of adaptation policies;
- A working implementation of the proposed framework, and extensive performance evaluations.

The rest of this paper is organized as follows. In Sect. 2, we discuss related work. Section 3 presents an overview of the MIMOSA framework. Section 4 describes the software implementation. Section 5 presents the adopted mechanisms for user support. Section 6 illustrates a prototype context-aware service that takes advantage of MIMOSA. Section 7 presents the experimental evaluation. Finally, conclusions are drawn in Sect. 8.

2 Related work

The issue of adapting Web contents to the network context and device capabilities has been extensively addressed in the last years. Proposed solutions can be broadly classified into three main categories: client-side, server-side, and intermediary-based adaptation systems.

In the client-side approach, adaptation is demanded to the client device itself; typically, adaptation is performed by the Web browser. We believe that client-side adaptation is not appropriate for network- and resource-constrained devices, since (a) it does not determine a reduction in network consumption, and (b) it burdens the client device with computationally intensive tasks such as transcoding of multimedia resources.

Server-based approaches essentially consist in adding content adaptation services to traditional Web server functionalities. Multimedia content transformation is typically generated off-line at content creation time, often involving a human designer to hand-tailor the content for the specific requirements of a few classes of devices (e.g., desktop computer, personal digital assistant, and phone). The multiple variants of the same resource are stored on the server, and selected at the time of the service request by analyzing the HTTP request headers. Usually, the most appropriate Web page markup is generated on-the-fly by applying XSLT transformations to XML-based descriptions of the page content and structure. As an example, the personalization scheme of the IBM WebSphere Portal [24] is based on the creation of Web pages and services using XDIME, a proprietary XML-based device-independent markup language. Depending on the specific device, XDIME contents are transformed by proper predefined XSLT stylesheets into the most appropriate format (e.g., WML, XHTML Basic, etc.), evaluating policies that take into account the capabilities of the particular device that issued the request. The framework also includes a repository of mobile device profiles describing the capabilities of a broad range of terminals. Similar solutions are provided by other well-known application servers like BEA Weblogic and OracleAS Wireless. As an alternative, also on-the-fly multimedia content adaptation and delivery has been proposed [35].

Server-side adaptation systems offer an efficient solution to the addressed issue; however, the practical consideration that—at the time of writing—the vast majority of Web portals does not provide any form of adaptation brought us to prefer the intermediary-based approach. As a matter of fact, pure intermediary-based approaches do not need any form of cooperation by the external Web servers, since adaptation is performed by systems that lay between the server and the client nodes.

Intermediary-based approaches to Web content adaptation have been the focus of several research proposals. Some systems (e.g., RabbIT [42], Muffin [9], WBI [3], WebCleaner [53], Privoxy [41]) provide content adaptation without considering any user/device profile, but only allowing simple services configuration mechanisms. More sophisticated architectures support location- and context-aware adaptation of Internet services. An example of middleware that supports location-aware applications for mobile users is Nexus [15]. The user interface, running on mobile devices, is used to interact with the Nexus platform, which is composed by communication facilities (for accessing information sources through wireless networks), distributed data management (for multiple representation of spatial data), and sensor elements (for system positioning). The focus of the Nexus platform is on location-aware services, while MIMOSA addresses a wider set of context data (including location). MIMOSA considers elements such as the available bandwidth, battery level, and CPU usage, to dynamically tailor Web contents to the capabilities of the client devices, whereas only a subset of these constraints is addressed by NEXUS. The user interface, running on the mobile device, provides support to adapt devices with different levels of computing power, memory, and network connection, but at the same time, it exhibits all drawbacks of the client-side approaches.

A software infrastructure aimed at simplifying the development of self-adaptive applications is presented in [21]. The proposed infrastructure provides high-level abstractions for representing, reasoning with, and exploiting context information and preferences for the adaptation of services. Even if the underlying context model of this infrastructure shares some common features with ours, the approach to adaptation is different, since in MIMOSA adaptation is performed by intermediary proxies.

The SCaLaDE [5] middleware (as well as its predecessor CARMEN [4]) aims at supporting mobility-enabled Internet services. SCaLaDE is able to dynamically adapt contents in response to modifications concerning context and location awareness, user preferences, and available resources, without affecting the implementation of the service application logic. To realize a separation between the service management and the service application logic, SCaLaDE exploits a policy-driven approach, by taking decisions according to events triggered at the provision time. Policies are expressed in the Ponder language [13], which turns out to be a good choice for the class of policies used in this middleware; on the other hand, the policy language adopted by the MIMOSA CONTEXT-AWARENESS FRAMEWORK is well suited for adaptation rules, since it is extremely efficient and has mechanisms for solving conflicts (more details will be given in Sect. 3.1.2).

MASHA [46] is a middleware for adaptation of Web sites driven by user and device profiles. In this system, customization is based on content selection, performed on the basis of users' interests and device capabilities. Users' interests are calculated by statistical analysis of Web navigation logs. In our work, we use a similar technique for detecting adaptation misbehaviors. In addition, MIMOSA allows users to explicitly express policies regarding their adaptation preferences, while in MASHA adaptation directives are defined server-side. Another difference with respect to our work is that MASHA requires Web sites to cooperate in the customization process, since Web resources must be semantically annotated in order to calculate users' interests and to perform content selection. On the other hand, our adaptation approach does not require any form of cooperation by external Web sites, making it feasible for virtually any resource published on the Web.

The VESPER [37] architecture provides functionalities to adapt contents according to network and client capabilities, by also controlling the quality of the provided services in the case of changes in network conditions. The adaptation component realizes these tasks by analyzing profiles describing personal preferences of users. While VESPER only takes into account a restricted set of context parameters, that is, device capabilities and network status conditions, MIMOSA instead is able to address a wider set of context data, as an example socio-cultural context data. Finally, while both systems provide policies to dynamically infer user preferences, only MIMOSA is able to provide a mechanism for their conflict resolution.

MoCA [47] is an intermediary architecture whose main goal is to provide collaborative applications in mobile environments. The MoCA framework was designed for supporting these applications with functionalities like caching management, transcoding, and data compression. MoCA collects context data from client systems and network infrastructure, in order to trigger appropriate adaptations. In particular, MoCA allows to develop and customize proxies according to specific needs. This choice simplifies the development of distributed and context-aware applications that need reconfigurations according to users' needs and changing conditions in mobile environments. While MIMOSA is mainly addressed to the adaptation of existing Web contents, MoCA is mainly intended to support collaborative applications in mobile networks. Another difference with respect to our work is that MoCA only takes into account a restricted set of context parameters (i.e, device capabilities, network status, and indoor location). On the other hand, MIMOSA supports a much wider set of context data including, among the others, particular type of disabilities (e.g., color blindness, physical disabilities), geographic location, and current activities. These data can be profitably exploited for providing a more effective adaptation of Web resources to the users' situation. Moreover, while in MoCA, the adaptation directives are set by the application developer, MIMOSA allows users to express their preferences regarding the adaptation services by means of policies.

The MobiGATE framework, **Mob**ile **GAT**eway for the **A**ctive deployment of **T**ransport **E**ntities [55], exploits the use of intermediaries to facilitate the adaptation across wireless and mobile environments. Its main goal is to address the challenges related to the implementation and the deployment of service entities, called streamlets, that provide content adaptation functionalities and that can be reconfigured according to changes in mobile environments. The MobiGATE framework exhibits a clear distinction between computation and composition, by envisioning an execution environment for scheduling streamlets and a coordination environment for the interaction and the composition of streamlets. Each streamlet encapsulates the application logic and is specialized for a specific task, such as scaling/dithering of images in a particular format, translation among different data formats, and so on. The composition and coordination of the streamlets is managed by using a coordination language that provides rich constructs to support the definition of compositions, with constrained type validation and checking. Like MIMOSA, the main goal of the MobiGATE framework is to provide an effective solution for the adaptation of Internet services according to different context conditions. In addition to the easy composition and programming of adaptation services, MIMOSA also provides a mechanism to retrieve a comprehensive description of the user context and to derive the most appropriate adaptation service according to the context data and adaptation policies. Finally, MIMOSA provides mechanisms for conflict resolution and for assisting users in defining profiles and customizing services.

Finally, the adaptation to cope with the variability of user devices is also addressed by the framework described in [27]. This framework is composed by three main components: the client tier, the context-aware service tier, and the repository service tier. Front-end modules running on clients are able to send to the servers context profiles to inform them about their capabilities. The context-aware service tier includes at least a CAAS server that supports the migration of agent, execution of back-end modules, and application adaptation (i.e., transcoding applications). To adapt functions of applications to the context of user devices, this framework uses CC/PP and UAProf, while in the MIMOSA framework, we extended CC/PP vocabularies for describing features like users' personal information, interests, locations, actions, surrounding environments. They also developed an attribute-based component decision algorithm to choose the components suitable for the context of the user's devices. In MIMOSA, we have defined policies to determine the value of profile attributes on the basis of the values of other profile attributes. These policies are expressed in terms of a logic programming language. Finally, context data retrieved from these distributed profile managers are merged before policy evaluation, and possible conflicts are solved before invoking the requested services with the derived service parameters, whereas neither conflicts nor consistency between CC/PP profiles and back-end modules have been contemplated in the described framework.

3 The MIMOSA framework

In this section, we describe MIMOSA, a distributed framework whose main goal is to adapt Web contents to the heterogeneity of networks and client devices as well as to variation in users' environments. Figure 1 shows the MIMOSA architecture, its components and how data flow through them for any user request. In order to provide the system with the user's identification and with the information required to retrieve distributed context data and policies, the client request is intercepted by a LOCAL PROXY that runs on the user device and adds custom fields into the HTTP headers. In Step 1, the request is forwarded to the CONTEXT AWARENESS MODULE. Then, context data are retrieved from distributed CONTEXT SOURCES by the DISTRIBUTED CONTEXT RETRIEVAL module, and aggregated by the AGGREGATOR module (Step 2). These data, together with the adaptation policies, are provided to the REASONER engine, which is in charge of performing policy evaluation. The resulting aggregated set of context data—which includes the list of content adaptation services to be applied, as well

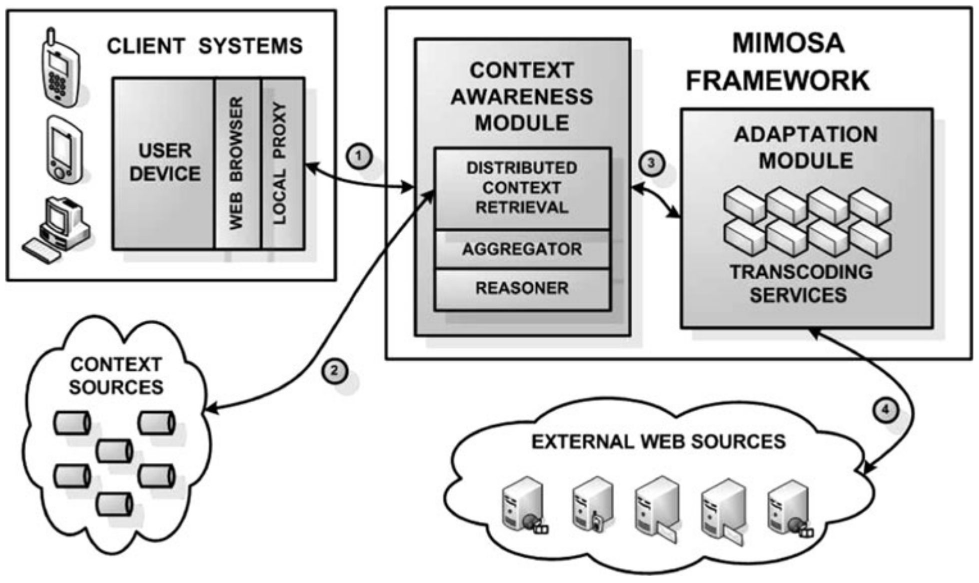


Fig. 1 The MIMOSA architecture and the data flow among its components

as their parameters—is inserted into the HTTP headers. Then, the request is sent to the ADAPTATION MODULE (Step 3), which retrieves the requested resource from the external Web sources (Step 4), and invokes the content adaptation services according to the aggregated context data. Finally, the adapted resource is sent back to the user's device.

3.1 Context aggregation and reasoning

In order to retrieve a comprehensive description of the context describing the user request, and to derive the most appropriate adaptation service parameters on the basis of context data and adaptation policies, MIMOSA takes advantage of a customized version of the CARE [6] middleware for context awareness. In particular, context data aggregation and reasoning (including conflict resolution) is performed by means of logic programming techniques and ontological reasoning. An exhaustive description of the adopted technical solutions is outside the scope of this paper, and can be found in [1] and [7]; however, in this section, we recap the basic mechanisms of context aggregation and reasoning.

3.1.1 Overview

Figure 2 shows an overview of the middleware for context aggregation and reasoning. We call profile a subset of context information collected and managed by a certain entity. Profiles are managed by three entities, namely: the user with her devices, the network operator with its infrastructure, and the service provider. Each entity has a dedicated profile manager (called UPM, OPM, and SPPM, respectively) to handle its own context data. The middleware includes ontology services for managing and reasoning with socio-cultural context data. Adaptation and personalization parameters are determined, at the time of the service request, by policy rules defined by both the user and the service provider, and managed by their respective profile managers. For instance, users can declare policies that determine the activation of certain adaptation services on the basis of context (e.g., "downgrade image quality if the bandwidth is low"). The CONTEXT PROVIDER module is in charge of calculating the aggregated context information that will be used by the application logic for the adaptation services. In particular, it retrieves context data from the profile managers, and evaluates adaptation policies solving possible conflicts arising among context data and/or policies.

3.1.2 Representation of context data and policies

Essentially, context data are represented adopting the CC/PP [28] specification, and can possibly contain references to ontological classes and relations. However, for the sake of this paper, we can consider profiles as sets of attribute/value pairs. Each attribute semantics is defined in a proper vocabulary, and its value can be either a single value, or a set/sequence of single values.

Well-known CC/PP vocabularies such as UAProf [39] provide attributes for describing device capabilities. However, in order to support a very accurate adaptation of Internet services with respect to the users' situation, a wider set of context attributes is needed. For this reason, we have extended CC/PP by defining new vocabularies for describing features such as users' personal information, interests, location, current action, surrounding environment. Moreover, we have defined new vocabularies for representing the users' preferences with respect to the

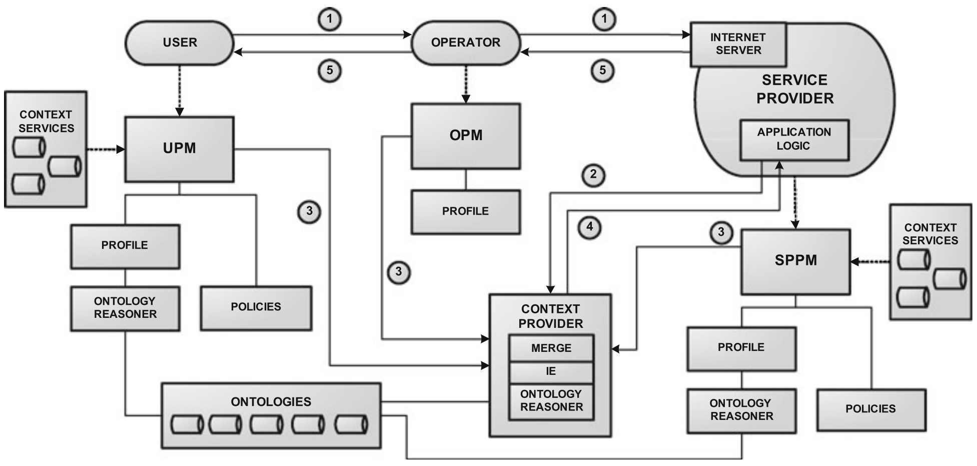


Fig. 2 Middleware for context aggregation and reasoning

Fig. 3 An excerpt of the CC/PP vocabulary for representing preferences regarding the “DeleteImage” service. Here, we only show the vocabulary entry for the “height” attribute parameter, as the same coding is required for the “width” parameter

```
<rdf:RDF
  xmlns:rdf = '&ns-rdf;'
  xmlns:rdfs = '&ns-rdfs;'
  xmlns:ccpp = '&ns-ccpp;'
  xmlns:mimosa_services = '&ns-mimosa_services;'
  xmlns:mimosa_params = '&ns-mimosa_params;'>

<!-- ***** Component: MIMOSA_DeleteImage ***** -->
<ccpp:Attribute rdf:about='&ns-mimosa_params;activate'>
   <rdfs:label xml:lang="en">Attribute: activate</rdfs:label>
   <rdfs:domain rdf:resource='&ns-mimosa_services;MIMOSA_DeleteImage'/>
   <rdfs:range  rdf:resource='&ns-ccpp;string'/>
   <rdfs:comment xml:lang="en">
      Description:  The service activation
      Type:         String
      Examples:     "on", "off"
   </rdfs:comment>
</ccpp:Attribute>

<ccpp:Attribute rdf:about='&ns-mimosa_params;height'>
   <rdfs:label xml:lang="en">Attribute: height</rdfs:label>
   <rdfs:domain rdf:resource='&ns-mimosa_services;MIMOSA_DeleteImage'/>
   <rdfs:range  rdf:resource='&ns-ccpp;string'/>
   <rdfs:comment xml:lang="en">
      Description:  The lower bound for the 'height' parameter.
                    Greater values determine the image deletion.
      Type:         String
      Examples:     "178", "800"
   </rdfs:comment>
</ccpp:Attribute>
```

activation parameters of each adaptation service provided by MIMOSA. Figure 3 shows an excerpt of the vocabulary for representing preferences regarding the “DeleteImage” service.

Policies are logical rules that determine the value of profile attributes on the basis of the values of other profile attributes. Hence, each policy rule can be interpreted as a set of conditions on profile data that determine a new value for a profile attribute when satisfied. Policies are expressed by means of a restricted logic programming language for which a linear-complexity inference engine exists. Experimental results have shown that the evaluation of policy rules is executed in few milliseconds [7].

Example 1 Consider the case of a *FilterImg* service for image transcoding, which determines the type of adaptation to apply on the basis of network conditions and available memory on the user's device. These parameters are determined by the evaluation of the following policy rules:

R1: "**If** *AvBandwidth* $\geq$ 128 kbps **And** *Bearer* = 'UMTS'
Then Set*NetSpeed* = 'high"'
R2: "**If** *NetSpeed* = 'high' **And** *AvMem* $\geq$ 4 Mb
Then Set*FilterImg:Downgrade* = 'off"'
R3: "**If** *NetSpeed* = 'high' **And** *AvMem* < 4 Mb
Then Set*FilterImg:Downgrade* = '20%"'
R4: "**If** *NetSpeed!*='high' **Then Set***FilterImg:Downgrade*='50%"'

The value of the NetSpeed attribute is determined by rule R1 on the basis of the current available bandwidth (AvBandwidth) and Bearer. Rule R2 deactivates the image quality downgrading service in the case the available bandwidth is high and the device has a sufficient amount of free memory. Rule R3 instructs the adaptation service to slightly downgrade the image quality when the available bandwidth is high but the device runs out of memory. Finally, rule R4 is used to strongly downgrade image quality when the network is congested.

We point out that the rules shown above are rather naïve, and used just for the sake of this example in order to illustrate the basic system behavior.

3.1.3 *Context aggregation, policy evaluation, and conflict resolution*

Once the CONTEXT PROVIDER has obtained profile data from the other profile managers, at first this information is passed to the MERGE module (Fig. 2), which is in charge of merging profiles. Conflicts can arise when different values are provided by different profile managers for the same attribute. For example, suppose that the OPM provides for the AvBandwidth attribute a certain value x, while the SPPM provides for the same attribute a different value y, obtained through some probing technique. In order to resolve this type of conflict, the CONTEXT PROVIDER has to apply a resolution rule at the attribute level. These rules (called profile resolution directives) are expressed in the form of priorities among entities, which associate to every attribute an ordered list of profile managers. We recall that UPM, OPM, and SPPM are abbreviations for the user, network operator, and service provider profile managers, respectively.

Example 2 Consider the following profile resolution directives, set by the provider of the transcoding service cited in Example 1:

PRD1: *setPriority AvBandwidth = (OPM, SPPM, UPM)*
PRD2: *setPriority FilterImg:Downgrade = (UPM, SPPM)*

In PRD1, the service provider gives highest priority to the network operator for the *AvBandwidth* attribute, followed by the service provider and by the user. The absence of a profile manager in a directive (e.g., the absence of the OPM in PRD2) states that values for that attribute provided by that profile manager should never be used. The conflict described above is resolved by applying PRD1. In this case, the value x is chosen for the available bandwidth. The value y would be chosen in case the OPM did not provide a value for that attribute.

The semantics of priorities actually depends on the type of the attribute. A more in-depth discussion of the merge mechanism can be found in [7].

Once conflicts between attribute values provided by different profile managers are resolved, the resulting merged profile is used for evaluating policy rules. Since policies can dynamically change the value of an attribute that may have an explicit value in a profile, or that may be changed by some other policies, they introduce non-trivial conflicts. The intuitive strategy is to assign priorities to rules having the same head predicate on the basis of its profile resolution directive. Hence, rules declared by the first entity in the profile resolution directive have higher priority with respect to rules declared by the second entity, and so on. When an entity declares more than one rule with the same head predicate, priorities are applied considering the explicit priorities given by that entity. Details on rule conflict resolution can be found in [7].

Example 3 Consider the set of rules shown in Example 1 and profile resolution directives shown in Example 2. Suppose that R2 and R3 are declared by the user, and R4 is declared by the service provider. Since the user declared two rules with the same attribute in the head, she has to declare an explicit priority between R2 and R3. Suppose the user gives higher priority to R2 with respect to R3. Since the UPM has higher priority with respect to the SPPM, according to the profile resolution directive regarding *FilterImg:Downgrade* (i.e., PRD2 in Example 2), if $p(R)$ is the priority of rule R, we have that:

$$p(R2) > p(R3) > p(R4)$$

The intuitive evaluation strategy is to proceed, for each attribute A, starting from the rule having the predicate $A()$ in its head with the highest priority, and continuing considering rules on $A()$ with decreasing priorities till one of them fires. If none of them fires, the value of A is the one obtained by the MERGE module on A, or *null* if such a value does not exist.

3.2 Provisioning of adaptation services

To provide adaptation services, MIMOSA exploits many internal components (derived from the SISI framework

[11]) to intercept user requests, fetch the origin Web resources and apply contents adaptation, if required. These components are part of the CONTENT ADAPTATION MODULE within the MIMOSA framework.

The CONTENT ADAPTATION MODULE provides many already implemented content adaptation services, that may be classified in three main categories: accessibility services, annoyance filtering services and personalized ubiquitous computing services.

The first category includes services whose main goal is to promote Web accessibility and improve the navigation on the Web for users with disabilities. Examples include services that add a toolbar containing the LINK attributes on top of each HTML page, or a numeric Access Key to any link in a Web page in such a way to make it accessible through a simple combination of keyboard keys, ALT+Access Key+Return, and finally services that modify the structure, by reorganizing links in a table, or the presentation of a Web page, by modifying, for example, text color and size, to make Web content more accessible for users with visual disabilities (i.e low and color deficiency vision).

The second category includes a set of services whose main goal is to get rid of particularly annoying abuse during the navigation on the Web. They provide functionalities for removing advertisement, banners, JavaScript code, for disabling unsolicited pop-up windows, and so on. In addition, they also provide mechanisms for protecting users by avoiding some other users to steal their identities or malicious software to track and collect their personal information.

The third category includes services that provide functionalities for context-aware adaptation and personalization. For example, it includes services for geo-localization or services that are able to provide alternate contents for Web resources before their delivery to end users (the goal is to change the behavior of the origin Web server according to the capabilities of the requesting devices through the content selection mechanism) [14, 19]. Other examples for matching both client capabilities and users' preferences include functionalities for image color depth reduction, resizing, quality downgrade, color to grey conversion, colorblindness filtering [23] (to modify any color in Web pages, by increasing contrast and lightness, in order to make them accessible for users with such a disability).

Content adaptation services, developed through the Perl programming language, are very simple to implement as programmability represents one of the main strengths of the CONTENT ADAPTATION MODULE. The CONTENT ADAPTATION MODULE programming model, as intermediate level between the underlying system and the provided functionalities, has been designed to provide a transparent support for a quick development and an easy deployment of new content adaptation services. By exploiting the CONTENT ADAPTATION MODULE programming model, based on APIs and internal functions, programmers can focus on the design of the functionality, and its application logic, without taking care of any issue of the underlying infrastructure that will host these services (i.e. scalability, profile management, authentication, and so on).

The CONTENT ADAPTATION MODULE also provides a very efficient mechanism to chain adaptation services so that complex services can be easily obtained by chaining many simple services. For instance, a translation service from French to English might be chained to a service of image quality reduction and resizing to match both network available bandwidth and device capabilities besides user's preferences.

4 Software architecture

In this section, we present the software implementation of MIMOSA, and the protocols adopted for the communications among its distributed modules.

4.1 Implementation

The overall software architecture of MIMOSA is shown in Fig. 4. In the following, we illustrate the software implementation of the main modules of MIMOSA, outlining their tasks and the relationships with other modules. We also briefly motivate our choices of languages/frameworks in MIMOSA in order to combine efficiency (MIMOSA works on the HTTP request/response path and must not impact on the user's perceived latency) and extensibility required by a dynamic environment composed by heterogeneous client devices, networks and services.

4.1.1 Client-side modules

The LOCAL PROXY on the client system is the application that adds custom fields to the HTTP request headers, thus providing the CONTEXT PROVIDER with the user's identification, and with the references of her user profile manager and network operator profile manager. The modules executed on the user device are developed using C# for the .NET (Compact) Framework. A command-line proxy is also available for Linux clients and it is a customized version of the well-known Squid Web proxy [49].

4.1.2 Context-awareness module

With regard to the CONTEXT AWARENESS MODULE, we have chosen Java as the preferred programming language; however, the most computational intensive algorithms have

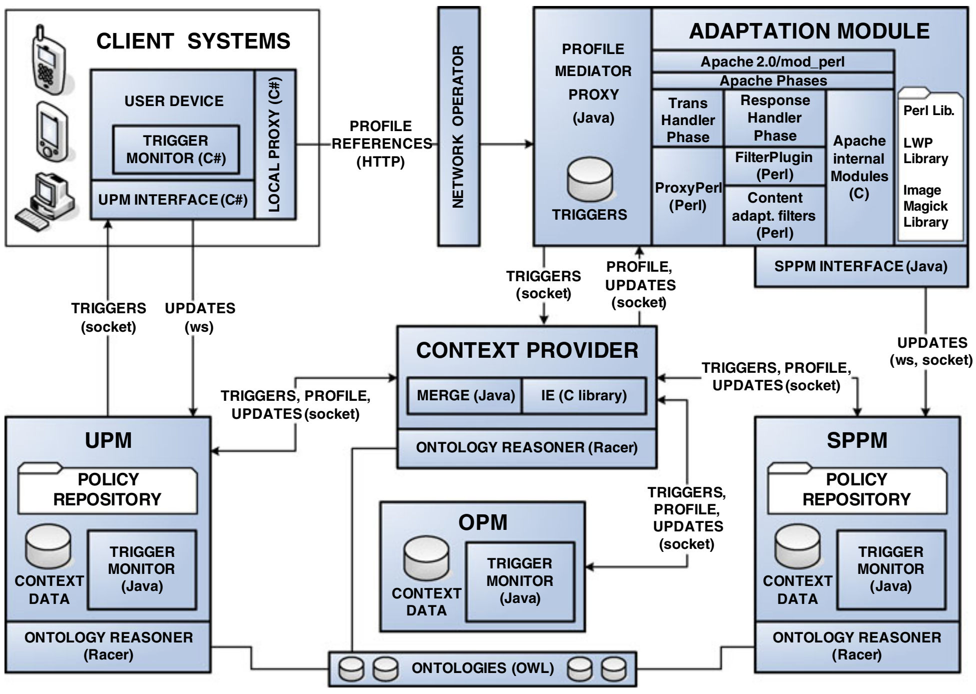


Fig. 4 The software architecture of MIMOSA

been developed in C, and integrated into the corresponding modules using Java Native Interface. The PROFILE MEDIATOR PROXY (PMP) is a server-side Java proxy that is in charge of intercepting the HTTP requests from the user's device, and of communicating the user's profile (retrieved from the CONTEXT PROVIDER) to the ADAPTATION MODULE. In order to relieve the modules of MIMOSA from the burden of parsing RDF documents, CC/PP profiles are represented by means of Java classes. Similarly, adaptation policies are serialized into Java objects according to specifically defined classes.

Context data retrieved from the distributed profile managers are merged before policy evaluation by a devoted software module; since our conflict resolution algorithm is not computationally intensive, the MERGE module has been developed in Java. On the other hand, the evaluation of policies expressed in a logic programming language can pose serious performance issues. For this reason, we have developed an ad-hoc INFERENCE ENGINE for our restricted logic programming language, using the C programming language. As illustrated in [7], policy evaluation with this inference engine is particularly efficient—its complexity being linear with the policy set size—and our ad hoc engine outperforms an optimized reasoner for a logic programming language having higher expressivity than ours.

The profile managers are developed in Java; context data and policies are stored by the profile managers into ad-hoc repositories that make use of the MySQL DBMS [38].

4.1.3 Adaptation module

The ADAPTATION MODULE has been developed on top of Apache Web Server (v. 2.0) [50] and mod_perl (v. 2.0) [34]. This software architecture consists of different components whose main goals are: intercept the HTTP requests coming from the PMP, fetch the requested Web pages from the external Web sources and apply transformations on them according to the directives obtained by the CONTEXT AWARENESS MODULE, forward this data to other internal components or, if no other adaptation is required, forward the adapted Web page back toward the client.

The main strengths of the ADAPTATION MODULE are programmability, extensibility and efficiency, as new functionalities can be easily programmed and deployed without performance degradation since the robustness of the Apache Web server.

In the Apache server programming environment, each HTTP request is processed in sequential phases, and at each phase different decisions can be taken about the request. Therefore, in addition to its robustness and its wide popularity, the Apache Web server was the best solution for the implementation of the MIMOSA's ADAPTATION MODULE components also because of its flexibility in taking decisions based on the current request. These components, because of their Perl implementation, are integrated into Apache through the mod_perl mechanism. mod_perl is an interface to the Apache C API and its main goal is to ensure a quick and easy implementation of Web applications. Since with mod_perl all phases of the Apache Request lifecycle can be accessed and controlled, we implemented the Adaptation Module components as mod_perl “handlers”, that, on the other hand, are able to provide hooks responsible for the processing of the specified phase.

From an architectural point of view, the Adaptation Module consists of two main categories of components, that we denote as: (a) core modules, that provide the internal functions for accepting HTTP requests, fetching Web contents, manipulating incoming and/or outgoing HTTP headers, forwarding the response to the client, and (b) service modules that is, a set of filters that encapsulate the adaptation application logic. In particular, the Adaptation Module consists of two core components, namely the ProxyPerl and the *FilterPlugin* components, and a set of provided filters (that belong to the service modules category). These components, their characteristics and their main functionalities are described below.

ProxyPerl component. The ProxyPerl component, registered in thePerlTransHandler phase, intercepts requests coming from the Context Awareness Module, and by using Perl libraries (i.e., LWP User Agent) and internal APIs, fetches the requested Web page from the origin server and forward it to the next *handler* whose hook has been registered in the Apache Request lifecycle. By registering the *ProxyPerl* component in this early phase of the HTTP Request Apache lifecycle (before the request has been associated with a particular file name), we are able to ensure that it will be invoked for any user request, and that it will be able to apply specific actions about URIs (i.e., blocking URIs redirect to third party servers) or specific transformation on the HTTP incoming/outgoing headers (i.e., removing cookies in the HTTP responses coming from third party servers).

FilterPlugin component. The *FilterPlugin*component is registered in thePerlOutputFilterHandler phase and its main goal is to check which handlers have to be invoked during a transaction. Since different content adaptation filters are available and preloaded in the main memory (by specifying thePerlModule directive in the main Apache configuration file) and can be customized according to user preferences, the *FilterPlugin*component has to activate only the right filter (or sequence of filters) selected according to context information and user preferences.

Adaptation Filter components. This category includes all functionalities to implement content adaptation service modules. Filters are implemented via handlers in thePerlResponseHandler phase. In addition to the provided MIMOSA service modules (whose categorization is described in Subsect. 3.2), new services can be easily developed and deployed due to the programmability and the extensibility of the Adaptation Module.

The Adaptation module exhibits a flexible and extensible environment in which a compositional Perl-based framework is used to provide the basic components to develop new content adaptation services. By providing a specific programming model and a set of internal APIs and software libraries, new service modules can be quickly prototyped and coded. This programming environment include:

- **APIs for processing HTTP requests/responses**. The LWP::UserAgent [12] is a class implementing a World-Wide Web user agent in Perl. The ProxyPerl component uses this library to fetch the requested Web pages from the origin servers by also allowing specific configurations (i.e.,timeouts ,user agent, user credentials, and so on). It also offers APIs for manipulating HTTP requests/responses headers.
- **APIs for image transformations**. The Adaptation module implements several service modules for image transformation by using PerlMagick [40], an object-oriented Perl interface to ImageMagick [25]. We implemented service modules for image resizing and quality downgrade to match device display capabilities, and for color transformations, to match preferences for users with visual disabilities. In particular, the Color-Blind filter, implements algorithm that modifies background and foreground colors in HTML pages and recolors embedded images (also animated GIF images), in order to make more recognizable the red/green contrast for dichromatic users [23].
- **APIs for HTML parsing**. For a quick and effective stream-oriented filtering of HTML, we implemented a Perl-based parsing library. We implemented general methods for searching links, images, scripts in a Web page, by allowing both extraction and transformation functionalities. The extraction allows us to take very useful information from an HTML document with minimal programmer efforts: text extraction, link extraction, checking, and so on, while the transformation task includes URL rewriting, HTML cleanup, and removal functionalities.
- **APIs for *service* modules programming**. We implemented a set of APIs for implementing functionalities starting from simple building blocks, developed in Perl language. These building blocks, that derive from implemented superclasses, are packaged into services and produce transformations on the information stream as it flows through them.

The implementation of a new filter requires three basic steps: (a) writing the filter (i.e, its application logic) in Perl language, (b) adding the definition of the new handler in the *FilterPlugin* component to dynamically allow its invocation, and finally (c) writing the filter configuration files (to configure the filter, it is necessary to write simple HTML and XML files, see Fig. 5). After that, the new filter is added to the Apache pool of handlers and can be invoked on a transaction according to the directives derived by the CONTEXT AWARENESS MODULE.

4.2 Communication protocol

Since efficiency is a fundamental requirement of the MIMOSA framework, we have adopted efficient protocols for the communication between those modules that need to exchange messages at the time of the service request. On the other hand, we have preferred Web Services for non time-critical communications, such as updates of context data that are executed in advance to the service requests.

In order to allow the evaluation of adaptation policies, the client system is in charge of providing to MIMOSA those information that are necessary for obtaining an exhaustive description of the current context. Since in a mobile computing scenario, the client system can possibly be connected to a wireless network, our choice has been to minimize the communication between the client system and MIMOSA. For this reason, the client system only communicates to MIMOSA those data that allow MIMOSA to obtain context data and adaptation policies from the distributed profile managers (that are expected to be hosted on wired nodes). That data are inserted as custom fields into the HTTP request headers, as shown below:

```
Upm: <upm_IP>:<upm_port>
Opm: <opm_IP>:<opm_port>
User-Id: <userid>
```

The <upm_IP> and <upm_port> correspond to the IP address and port of the user profile manager interface that provides those context information that is managed by the user and her devices. Similarly, the <opm_IP> and <opm_port> identify the interface provided by the network operator profile manager.

Once received this message, the PMP checks if a session for that user is present. If this is the case and if the session is still valid, the PMP retrieves the context data that it has stored locally, and forwards the request to the ADAPTATION MODULE after having added context data into custom HTTP headers. HTTP request headers have the following form:

```
voc1#comp1|voc2#attr2: "<val>"
voc3#comp3|voc4#attr4: "<val_1>";"<val_2>";...;"<val_n>"
```

The custom header fields correspond to context data, that in our framework are represented in the CC/PP language. In the above notation, voc1 refers to the vocabulary the component belongs to; comp1 is the ID of the component containing the attribute; voc2 refers to the vocabulary the attribute belongs to; and attr2 is the ID of the context attribute. Context values are enclosed into double quotes, and are separated by semicolons if the attribute value is a set or sequence of values. Context data and their corresponding values are then parsed by the ADAPTATION MODULE, and used for customizing the service.

Instead, in the case the user session is not present or has expired, the PMP is in charge of requesting the context data to the CONTEXT PROVIDER module. Hence, the PMP uses a socket-based communication for requesting context data to the CONTEXT PROVIDER, specifying the user ID and the references to her network operator profile manager and user profile manager. As explained before, context data are represented as objects belonging to classes that encode CC/PP profiles; context data are exchanged among the distributed modules by means of the socket-based binary serialization of Java objects. This protocol is more efficient than Java Remote Method Invocation and Web services.

Similarly, the communication between the CONTEXT PROVIDER and the profile managers is based on the socket-based binary serialization of Java objects. In addition to context data, the profile managers also communicate the

Fig. 5 An excerpt of the XML file representing information about the "FilterImg" service module provided by MIMOSA

```
<?xml version="1.0"?>
 <config>
    <param name="FilterImg_colorBlind" value="off" />
    <param name="FilterImg_downGrade" value="off" />
    <param name="FilterImg_dither" value="off" />
    <param name="FilterImg_whiteBlack" value="off" />
    <param name="FilterImg_resize" value="100" />
    <param name="FilterImg_activate" value="off" />
</config>
```

adaptation policies to the CONTEXT PROVIDER. Since profile updates are generally executed asynchronously with respect to the service request, efficiency is not an issue; hence, the profile managers adopt Web services for allowing their corresponding entities to keep up-to-date context data and policies.

5 Support for user-centric adaptation

User-friendliness is of paramount importance for services that adapt their behavior on the basis of users' needs and expectations. As a matter of fact, inexperienced users can be frustrated when the system behaves unexpectedly, apparently contradicting their preferences. Hence, our system provides various tools for assisting the user in:

- declaring their preferences,
- detecting inappropriate adaptation behaviors, and
- suggesting new preferences (in the form of policy rules) for adapting the service according to the user context.

5.1 Basic mechanism

As anticipated in Sect. 3.1.3, the adaptation parameters are determined at the time of the user request by evaluating user and service provider policies against the current context data.

Since the user and the service provider can possibly declare conflicting rules (i.e., rules that determine different adaptation parameters based on the same context), a conflict resolution strategy is needed. In order to minimize the user intervention in the customization of the adaptation services, our choice is to give the highest priority to the rules declared by the user, who can override default rules declared by the service provider. In this way, the user of the system is provided with a default set of policy rules that are declared by domain experts for effectively adapting Internet services on the basis of context data such as network connectivity, device capabilities, device status (e.g., quantity of free memory, installed applications), and data about the service itself. Then, the user can start to use the adaptation services without the need of specifying her preferences; when she realizes that the service adaptation does not fit her preferences in a particular case, she can change the system behavior by declaring a new policy rule.

5.2 Supporting user policy specification

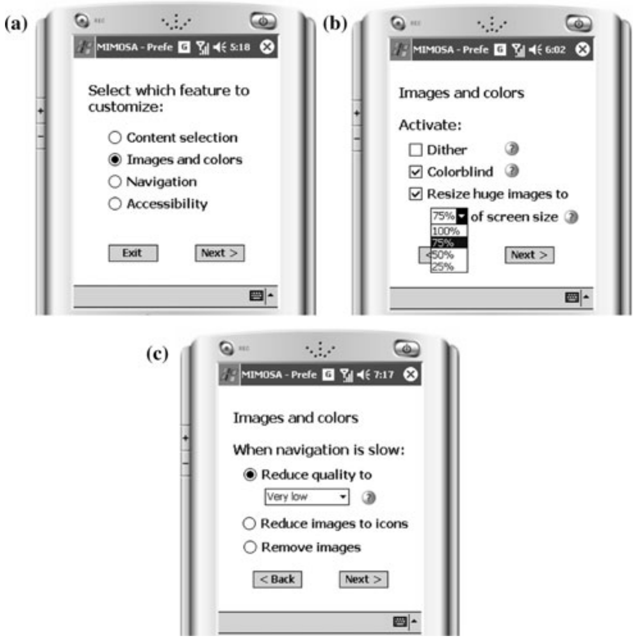
Of course, policies expressed by means of the formal language shown in Example 1 (Sect. 3.1.2) are not intuitive for the final users of the system. For this reason, users are provided with a *wizard* and Web interfaces to manage their preferences regarding the activation and parameters of content adaptation services.

The wizard assists the user step-by-step in the declaration of new preferences. The wizard allows the user to declare policies that control the activation and parameters of eight adaptation services, on the basis of various context data (e.g., connection type, available memory, battery level). The wizard is a C# application for the .Net framework [32], and communicates via Web Services to the user profile manager for updating the set of policy rules (for instance, the Main menu displayed by the wizard, is shown in Fig. 6a).

Example 4 Suppose that a user is browsing through a PDA, and wants to set her preferences regarding the adaptation services. When she launches the wizard, the main menu is shown, which allows the user to select the type of feature she wants to adapt (Fig. 6a).

If the user chooses to set her preferences regarding the adaptation of images and colors, a form is shown, which allows her to activate dithering, color transformation for colorblind people, and to resize huge images (Fig. 6b). Suppose that the user activates the colorblind service, and chooses to resize huge images (i.e., images that exceed the screen size) to 75% of her device screen size. Hence, the wizard automatically generates the following policy rules:

R5: "**Set***FilterImg:ColorBlind* = 'on"'
R6: "**If** *DeviceType*='PDA' **Then** **Set***FilterImg:Resize* = '180x240"'


(a)
MIMOSA - Prefe 5:18
Select which feature to customize:
Content selection
[✓ Images and colors]
Navigation
Accessibility
Exit Next >

(b)
MIMOSA - Prefe 6:02
Images and colors
Activate:
Dither
[✓ Colorblind]
[✓ Resize huge images to]
75% of screen size
100% 75% 50% 25%
Next >

(c)
MIMOSA - Prefe 7:17
Images and colors
When navigation is slow:
[✓ Reduce quality to]
Very low
Reduce images to icons
Remove images
< Back Next >

Fig. 6 Wizard for supporting specification of user policies

Note that the value of the *FilterImg:Resize* parameter in rule *R*6 is automatically customized by the agent on the basis of the actual screen size of the device she is currently using.

On the next step, the wizard allows the user to declare her preferences for reducing bandwidth consumption in the case the navigation is slow. The user has the options to whether reduce image quality (to medium, low, or very low), reduce images to icons, or completely remove them. Suppose that the user chooses to reduce image quality to very low in the case of slow navigation; hence, the agent generates the following policy rules:

R7: "**If** *NetSpeed*='slow' **Then** **Set***FilterImg:Downgrade* = '30%"'

R8: "**If** *NetSpeed* = 'slow' **Then** **Set***FilterImg:Monochrome* = 'yes"'

Rule *R*7 states to apply a 30% image quality downgrading when the navigation is slow, and rule *R*8 states to transform images to black and white on the same condition.

Once the user has specified her preferences, the agent updates the user policies on her remote profile manager.

In the case the device cannot run the .Net framework, policies can be managed by means of Web interfaces, like the one shown in [18].

5.3 Detecting inappropriate adaptation behaviors

Even if the system is provided with a default set of adaptation policies chosen by domain experts, and users can define new rules for specifying their particular preferences, in some cases the system can misbehave due to a choice of policy rules that is inappropriate for a particular context. For instance, a policy rule declared by a user can determine the request of high resolution media even if the available bandwidth is low. When such an event happens, in most cases the inexperienced user is unable to recognize the reason of the system misbehavior, and simply draws the conclusion that the adaptation system "does not work".

In order to automatically recognize system misbehaviors, the user is provided with an agent running on her devices, which monitors the client resources (e.g., available memory, battery level) and the Web navigation logs. These data are statistically analyzed at run time in order to detect problems, such as memory overload, and long response times due to low bandwidth. In such a case, the agent queries the CONTEXT PROVIDER in order to collect the policies that drove adaptation. On the basis of this data, the agent can recognize the actual policy rule that determined the incorrect behavior, in order to assist the user in correcting the problem through a wizard.

The agent architecture is shown in Fig. 7. The agent performs a statistical analysis of data provided by the local proxy (i.e., the navigation logs, that provide information regarding the time taken to download Web pages), and by the device itself, such as the memory available at run-time, the remaining battery lifetime, and the CPU usage. Critical events are detected by proper algorithms that are periodically executed. The algorithms are in charge of detecting events such as slow navigation, device out of memory, and CPU overload. As an example, the algorithm for detecting slow navigation evaluates the navigation logs of the last ten Web pages downloaded by the browser, and draws the conclusion that the navigation is slow in the case the average download time per page is higher than one minute, and more than 70% of Web pages have requested more than 1 min to be downloaded.

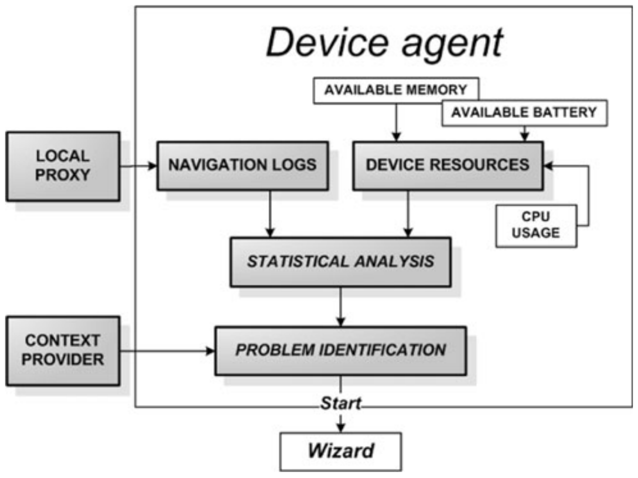
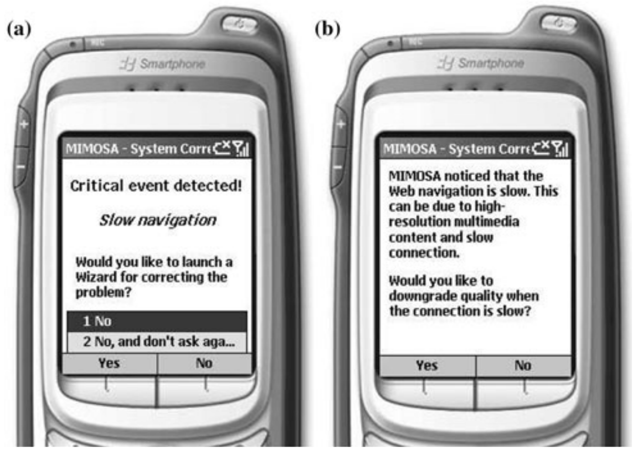


Fig. 7 Device agent for detection of inappropriate adaptation behavior

When a critical event is detected, the event is communicated to the agent module that is in charge of recognizing the policy rule that determined the system misbehavior. This module is capable of mapping events to the content adaptation services *S* that may have caused it. For instance, slow navigation can be determined by the deactivation of those services that reduce image quality (or completely remove them) in the case of slow bandwidth. Hence, the agent queries the CONTEXT PROVIDER in order to retrieve the set of rules that determined the parameters of the *S* service; analyzing the fired rules, the agent can recognize the rule that determined the deactivation of the service. Once the rule is recognized, the agent automatically launches a wizard that assists the user in the correction of the problem.

5.4 Assisting the user to correct the system behavior

Once the rule that determined the problem is recognized, the user is provided with a wizard to correct the adaptation behavior. Since the majority of Web users does not have a clear knowledge of the internal behavior of their client



Critical event detected!

Slow navigation

Would you like to launch a Wizard for correcting the problem?

MIMOSA noticed that the Web navigation is slow. This can be due to high-resolution multimedia content and slow connection.

Would you like to downgrade quality when the connection is slow?

Fig. 8 Wizard for assisting the user in the correction of the system behavior

systems, as well as of the Web infrastructure, it is important to motivate the reasons that caused the problem (e.g., by explaining that it is not possible to view high resolution videos when connected through a WAP phone), and to suggest a possible solution.

Example 5 Suppose that a given user declared (through the Web interface) a policy rule for requesting high quality media when she is connected through a smartphone, independently by any other condition (e.g., available bandwidth and memory). When connected through a GPRS connection to a Web portal rich of multimedia content such as images and sound, the response times become very long. Analyzing the Web navigation logs, the agent recognizes the problem, and prompts the user asking whether she is satisfied with her current experience, or wants to start a wizard for changing the adaptation behavior (see Fig. 8a; note that the user can also disable prompting by the agent in the case she finds it annoying). If the user chooses to start the wizard, she is provided with a brief description of the problem, and the agent suggests to modify the rule, in order to restore the default adaptation behavior determined by the service provider policies (Fig. 8b).

In the case the user decides to cancel the rule, the wizard updates the user's rule set on her profile manager through a Web service invocation.

6 The *GEOAWARE* prototype service

The interest for geolocalized services provided on the Internet is witnessed by the proliferation of applications for route planning, points of interests, and maps (e.g., [8]). For instance, the smartphone version of the *Google Maps* [17] service provides mobile users with facilities for calculating routes, visualizing maps, and looking for points of interest. Even though these applications provide an effective support to mobile users, they are somehow limited by the lack of an underlying framework for context awareness. As a matter of fact, mobility emphasizes the need of context analysis in order to adapt the service to network conditions, device capabilities, availability of resources. In particular, location-awareness in the above-mentioned applications is rather primitive, since generally the user is asked to manually specify the service parameter (e.g., her location).

In order to evaluate the effectiveness of MIMOSA in the provisioning of location- and context-aware applications, we have implemented a prototype service named *GeoAware*. This service is addressed to mobile users equipped with a mobile device and a GPS receiver. The main goal of this service is to provide a map with information about both the current location of the user and the locations (expressed as physical addresses) appearing on the Web page she is currently viewing.

We now describe the main steps performed by the GeoAware service. At first, MIMOSA context awareness module retrieves the GPS coordinates of the user from her profile manager, and communicates them to the content adaptation module by inserting a proper header in the client request. The GeoAware service parses the requested Web page and, by applying regular expressions-based analysis, matches all standard U.S. addresses [52]. When an address is recognized, GeoAware adds a hyperlink on the Web page, highlighted by a particular icon (see Fig. 9a) to let the user know that she can view the respective map.

When the user selects one of the hyperlink icons, GeoAware invokes the geocoder.us service [31] to obtain the coordinates of the respective address. Geocoder.us is a public service providing free geocoding of addresses in the United States, and relies on Geo::Coder::US [48], an open-source Perl module available from CPAN.

After having obtained the address coordinates, GeoAware builds a query string and issues it to the U.S. Census Bureau *TIGER Map Server* [51], which provides public-

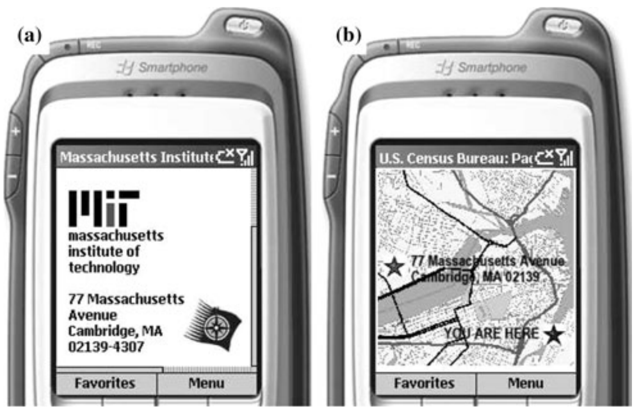

(a)
(b)
Smartphone
Massachusetts Institut
massachusetts institute of technology
77 Massachusetts Avenue Cambridge, MA 02139-4307
Favorites
Menu
U.S. Census Bureau: Pa
77 Massachusetts Avenue Cambridge, MA 02139
YOU ARE HERE
Favorites
Menu

Fig. 9 The GeoAware service

domain, customized U.S. maps. Figure 9 b shows the map obtained from the address in Fig. 9a. We represent the current user position with a blue star with a label YOU ARE HERE, while the position corresponding to the clicked address is represented by a red star labeled with the address itself. Exploiting the other adaptation services (e.g. an image resizing service), the map is properly tailored to device capabilities and available bandwidth.

As a final remark, we point out that we did not evaluate the user response times of the GeoAware service, since it involves the activation of services that are deployed by remote servers. This could bring to possibly unreproducible results because of both the response times of the remote servers and the load status of the network between those servers and the MIMOSA framework.

7 Performance evaluation

In order to evaluate the efficiency and effectiveness of our system in speeding-up Web surfing through mobile devices, in this section, we present experimental results about a comparison between user response times with or without the usage of the MIMOSA framework.

7.1 Workload model and testbed

We set up a testbed platform consisting of four nodes of a cluster (Fig. 10). Each node is equipped with a hyper-threaded Intel Xeon at 2.4 GHz and 1 GB RAM, running Debian GNU Linux with kernel 2.6.13. We should mention that MIMOSA does not require special hardware because it may run on off-the-shelves PCs. Hardware requirements only depend on the number of concurrent users we aim to serve and on the kind of content adaptation services that have to be carried out.

To create the working set, we collected about a hundred of Web pages in a random way from Yahoo directories [54] and saved them locally on a node that, through the Apache [50] software, acts as the origin Web server. All the nodes

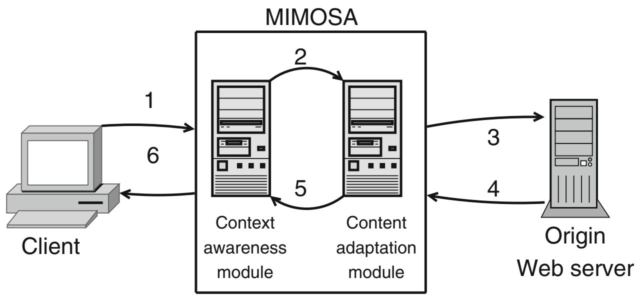


Fig. 10 The experimental testbed

are connected through a switched fast Ethernet LAN to avoid possible not predictable network effects and we consider it as the fairest way to test the MIMOSA framework for content adaptation.

Requests are referred to a mix of content types consisting of images (85%), HTML documents (8%), others (7%). HTML resources typically contain embedded objects such as images (GIFs, JPEGs and PNGs), cascading style sheets, Flash animations and Javascript codes. A Web page contains an average of 13 embedded objects, and the average dimension of the whole Web page (HTML base code plus embedded objects) is about 104 kb.

The client node runs *httperf* [36], a tool to generate various HTTP workload models. We created a trace that contains the requests for all of the HTML pages and the respective embedded objects. The trace is replayed many times changing the context information that is sent to the MIMOSA framework, so that each test implies the application of different content adaptation services, e.g. image resizing, downgrading or removal. The client connects to a node that runs, beyond the others, the context awareness module of MIMOSA (step 1 in Fig. 10). A third node runs the content adaptation module of MIMOSA, that, depending on the forwarded context information (step 2), fetches the origin resource from the Apache Web server (Steps 3–4) and then applies the right content adaptation services (Step 5). We chose to separate the content adaptation module from the others because of performance reasons. The context awareness and content adaptation nodes together constitute the MIMOSA framework. This is what the client actually perceives of the system; that is, the client does not realize the system is split over two nodes, and receives the reply from the node it contacted when issuing the request (Step 6).

7.2 Test scenarios

We emulate three test scenarios, corresponding to different network bandwidths between the client and the rest of the network: a high transfer rate (corresponding to a realistic bandwidth for the UMTS technology), a medium transfer rate (corresponding to the EDGE technology), and a low transfer rate scenario (corresponding to the European GPRS technology), respectively. Our goal is to perform experiments in realistic scenarios, and these are common ranges of bandwidth in available networks for nowadays mobile browsing. For each scenario, we test the page response times for both the origin Web server and the MIMOSA framework, i.e., the time necessary to completely download each Web page and all of its embedded objects.

In each scenario, we properly update the context information so that MIMOSA can counterbalance the increasing

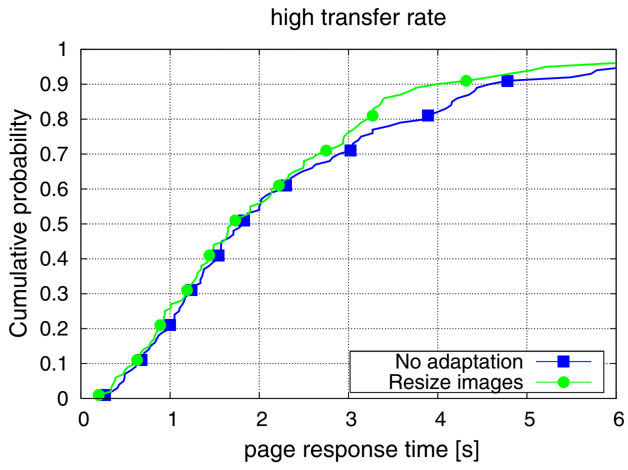


Fig. 11 Page response times for the first scenario

slowness of the link with content adaptation services that reduce more and more the size of the requested resources.

7.2.1 *High transfer rate scenario*

In the first scenario, we set up the available bandwidth between the client node and the rest of the network (called client available bandwidth in the following of this section) to 384 Kb/s. In this case, MIMOSA only resizes the dimensions of images to fit the screen of the user's mobile device. We suppose to have an available screen size of 150 × 150 pixels. As we previously said, the average size of a Web page with the respective embedded objects is about 104 kb. With the resize service, MIMOSA reduces this size to about 70 KB that is, a reduction of about 67.31%. Figure 11 shows the cumulative probability of the page response time when directly contacting the origin Web server (No adaptation) or when connecting through the MIMOSA framework (Resize images). We can see that MIMOSA obtains better response times than the origin Web server, and if we focus on the 90 percentile (Table 1), we have 4.58 s for the origin Web server vs. 3.98 s for MIMOSA, i.e., MIMOSA is more than 13% faster than the origin Web server. We would like to emphasize that MIMOSA gain over Apache for what concerns the page response time is not as good as the size reduction gain, because in the page response time there are many factors that contribute to the whole, such as the origin resource fetching time, the adaptation time, the client download time, and so on, as it will be clearly explained in Sect. 7.4.

7.2.2 *Medium transfer rate scenario*

In the second scenario, we set a client available bandwidth of 216 Kb/s. The MIMOSA framework, besides re-sizing the dimensions of images to fit the screen of the user's mobile device, also applies a downgrading of the image quality of 20%. In this way, the total size of downloaded resources is further decreased to an average value of about 56 KB. This implies a further size reduction of 13.5% with respect to the previous scenario, and this allows MIMOSA (Resize & Downgrade images) to obtain better response times than those of the origin Web server (No adaptation), as shown in (Fig. 12). We should also notice that we have another slight gain with respect to the previous scenario. In fact, focusing on the 90 percentile of the page response time (Table 1), the origin Web server has a 7.07 s response time vs. a 6.05 s time for MIMOSA. MIMOSA outperforms the origin Web server of more than 14.4%.

7.2.3 *Low transfer rate scenario*

In the last scenario, the client available bandwidth is 56 Kb/s. In this case MIMOSA, in order to counterbalance effects of this very slow connection, removes all the images from each Web page. Each image is replaced by a link and a description (taken from theALT attribute of theIMG HTML tag), so that the user is still able to download some images—if she wants to—by clicking the respective link. The average size of a page in this case is about 20 KB, which corresponds to a reduction of about 80.77% with

Table 1 Page response times (s) for the three scenarios

Server/scenario	Min	Avg	90%	Max
Apache/high transfer rate	0.27	2.52	4.58	17.81
MIMOSA/high transfer rate	0.20	2.32	3.98	12.09
Apache/medium transfer rate	0.36	4.11	7.07	21.41
MIMOSA/medium transfer rate	0.19	3.24	6.05	20.34
Apache/low transfer rate	1.40	15.81	27.19	82.25
MIMOSA/low transfer rate	0.47	3.14	6.20	23.13

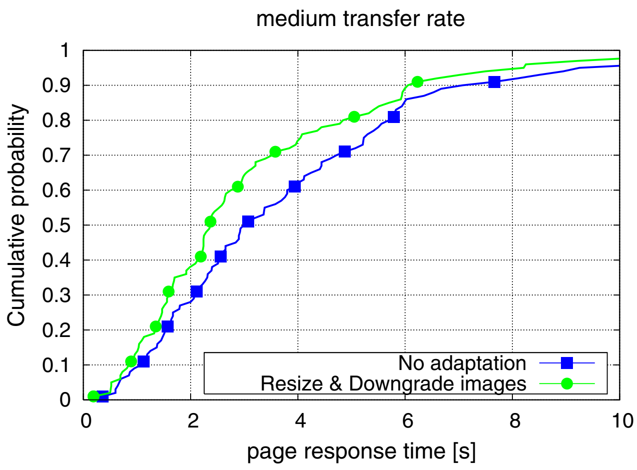


Fig. 12 Page response times for the second scenario

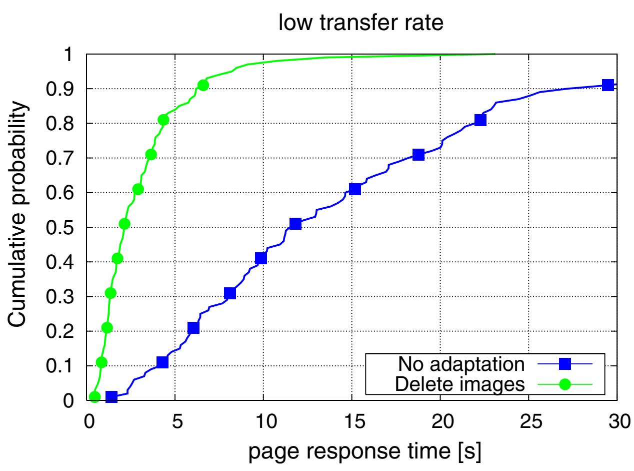


Fig. 13 Page response times for the third scenario

respect to the origin Web server. As shown in Fig. 13, MIMOSA (Delete images) obtains also in this case better response times than the origin Web server (No adaptation) and if we consider the 90 percentile of the user response time (Table 1), the origin Web server has a 27.19 s response time vs a 6.20 s time for MIMOSA. In this last scenario, MIMOSA greatly outperforms the origin Web server with a gain of more than 77%. This happens because MIMOSA realizes that the available bandwidth is very narrow and drastically reduces the traffic sent on the link; hence, it is able to limit the network load, while the origin Web server is not.

7.3 Summary of the performance evaluation

From the results presented in this section and from Table 1, we have that MIMOSA, for the considered scenarios, always has better response times than the origin Web server (both average values and 90 percentiles, which are a better measure unit in case of having to deal with quality of service and having to guarantee some Service Level Agreements). The narrower the available bandwidth is, the bigger the MIMOSA gain is. For easiness of explanation, we considered three scenarios in which the available bandwidth is fixed, but MIMOSA is able to react to environment changes in a timely way that is, it can take decisions on the fly depending on context information, while the origin Web server cannot. For instance, MIMOSA senses the available bandwidth between itself and the client devices and applies on the fly the most suited content adaptation service by taking advantage of its context knowledge. We consider these results satisfactory, especially bearing in mind that if the client connects to the Internet through a proxy server (MIMOSA in this case), the number of hops is increased of at least two (one for the request and one for the reply). We have shown that MIMOSA is able to counterbalance the delay related to these additional hops by applying the best suited content adaptation service, that brings MIMOSA response times always shorter than that of the origin Web server. We should also bear in mind that the application of transcoding services is not a zero-time process. On the other hand, it is quite time consuming, but the gain because of the much smaller resources sent over a possibly congested link overcomes this critical aspect. Clearly, if we do not have strict bandwidth limitation (e.g. we consider a LAN connection), MIMOSA effectiveness is reduced (the time spent for the application of services becomes comparable or even greater than that spent to download the resource) and MIMOSA response time may also be the same or bigger than those of the origin Web server. However, by now, mobile connections are affected by strict bandwidth limitations, so we think that MIMOSA can be a valuable aid when surfing the Web with a mobile device.

7.4 Decomposition of the response time

In this section, we try to give a better insight of the response time that is, we want to decompose it in its main parts, to better understand how they contribute to the whole and how they change depending on the scenario. To this aim, we configured Apache to log the time taken to serve the request that is, the time from when the request was received to the time the response headers are sent on the wire. (%D option in the LogFormat directive). Then we configured MIMOSA to log various timestamps for each request, in order to get the decomposition of the response time in its components such as resource fetching time, adaptation time, client download time and so on.

For what concerns Apache response time, we point out two main contributions:

$$T_{\mathrm{TOT}} = T_{\mathrm{Serv}} + T_{\mathrm{Cli\text{-}Down}}$$

where T_{TOT} represents the whole response time, while T_{Serv} and $T_{\mathrm{Cli\text{-}Down}}$ represent the time necessary to serve the request (i.e. to accept the request and generate the origin contents) and to download the resource over the Internet link that connects the client device to the origin Web server, respectively.

From our experiments (that involved static contents) we may notice two interesting aspects:

- T_{Serv} is negligible with respect to T_{Down}. In all scenarios we have a T_{Serv} in the order of magnitude of milliseconds, while the whole response time has an order of magnitude of seconds. From our experiments, T_{Serv} represents from 0.2 to 2% of the whole response time.
- T_{Serv} depends on the client available bandwidth and is not constant for a given resource, even though it varies

very slightly. This happens because it includes the time necessary to receive the request from the client and the time to generate the contents (in our case of static content, to retrieve it from the storage device). Provided that the request always consists of the same amount of bytes for a given resource, the component that changes is the time spent over the Internet link for the request to completely reach the origin Web server from the client device.

Now, we outline some characteristics of the MIMOSA response time. In this case, we point out more contributions:

$$T_{\mathrm{TOT}} = T_{\mathrm{Internal}} + T_{\mathrm{Context}} + T_{\mathrm{Fetch}} + T_{\mathrm{Adaptation}} + T_{\mathrm{Cli\text{-}Down}}$$

where T_{TOT} represents the whole response time, while the other components represent respectively:

- T_{Internal} is the time necessary to receive the request, to parse it and to allow the communication among the various modules that constitute the MIMOSA framework.
- T_{Context} is the time necessary to calculate, depending on the context information and users preferences, what are the adaptation services that have to be activated for a given resource.
- T_{Fetch} is the time spent for fetching the unadapted resource from the origin Web server.
- $T_{\mathrm{Adaptation}}$ is the time spent for adapting the resource according to the user's preferences and context data.
- $T_{\mathrm{Cli\text{-}Down}}$ stands for Client Download time and represents the time the client device spends to completely download the adapted resource from the MIMOSA framework.

Figure 14 shows the main components of the response time for the MIMOSA framework, that is T_{Internal}, T_{Context}, T_{Fetch} and $T_{\mathrm{Adaptation}}$, while $T_{\mathrm{Cli\text{-}Down}}$ is shown in Fig. 15 for both Apache and MIMOSA.

Let us draw some considerations on those times.

As you can see from Fig. 14, T_{Internal} is about 129 ms for the low transfer rate scenario, while it is about 530 ms for both the medium and high transfer rate scenarios. We recall that this time includes the file transfer among the various MIMOSA modules, thus we can explain in a very simple way why it is lower for the low transfer rate scenario, while it is almost the same for the other two scenarios. In the low transfer rate scenario, MIMOSA deletes the references to all the images from the HTML Web pages, so the average page size is greatly reduced in this case, while it is the same for the other two scenarios when also images, besides the HTML container, are downloaded, so that the average page size is increased.

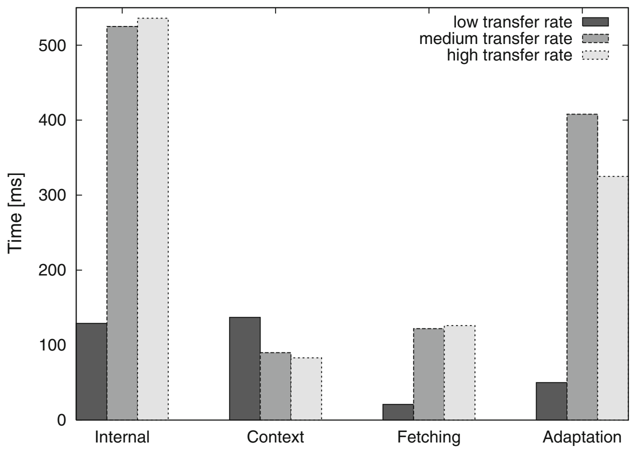


Fig. 14 Response time components (average values) for MIMOSA

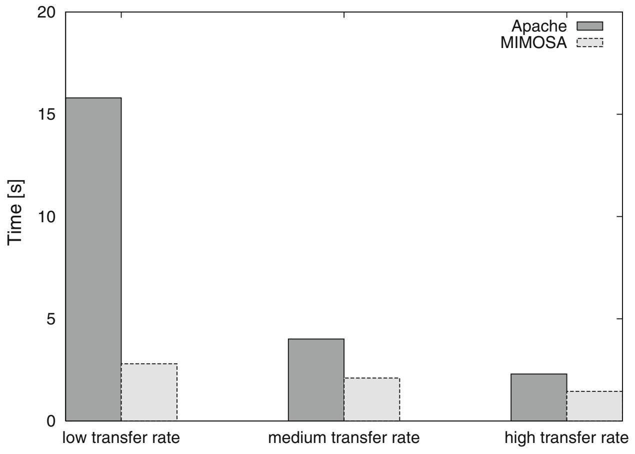


Fig. 15 Average client download times for Apache and MIMOSA

Concerning T_{Context}, we have a similar phenomenon: the time is about 137 ms for the low transfer rate scenario, while it is about 88 ms for both the medium and high transfer rate scenarios. We would expect no differences among those times, since the context information and user preferences that have to be taken into account are very similar for all the three scenarios. Actually this happens for the medium and high transfer rate scenarios, but not for the low transfer rate one. The reason is that MIMOSA takes into account those information once per page that is, when it receives a request for the HTML document, while it simply keeps the old calculated values for the embedded objects. In this way, MIMOSA has to perform calculations almost at each request for the low transfer rate scenario (when all images are removed), while in the other cases it performs the calculations only one out of 14 requests (we recall that on average a Web page has 13 embedded objects). In this way in the low transfer rate scenario the context module is more loaded and this motivates the longer times it needs to perform the calculations.

Regarding T_{Fetch}, we can draw the same considerations of T_{Internal}. The difference between the low transfer rate scenario and the other two scenarios is mainly motivated by the different average page size.

$T_{\text{Adaptation}}$ needs some more considerations to be fully explained. In the low transfer rate scenario it is very low (about 50 ms), while both in the medium and high transfer rate scenarios it is much higher (408 and 325 ms, respectively). First of all, we should recall that in the various scenarios different content adaptation services are applied. In the low transfer rate scenario, HTML Web pages are parsed to delete every reference to images, while in the other two cases image transcoding is performed. HTML parsing is much less time consuming than image transcoding and this motivates the low $T_{\text{Adaptation}}$ for the low transfer rate scenario. We should recall that in the high transfer rate scenario images are scaled to fit the client device screen size, while in the medium transfer rate scenario they are also downgraded. This motivates why the times in the medium transfer rate scenario are higher than the ones in the high transfer rate scenario, but still comparable.

Finally, we consider $T_{\text{Cli-Down}}$ that is, the Client Download time both for Apache and MIMOSA, as shown in Fig. 15. We consider it separately from the other times because it is the only component that is comparable to the respective component of the Apache response time. As we could expect, $T_{\text{Cli-Down}}$ for Apache strictly depends on the client available bandwidth. In fact, if we multiply it by this bandwidth we obtain almost the same result for all the three scenarios, while we can see that this does not happen for MIMOSA. In this case, $T_{\text{Cli-Down}}$ does not vary so greatly among the scenarios. This happens because MIMOSA realizes that the available bandwidth changes and tries to counterbalance the possible slowness of the link with a smaller size of the resources by applying different content adaptation services. In this way we obtain two important results: $T_{\text{Cli-Down}}$ is kept small also in case of very narrow bandwidth and, what perhaps is more interesting from the user point of view, the amount of bytes that are transferred between the client and MIMOSA are always less than those transferred between the origin Web server and the client. This usually means lower costs for the user, provided that many contracts between users and ISPs involve a cost that depends on the transmitted bytes.

8 Conclusions

Today, Internet clients exhibit wide differences in terms of both hardware and software properties. These different properties, together with the rapidly changing usage context of users in ubiquitous computing environments, represent a challenge mainly in wireless environments, where the dynamic reconfiguration of the services to apply is much more frequent and needful. Context-aware applications, which are able to adapt their behaviors to changing environments, can be efficiently provided on top of programmable and configurable intermediary software infrastructures. On the fly adaptation by intermediaries, in fact, is a wide applicable, cost-effective and flexible technique for addressing all of these types of variations.

In this paper, we presented a new framework that couples an intermediary architecture for Web transcoding with a middleware for context-awareness. The resulting framework provides an effective and efficient solution for the adaptation of Internet services on the basis of a comprehensive notion of context, that includes device capabilities, network characteristics, and user activity and location. The distinguishing features of this framework are the aggregation of context data provided by different heterogeneous sources, dynamic evaluation of user/service provider adaptation policies, mechanisms for conflict resolution, easy composition and programming of adaptative services, and tools for assisting the user in the declaration of her adaptation preferences. The efficiency of our solution is supported by an extensive experimental evaluation with different scenarios, and by experiments with a prototype adaptive service that takes advantage of the framework.

References

1. Agostini A, Bettini C, Riboni D (2009) Hybrid reasoning in the CARE middleware for context-awareness. Int J Web Eng Technol (in press)
2. Barnard L, Yi JS, Jacko JA, Sears A (2007) Capturing the effects of context on human performance in mobile computing systems. Pers Ubiquitous Comput 11(2):81–96
3. Barrett R, Maglio PP (1999) Intermediaries: an approach to manipulating information streams. IBM Syst J 38(4):629–641
4. Bellavista P, Corradi A, Montanari R, Stefanelli C (2003) Context-aware middleware for resource management in the wireless internet. IEEE Trans Softw Eng (TSE) 29(12):1086–1099
5. Bellavista P, Corradi A, Montanari R, Stefanelli C (2006) A mobile computing middleware for location- and context-aware internet data services. ACM Trans Internet Technol (TOIT) 6(4):356–380
6. Bettini C, Maggiorini D, Riboni D (2007) Distributed context monitoring for the adaptation of continuous services. World Wide Web J 10(4):503–528
7. Bettini C, Pareschi L, Riboni D (2008) Efficient profile aggregation and policy evaluation in a middleware for adaptive mobile applications. J Pervasive Mobile Comput 4(5):697–718
8. Bettini C, Riboni D (2007) Context-aware web services for distributed retrieval of points of interest. In: Proceedings of the second international conference on internet and web applications and services (ICIW 2007). IEEE Computer Society
9. Boyns MR (2000) Muffin: world wide web filtering system. http://muffin.doit.org/

10. Chen H, Finin T, Joshi A (2004) Semantic web in the context broker architecture. In: Proceedings of the second IEEE international conference on pervasive computing and communications (PerCom 2004). IEEE Computer Society, pp 277–286
11. Colajanni M, Grieco R, Malandrino D, Mazzoni F, Scarano V (2005) A scalable framework for the support of advanced edge services. In: Proceedings of the 2005 international conference on high performance computing and communications (HPCC-05), pp 1033–1042
12. CPAN. LWP::UserAgent-Web user agent class. http://search.cpan.org/~gaas/libwww-perl-5.805/
13. Damianou N, Dulay N, Lupu E, Sloman M (2001) The Ponder policy specification language. In: Proceedings of the international workshop on policies for distributed systems and networks (POLICY 2001), volume 1995 of lecture Notes in Computer Science, Springer, pp 18–38
14. DIWG (2006) W3C working draft: content selection for device independence (DISelect) 1.0. http://www.w3.org/TR/cselection/
15. Fritsch D, Klinec D, Volz S (2000) NEXUS positioning and data management concepts for location aware applications. In: Proceedings of the 2nd international symposium on telegeoprocessing, Nice-Sophia-Antipolis, France, pp 171–184
16. Gandon F, Sadeh NM (2003) A semantic E-wallet to reconcile privacy and context awareness. In: Proceedings of ISWC 2003, second international semantic web conference, Springer, pp 385–401
17. Google. Google Maps. http://maps.google.com/
18. Grieco R, Malandrino D, Mazzoni F, Riboni D (2006) Context-aware Provision of Advanced Internet Services. In: Proceedings of the 4th Annual IEEE international conference on pervasive computing and communications (PerCom 2006), Pisa, Italy
19. Grieco R, Malandrino D, Mazzoni F, Scarano V (2005) Mobile Web Services via programmable proxies. In: Proceedings of the IFIP TC8 working conference on mobile information systems—2005 (MOBIS), Leeds, UK, pp 139–146
20. Gu T, Wang XH, Pung HK, Zhang DQ (2004) An ontology-based context model in intelligent environments. In: Proceedings of communication networks and distributed systems modeling and simulation conference, San Diego, California, USA
21. Henricksen K, Indulska J, Rakotonirainy A (2006) Using context and preferences to implement self-adapting pervasive computing applications. Softw Pract Exper 36(11/12):1307–1330
22. Hull R, Kumar B, Lieuwen D, Patel-Schneider P, Sahuguet A, Varadarajan S, Vyas A (2004) Enabling context-aware and privacy-Conscius user data sharing. In: Proceedings of the 2004 IEEE international conference on mobile data management, IEEE Computer Society, pp 187–198
23. Iaccarino G, Malandrino D, Percio MD, Scarano V (2006) Efficient edge-services for colorblind users. In: WWW '06: Proceedings of the 15th international conference on World Wide Web, ACM Press, New York, NY, USA, pp 919–920
24. IBM (2007) IBM WebSphere Transcoding Publisher. http://www-306.ibm.com/software/pervasive/transcoding_publisher/
25. ImageMagick 6.2.5 (2005) http://www.imagemagick.org/script/index.php
26. Kamvar M, Baluja S (2007) The role of context in query input: using contextual signals to complete queries on mobile devices. In: Proceedings of the 9th Conference on human-computer Interaction with mobile devices and services (Mobile HCI 2007). ACM Publishing
27. Kao T-H, Yuan S-M (2005) Automatic adaptation of mobile applications to different user devices using modular mobile agents: research articles. Softw Pract Exper 35(14):1349–1391
28. Klyne G, Reynolds F, Woodrow C, Ohto H, Hjelm J, Butler MH, Tran L (2004) Composite capability/preference profiles (CC/PP): structure and vocabularies 1.0. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/
29. Lehmann O, Bauer M, Becker C, Nicklas D (2004) From home to world—supporting context-aware applications through World Models. In: Proceedings of the second IEEE international conference on pervasive computing and communications (PerCom 2004), IEEE Computer Society, pp 297–308
30. Li C, Willis KS (2006) Modeling context aware interaction for wayfinding using mobile devices. In: Proceedings of the 8th Conference on human-computer interaction with mobile devices and services (Mobile HCI 2006). ACM Publishing, pp 97–100
31. Locative Technologies. Geocoder. US. http://geocoder.us/
32. Microsoft Inc. NET Framework, June (2006). http://msdn2.microsoft.com/en-us/netframework/default.aspx
33. Milic-Frayling N, Hicks M, Jones R, Costello J (2007) On the design and evaluation of web augmented mobile applications. In: Proceedings of the 9th Conference on human-computer interaction with mobile devices and services (mobile HCI 2007), ACM Publishing
34. mod_perl. http://www.perl.apache.org
35. Mohan R, Smith JR, Li C-S (1999) Adapting multimedia internet content for universal access. IEEE Trans Multimedia 1(1):104–114
36. Mosberger D, Jin T (1998) httperf, A tool for measuring web server performance. Perform Evaluation Rev 26(3):31–37. http://www.hpl.hp.com/research/linux/httperf/wisp98/httperf.pdf
37. Moura J, Oliveira J, Carrapatoso E, Roque R (2002) Service provision and resource discovery in the VESPER VHE. In: IEEE international conference on communications (ICC'02), New York, USA, pp 1991–1995
38. MySql Database Management System. http://www.mysql.com/
39. OpenMobileAlliance (2001) User agent profile specification. Technical report WAP-248-UAProf20011020-a, Wireless Application Protocol Forum. http://www.openmobilealliance.org/
40. PerlMagick 6.22, (2005) http://www.imagemagick.org/script/perl-magick.php
41. Privoxy Web Proxy (2006) http://www.privoxy.org/
42. RabbIT proxy (2006) http://rabbit-proxy.sourceforge.net/
43. Ranganathan A, Campbell RH (2003) An infrastructure for context-awareness based on first order logic. Pers Ubiquitous Comput 7(6):353–364
44. Rashid O, Coulton P, Edwards R (2008) Providing location based information/advertising for existing mobile phone users. Pers Ubiquitous Comput 12(1):3–10
45. Reponen E, Huuskonen P, Mihalic K (2008) Primary and secondary context in mobile video communication. Pers Ubiquitous Comput 12(4):281–288
46. Rosaci D, Sarne GM (2006) Masha: a multi-agent system handling user and device adaptivity of web sites. User Modeling User Adapted Interact 16(5):435–462
47. Sacramento V, Endler M, Rubinsztejn HK, Lima LS, Gonçalves K, Nascimento FN, Bueno GA (2004) MoCA: a middleware for developing collaborative applications for mobile users. IEEE Distributed Syst Online 5(10)
48. Schuyler E Geo-Coder-US. http://search.cpan.org/~sderle/Geo-Coder-US/
49. Squid Web Proxy Cache. http://www.squid-cache.org/
50. The Apache Software Foundation (2007) The Apache HTTP Server. http://httpd.apache.org/
51. U.S Census Bureau. http://tiger.census.gov/cgi-bin/mapbrowse-tbl
52. US Postal Service (2000) Postal addressing standards. Technical report publication 28, November
53. Webcleaner—a filtering HTTP proxy 2006. http://webcleaner.sourceforge.net/
54. Yahoo! Inc. Yahoo! directories, 2006. http://dir.yahoo.com/
55. Zheng Y, Chan ATS, Ngai G (2006) Mcl: a mobigate coordination language for highly adaptive and reconfigurable mobile middleware: experiences with auto-adaptive and reconfigurable systems. Softw Pract Exper 36(11/12):1355–1380