Journal of Mobile Multimedia, Vol. 4, No. 1 (2008) 002–018
© Rinton Press

AN AGENTS BASED MIDDLEWARE FOR PERSONALIZATION OF MULTIMEDIA SERVICE DELIVERY IN SOAS*

ROCCO AVERSA BENIAMINO DI MARTINO

NICOLA MAZZOCCA SALVATORE VENTICINQUE

Department of Information Engineering, Second University of Naples
via Roma 29, 81031 Aversa, Italy

{*rocco.aversa, salvatore.venticinque*}*@unina2.it*
{*beniamino.dimartino, nicola.mazzocca*}*@unina.it*

Received July 1, 2007
Revised October 15, 2007

In this paper we present a proxy based middleware that is able to adapt the exploitation of services in SOAs according to the client profile. We focus on the possibility of adapting a multimedia service to client capabilities by means of a brokering activity and a platform reconfiguration.The platform is composed of several software agents acting as proxies, which intercept, process and forward incoming messages to the next agent, till the request is sent to the brokered provider. On the backward path the agents are able to adapt the content of returned responses. Proxies are dynamically created and configured. We describe the design of platform architecture and the personalization mechanisms. We show how a client application can be extended in order to exploit such platform facilities. A prototypal implementation is presented and some preliminary performance results are discussed.

Keywords: Mobile Agents, Content Adaptation, Web Services, SOA

1 introduction

SOA (Service Oriented Architecture) represents the widest accepted model to design geographical distributed systems. Web Services [1] are the standard de facto technology for developing Service Oriented Architectures. They have been widely accepted because they grant interoperability exploiting standard transport protocols (http, ftp, smtp) and text based messages. Interoperability moves the will of industries, the effort of standard organizations and the interest of research communities to make the same choices in different application fields. This technological alignment can be observed in Grid infrastructures, services providers and web farms. Even if this kind of architecture provides a certain degree of interoperability, service delivery over the Internet needs to address both the multimedia nature of the content and the capabilities of the different client devices to which the content is being delivered. The last issue is becoming such relevant as the heterogeneity of devices is increasing more and more nowadays. PDAs, smart-phone, hand-held computers, set-top boxes, embedded devices

*This work has been supported by LC3 - LABORATORIO PUBBLICO-PRIVATO DI RICERCA SUL TEMA DELLA COMUNICAZIONE DELLE CONOSCENZE CULTURALI - National Project of Ministry of Research MIUR DM17917

conceived for cars or for household appliances are some examples of intelligent equipments connected to the network. They are different for memory availability, computing power, software configuration, connectivity, communication protocols, bandwidth and so on.

Most of content adaptation systems are proxy based. Proxy components intercept the client requests, identify the client profile and adapt the delivered content. This approach appears winning since it is transparent both to the content providers and to the client developers allowing them to develop their applications independently.

The paper shows a content adaption mechanism, conceived to support service delivery to heterogeneous devices in Grid environments. A prototypal implementation of the proposed approach has been integrated in MAgDA[18], a mobile agents platform for Grid applications. The key issues that characterize our technique are:

- the description scheme of client capability;
- the specification of service requirements to be satisfied for a desired contents fruition;
- the proxy based adaptation mechanisms which perform the platform reconfiguration and the service composition;
- a straightforward extension of any client application in order to exploit the provided service adaptation.

2 State of art

Many example of frameworks to support development of context-aware applications and adaptation of content delivery are proposed in scientific literature [2, 3, 4]. How it is possible to extend these QOS management principles in Service Oriented Architecture is a relevant issue. Different architectures based on Service Level Agreement in SOAs have been proposed such as WSLA [5, 6] and WS-Agreement [7]. However QoS management is still critical for distributed SOAs because of clients with different QoS requirements and also service providers with different resources, workloads and ever-changing business rules [8]. Concepts and guidelines from ISO/IEC QoS Framework [9] form the basis for the design and implementation of QoS management in SOAs. Recently, new ideas have been proposed to extend the functionality of UDDI registry by introducing additional features such as UX [10] and UDDIe [11]. These works present a QoS model (with QoS certifiers used to verify the claims of web service's QoS). [12] provides a generalized approach for building a QoS-based registry for SOA. This framework enhances the current UDDI standard to improve the efficiency and the quality of the discovery process, by taking in account user specified QoS parameters. Others propose a new infrastructure for QoS management such as WSLA [13] or SLAng [14] where the authors address the dynamic selection of services via an agent framework coupled with a QoS ontology. [15] proposes to exploit correlations between QoS properties of each Web service to provide good candidates of Web services for the matchmaking process when service brokering is performed. [16] proposes a quality of service assessment model that is able to capture the reason behind the ratings. This allows to perform quality of service assessment in context, by using only the ratings that have similar expectations. [17] presents an integrated QoS management architecture and solution for the publish/subscribe style of enterprise SOA. Our solution implements the QoS management architecture at the middleware layer as a service

for Information Brokers. Major contributions include a general integrated QoS management architecture, a flexible and extensible XML-based QoS language, innovative resource models and exploitation of mobile agents technology.

3 MAgDA

MAgDA [18]is a Mobile Agents based framework for development and execution of parallel and distributed applications in Grid environments. MAgDA represents the middleware upon which we designed a software platform to support service delivery to heterogeneous and hand-held devices in SOAs.

The platform is able to provide the client terminal with some basic facilities such as device profiling, services discovery and authentication, and with the support to the fruition of a set of final services developed by third parties.

In our architecture the service is characterized by a service category, a 3-tuple of elements:

1. a *label* that identifies the service,
2. a *wsdl interface*,
3. an *agent* able to handle the requests for that service.

The platform architecture is shown in Figure 1. The service category is identified by a label

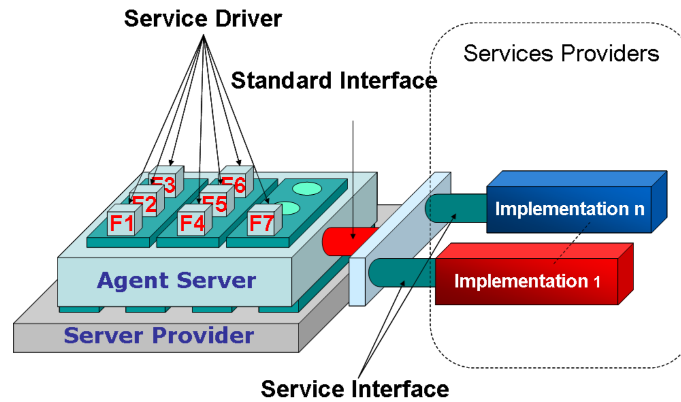

Fig. 1. The service integration

and represents all the services which can be utilized through the same interface. In order to deploy a service implementation within a chosen category the provider needs to implement at least the defined interface (the Standard Interface). So, like the Operating System is able to drive a peripheral device through a program driver, in the same way the platform is able to exploit, by means of a software agent (the Service Driver), the generic implementation of a discovered service within a specific category.

In order to handle a new kind of service the platform administrator needs to define a class (label, interface) and to implement the agent driver for that service. When the provider is publishing a new service he needs to select the class and to provide its WSDL together with some other information. A software authority performs a check to verify the compliance between the interface described in the WSDL (the Service Interface) and the one required in the selected category (the Standard Interface). If the check is successful the new service implementation is published in the UDDI register. Note that this deals with a syntactic compliance but we do not grant a semantic correctness yet.

Clients can exploit MAgDA facilities through a Web Services Interface. They can access the platform providing their credential and their profile. The authentication options offered by the Access Manager, at the state of art, are: 1)login and password, 2)challenge response based on the exploitation of a hash function, 3)challenge response with digital signature using a smart-card or not.

Web Service technology does not support a state-full interaction, but when a new client is authenticated, a new personal agent, that represents its image in the system, is created. Every further request from the same client is forwarded to the agent that handle the client profile and the intermediate results till the session has been closed.

The main facilities useful for service invocation are:

- device profiling
- service brokering
- service personalization

Figure 2 shows how this facilities are activated by the client request: the client accesses the

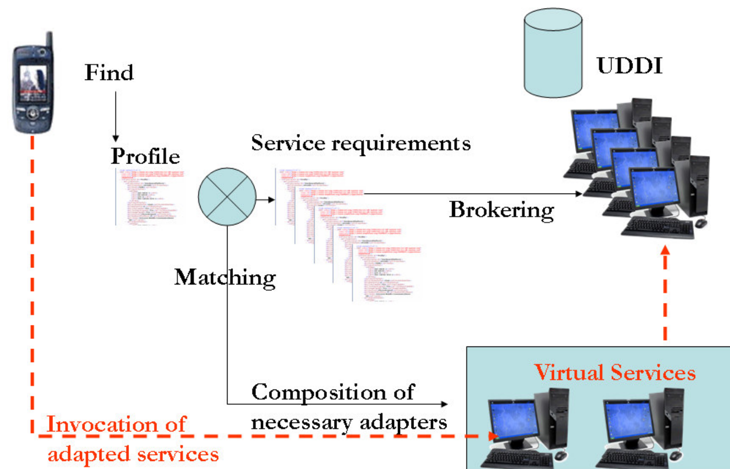

Fig. 2. Service brokering and personalization

platform and looks for a desired resource; its profile is retrieved in order to match the service

requirements with the client capabilities; the best service is brokered and if the adaptation is necessary a virtual services is built on the platform; the binding information to the virtual service is returned to the client and the service exploitation proceeds in a transparent way.

3.1 Client Profiling

A client profile is composed of personal information, device description, session properties, security level. The platform uses the selected profile to personalize the service discovery and/or to adapt the service. These information can be static or dynamic. In simplest case the client provides the link to its UAProf profile [19] that describes the capability of device in term of hardware and software resources. Other personal information can be filled manually by an user. When the device can host mobile agents, MAgDA is able to localize the user and to collect automatically dynamic information such as: amount of available memory, battery charge, software configuration, GPS position, available bandwidth, etc. This profiling phase requires an agent migrating (the agent profiler) to the device in order to collect a first coarse grain profile. According to this first information such as Operating System and CPU type, some native libraries can be downloaded in order to access the detailed information for each specific device. We developed and tested this service for several heterogeneous machines and PDAs equipped with different operative systems. Examples are IPAQ PDAs, Zaurus, Apple machines and PCs equipped with Windows or linux operative systems. The client communicates its static profile when it logs the platform. When the client looks for a specific service, the platform can retrieve other information, if it is necessary. Of course the agent profiler can migrate to the device at any time and can continue to monitor those parameters which are relevant to control the quality of service. It means that the presented architecture allows the platform to be notified about any changes of profile during service fruition. These parameters could be used to modify the service configuration dynamically. However, in our case study, presented below, profile (static and dynamic information) is retrieved just once, before of the brokering phase.

3.2 Service brokering

The client profile is employed to broker the service implementation that fits at best the device capability and the user preferences. The discovery of a service is performed by a specialized agent who is able to migrate on UDDI servers and to perform the research close the data thanks to specific algorithms. This approach optimizes the communications and the performance of the system. The results of discovery are filtered by a matching algorithm that compares the client profile with the service requirements. The selection of the best implementations of a service among the published ones is performed through a matching between two XML representations of user and service profiles. The matching algorithm is able to compute a distance between the two representations and to detect the differences. Service requirements must be described as XML subtrees of the UAPROF profile schema. When the matching detects full compliance between service requirements and client profile the binding information about the discovered implementation is returned to the requestor. The compliance between service requirements and client profile could not be granted because an optimal service implementation does not exist or it is unavailable at that moment.

3.3 Service personalization

When the compliance cannot be granted without adaptation the platform selects the service implementation that fits as well as possible the client capabilities. Then it can provide the necessary adaptation exploiting internal components or composing external services. In this case the binding information returned to the client will refer to a virtual service that is executing on the platform. Here a set of agents, acting as proxies, processes and throws client requests from an agent to another until the message reaches the brokered provider. In the same way the responses flow back to the client across a set of adapters. A tunneling of SOAP messages along different paths allows us to distribute the workload and to adapt the contents transparently. Message elaboration along the outward path makes it possible to distribute the workload among different providers and to adapt the input parameters of the brokered services. The backward path is exploited to adapt the contents returned by the provider.

4 The interaction model

A relevant issue is how the client application can use the platform facilities and exploit the delivered services. Here we propose a common approach that can be applied in general and does not depend by a specific technology. The interaction model among client, middleware and service providers has been conceived in order to design and develop each software module independently. The client application works as any other Web Service requestor. The application will invoke remote services by a set of stubs (S1, ..., Sn), which have been built by their WSDL. The extension of the client application in order to exploit the middleware facilities is straightforward. The programmer does not need to change the application code, in fact explicit interactions among requestors and platform take place just before service invocation, in order to initialize the session before the application is starting. In order to initialize the session, at middleware side, the required services are brokered and the platform is reconfigured to support the service exploitation with the new client profile. After that, at client side, the information necessary to bind the stubs to the right services is collected.

As it is shown in Figure 3 the new client will be composed of the original application and of an access manager (in Figure 3 *MS - Magda Stub*) that is used to exploit the facilities provided at middleware-side.

The MAgDA WS interface is shown in Figure 4.

In order to create a new session a client invokes the *connect* method to access the platform and to communicate its profile. It needs to communicate the user credentials and the profile for that session. For each successful authentication a personal agent is created in any container of the platform, which can be distributed geographically. A token, that identifies the personal agent for that session, is returned and must be used for all the following requests. Let us observe that Web Services technology allows to deploy stateless services. Here personal agents represent active sessions for logged users. The session can be suspended by the *logout* method, by which the personal agent is not disposed and it can be resumed at next connection. Session can be closed invoking the *closeSession* method that disposes the personal agent. In order to know those services which can be brokered and adapted by the platform the client invokes *listServiceType* method. It returns the list of service categories which are available in the platform and the WSDL address for each of them. The next step is to ask for the binding information needful to initialize the application stubs before to invoke the services. The WSDL

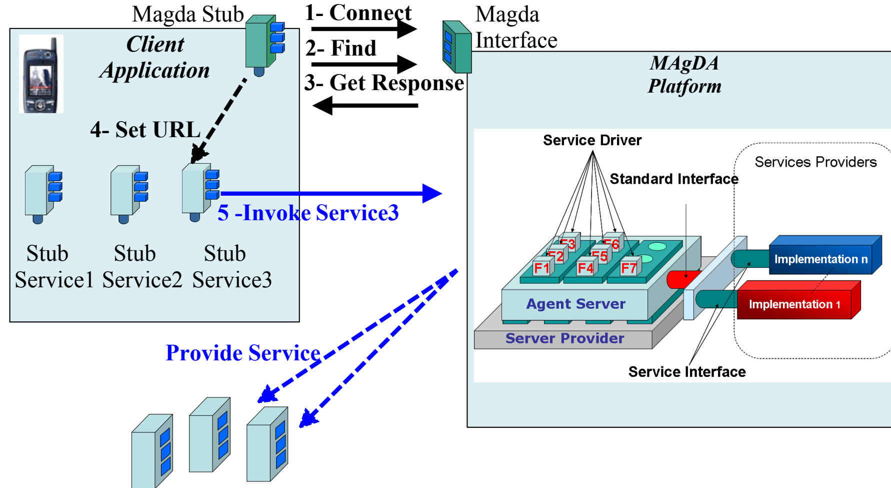

Fig. 3. The middleware interaction protocol

```
public interface MagdaWS extends Remote {
public String connect(String login, String pwd, String profile);
public void logout(String token);
public String[] listServiceType(String token);
public String find(String serviceType, String token);
public String getResponse(String requestID);
public String[] getTokenList(String token);
public void closeSession(String token);
}
```

Fig. 4. MAgDA Web Service interface

is used to build the service interface while the binding must be retrieved every time a new session is opened, because it depends by the client profile and by the dynamic condition of the system. The *find* method is used to broker the service and to collect its binding information. This request is not blocking and the received token will be used to retrieve, by the *getResponse* method, the response, when it will be available. The result will be used to initialize the client stub before to start the application. The *getTokenList* method retrieves the list of requestIDs of all pending requests belonging to the session identified by the same token. In Figure 3 the interaction protocol between the client and the middleware by the methods described above. Black lines show the interactions between the application and the MAgDA interface. By step 4 the application configure the service stubs. The 5th and the dashed lines are related to the SOAP messages which are tunneled to the services by MAgDA. The invocation of *find* method of MAgDA interface starts the brokering and the reconfiguration phase. For each available service that is able to provide the desired functionality a matching between the

service requirements and the client profile is performed. If a compliant service is found, its address is returned to the client in order to bind the stub to the right provider. Otherwise a link to a virtual service is returned, as it has been described below.

5 Proxy architecture

When a brokered service needs to be adapted a new proxy is started. The proxy performs a tunneling that allows to personalize the service exploitation. Its role is to intercept the client request and to adapt the service response to the client profile. It appears as the final service to which the client stubs are binded. Its main functionalities are the redirection of the incoming requests toward the services providers and the backward elaboration of responses. When many providers are able to satisfy the client request, the proxy can forward the SOAP messages according to load balancing criteria or according to some other business policies. Each proxy waits for the following requests:

- *start*: start of SOAP message handling;
- *stop*: discarding of incoming SOAP message ;
- *pause*: temporary suspension of SOAP messages handling;
- *restore*: restoring of SOAP message handling ;
- *move*: migration to a new node;
- *delete*: bridge termination.

Figure 5 shows the UML class diagram of the Proxy implementation.

Its configuration is done at runtime providing an XML file as input parameter. The *ProxyAgent* class parses the configuration file. It creates a *ConnectionWaiter* instance that is waiting on a specific port for each new connection. The Proxy manages two abstract classes: *Service* and *Repeater*. The Service class forwards the request to the next Agent or to the service provider. The Repeater is waiting for the response and will adapt the content if it is necessary. The configuration file specifies the concrete subclasses of Service and Repeater which will be instantiated by the Proxy and passed to the Bridge as input parameters. In order to implement a new policy to condition and redirect the SOAP requests the developer needs to extend the Service class and to provide the correct configuration for the proxy. The Service subclass will implement the *nextURLProvider* that returns the binding to the next virtual or real provider. The Repeater is delegated to process the returned response. The concrete subclass of Repeater must implement the its abstract *postProcess* method that adapts the response according to the client profile.

In Figure 6 is showed an example of configuration file.

A Proxy instance can handle more services. As it is showed in Figure 6, for each service we must specify in the configuration file the concrete Service subclass, the concrete Repeater subclass and the parameters for each of them. Furthermore a list of service bindings must be provided. The URL components (Host, Port and Path) represent the binding information of next proxy to which the SOAP request must be forwarded or the real provider. The Service subclass will implement the decision algorithm that makes the next choice among the available next processing units.

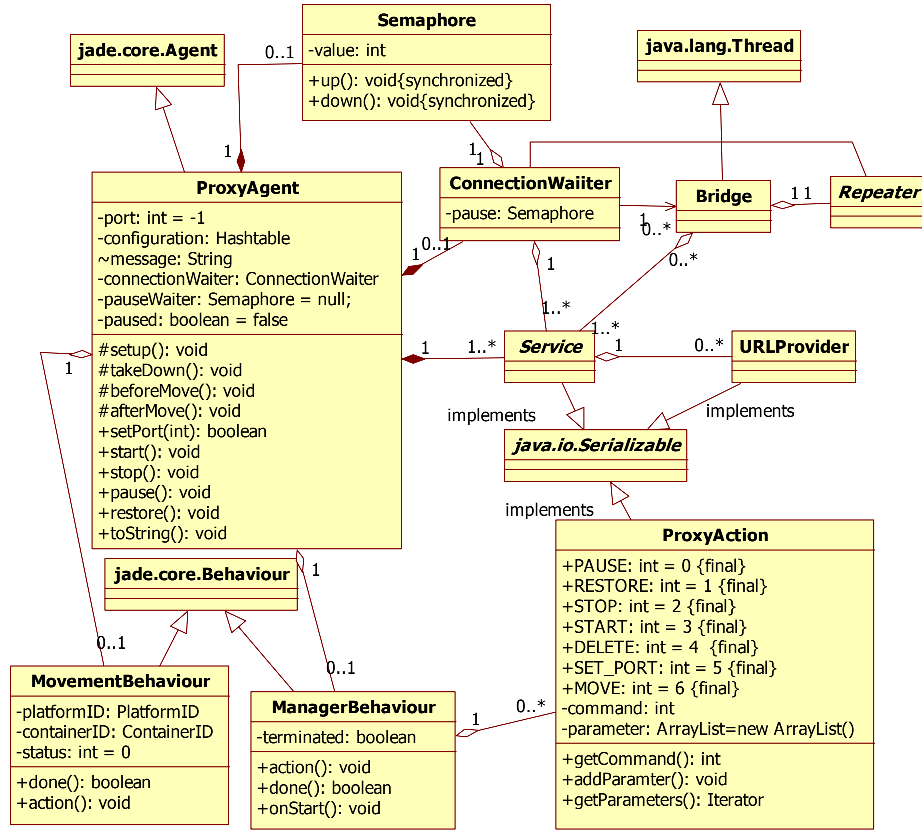

Fig. 5. UML class diagram of ProxyAgent

The input parameters of Service and Repeater subclasses can be primitive values or class themselves. In the latter case the leaves of the XML tree represent the constructor parameters. The Service subclass is mandatory, but the Repeater can be omitted. In this case the response is returned to the client as it is.

When it is created the ProxyAgent reads the file and migrates to the destination node where it will start to accept the SOAP requests. The agent will schedule two concurrent behaviors. The first one starts the ConnectionWaiter, the second one is waiting for ACL messages from the platform like START, STOP, PAUSE, MOVE and DELETE (their meaning was described above).

6 Implementation

The prototypal implementation of the described platform represents a test-bed for the integration of the experimented technologies and protocols. We deployed an implementation of the MAgDA interface by the AXIS web container and the Tomcat application Server. As it is showed in Figure 7, the MAgDA Web Service starts an agent container inside Tomcat and

```
<Services>
  <Service name="service category">
    <Class name="concrete Service subclass">
      <Parameter class="classname">
       <Parameter class="primitive type" value="this_value"/>
       <Parameter class="primitive type" value="this_value"/>
      </Parameter>
      ...
    </Class>
    <Repeater class="concret repeater subclass">
    <Parameter class="this_type" value="this_value"/>
    <Parameter class="this_type" value="this_value"/>
    </Repeater>
    <URLProvider>
      <Host>hostname1</Host>
      <Port>portnumber1</Port>
      <Path>path1</Path>
    </URLProvider>
    <URLProvider>
      <Host>hostname2</Host>
      <Port>port2</Port>
      <Path>path2</Path>
    </URLProvider>
    ...
  </Service>
...
</Services>
```

Fig. 6. Proxy configuration file

creates an agent, called *WSAgent*. The WSAgent is delegated to forward the SOAP invocation to the MAgDA platform. The WSAgent manages a queues of requests. A new pending request is pushed when a new SOAP message is received. The WSAgent is notified about it and forwards, by an ACL message, the request to other agents of the MAgDA platform that is distributed on different nodes. When the response is available a new ACL Message is sent from any MAgDA agent to the WSAgent that attachs its content to the pending request.

Each request is identified by a sessionID that is the name of the personal Agent assigned to the client, and by a requestID that identifies that request among all the pending requests for that client.

The personal Agent is able to identify the kind of any incoming messages and to activate a specific behavior for each of them. Its functionalities can be extended easily by implementing a new behavior and updating a configuration file that associates the message to the new handler.

Typical functionalities available at the state of art are:

- service publication;
- service removal;

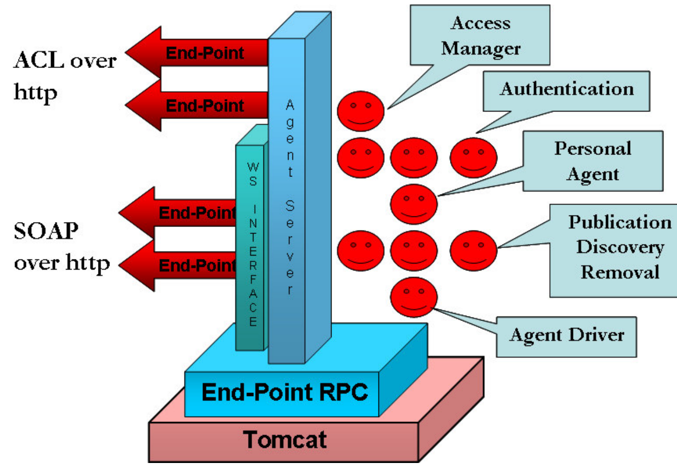

Fig. 7. The Server Provider

- service brokering;
- virtual service composition.

All of them are performed by specialized agents activated or queried when a new message has been received.

7 A case study: adaptation of map image retrieval service

In order to present a real application of the proposed approach we show, for example, a client application requiring a map of Los Angeles, or any other place. The simplest request receives as input a string that indicates the cultural site or location of interest. First of all, the application contacts the broker, asking for a valid implementation of a MapProvider service (e.g. ArcWeb `www.arcwebservices.com`). The service takes as input the address of the location (of complex type Address). The complex type Address is the sequence of address information such as city name, country name, etc. (all item belong to type String). The invocation of the getMap operation produces as result a getMapReturn element (of type String) that represents the Base64 encoding of a map associated with the target location. Different providers can return maps with different encodings , image sizes (3264x2448, 1024x768, 320x240, ...) and image weights (from few KBs to several MBs). Since heterogeneous devices can be connected to different map providers at different times, we can experience both relevant client-side failures (due to non homogeneous semantical implementations of the map provider service) and also a perceivable degradation of the system performances.

To reproduce this situation we developed three demo services: MapImage1, MapImage2 and MapImage3 . Each of them returns a geographic map with different image size (320x240, 320x240, 1024x768) and different encoding (PNG, JPG, JPG).

In the Figure 8 it is showed the SOAP message from the MapProvider.

We developed some pluggable components to configure the proxy behaviour:

```
<soapenv:Envelope>
  <soapenv:Body>
    <getMapResponse>
      <getMapReturn>
        base64_image_bytes
      </getMapReturn>
    </getMapResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Fig. 8. SOAP message from MapService

- *ServiceRR*. It extends the Service class. It performs a Round Robin scheduling among the Providers which have been specified in the configuration file.
- *ServiceRandom*. It extends the Service class. For each request it chooses, according to a random distribution, any Provider in the list.
- *ImageProcessor*. It extends the Repeater class. For each response it resizes and/or returns with a different encoding the intercepted image.

The ImageProcessor is conceived to handle images from any SOAP message. The SOAP message returned by the MapImage service is showed in Figure 8. The Repeater performs a resize and/or an trans-coding of the image identified in the SOAP message by a label provided as input parameter. It decodes the Base64 string and performs the required transformation. Then it encodes the image bytes again and return the message to the client.

We suppose to build a virtual service for a PDA with a screen that is 320x240 and that supports just PNG images. The screen resolution is a static parameter of the client device and is provided when the user logs the MAgDA platform. When it is not communicated explicitly, the agent profiler is migrates to the device and get it by invoking same native APIs. The incoming requests for the MapImage services are forwarded to a pipe composed of two proxies as it is showed in Figure 9. Of course this is not the optimal distribution of processing tasks, but this configuration is only an example that shows how the adaptations can be distributed and composed.

The first proxy distributes the requests among the three available providers, discovered at runtime. In Figure 10 only a Service instance ha been defined, while no post processors have been required.

The second proxy does not redirect the requests, but it adapts the responses according the specified parameters. Maps coming from MapImage2 are returned with a different encoding, while the ones from MapImage3 are also resized. Its configuration is showed in Figure 11.

8 Preliminary Experiments

Our preliminary experiments used Jakarta Tomcat 5.0 as application server, Apache Axis as WS platform, the J2ME Java Wireless Toolkit 2.5. Some experiments with self-reconfigurable clients have been presented in [20]. Performance evaluations aimed at estimating the overhead, due to the forwarding of requests and responses, introduced by the proxy. Furthermore we need to quantify the benefits in terms of throughput due to a distribution of requests among

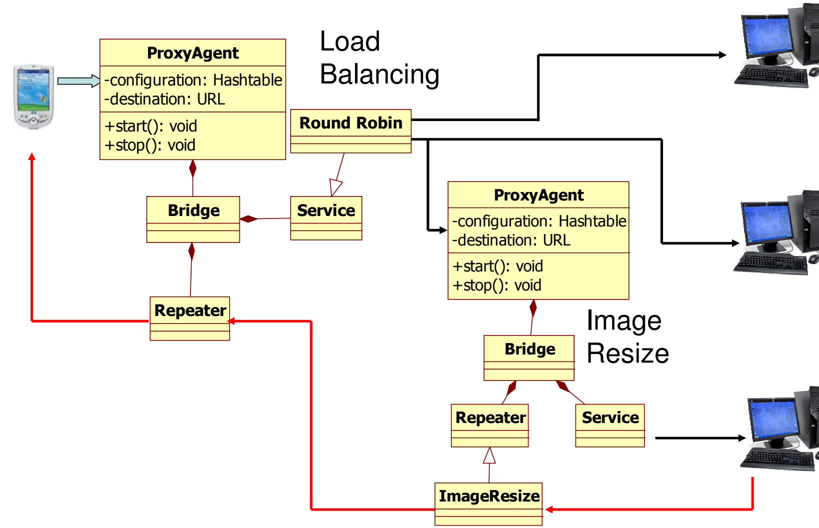

Fig. 9. The proxy configuration for the map retrieval case study

more providers. Figure 12 shows the behavior of the system when a single provider is accessed by seven clients, running with eight threads on different nodes, which invoke the service. On X axis is shown the number of sent requests for each thread. On Y axis it is shown the mean response time (ms). At the beginning of the experiment the number of contemporary messages sent by the clients do not overload the server. When all the servers are waiting for a response the response time reaches a stationary value.

In Figure 13 a proxy is used to dispatch the requests to two different providers. As it can be seen, the response time decreases. The proxy overhead it is not relevant as it processes just the HTTP message header in order to redirect the requests.

In Figure 13 a proxy is used to dispatch the requests to two different providers only when the load of the system exceed a fixed threshold. The means response time decreases and increases again till a new lower saturation level.

9 Conclusions

The paper presented an agent based software architecture for service delivery to hand-held device, that provide different facilities supporting users from service deployment to service fruition. We focused on the integration of a reconfiguration mechanism that allows service brokering and multimedia adaptation of contents downloaded by a web service interface. We exploited platform reconfiguration capabilities to build virtual services. Virtual services dispatch SOAP messages on different paths so that requests reach the optimal brokered provider and responses are adapted to the client profile. The interaction between the client and the

```
<Services>
  <Service name="MapImage">
    <Class name="services.ServiceRR"/>
    <PostProcessor class=""/>
    </PostProcessor>
    <URLProvider>
      <Host>143.225.250.126</Host>
      <Port>8080</Port>
      <Path>axis/services/MapImage1</Path>
    </URLProvider>
    <URLProvider>
      <Host>143.225.250.126</Host>
      <Port>8080</Port>
      <Path>axis/services/MapImage2</Path>
    </URLProvider>
    <URLProvider>
      <Host>143.225.250.127</Host>
      <Port>8080</Port>
      <Path>axis/services/MapImage3</Path>
    </URLProvider>
  </Service>
</Services>
```

Fig. 10. Configuration file for map service redirection

```
<Services>
  <Service name="MapImage2">
    <Class name=""/>
    <PostProcessor class="services.ImagePostProcessor">
      <Parameter class="java.lang.String" value="getMapReturn"/>
      <Parameter class="java.lang.Integer" value="320"/>
      <Parameter class="java.lang.Integer" value="240"/>
      <Parameter class="java.lang.String" value=""/>
    </PostProcessor>
  </Service>
  <Service name="MapImage3">
    <Class name=""/>
    <PostProcessor class="services.ImagePostProcessor">
      <Parameter class="java.lang.String" value="getMapReturn"/>
      <Parameter class="java.lang.Integer" value="320"/>
      <Parameter class="java.lang.Integer" value="240"/>
      <Parameter class="java.lang.String" value="image/png"/>
    </PostProcessor>
  </Service>
</Services>
```

Fig. 11. Configuration file for map resize

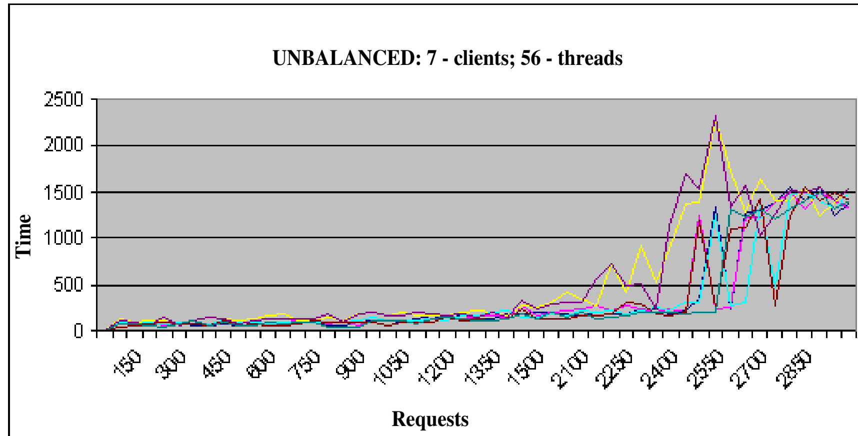

Fig. 12. Mean response time without load balancing

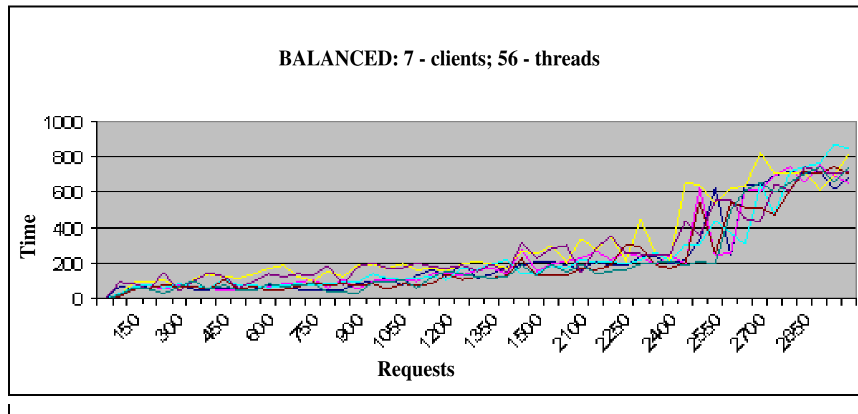

Fig. 13. Mean response time with load balancing between two providers

platform take place just in the brokering phase and does not affect the service invocation. We mean that the activities performed by the middleware are transparent during to the client fruition of the service performed by the original web service interface. A Mobile Agent based implementation has been described and a simple case study has been chosen , as proof of concept, to present real application of the presented approach. Preliminary results have been discussed.

References

1. Group, W.W.S.A.W.: Web service architecture recommendation (2004) http://www.w3.org/TR/2004/NOTE-ws-arch-20040211.
2. Steven D. Gribble, Matt Welsh, J. Robert von Behren, Eric A. Brewer, David E. Culler, Nikita Borisov, Steven E. Czerwinski, Ramakrishna Gummadi, Jon R. Hill, Anthony D. Joseph, Randy H. Katz, Zhuoqing Morley Mao, S. Ross, Ben Y. Zhao: The Ninja architecture for robust Internet-

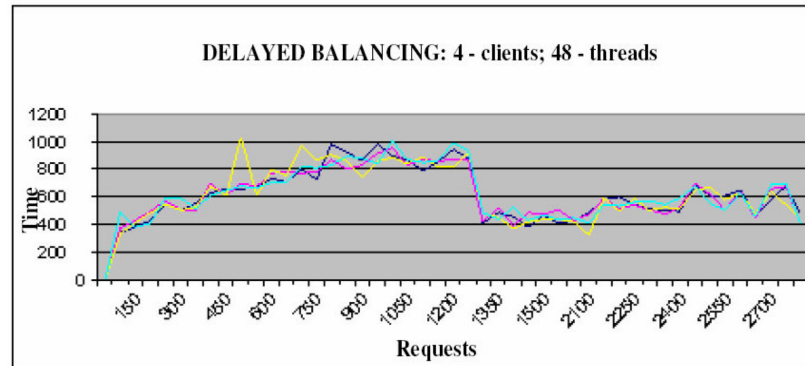

Fig. 14. Mean response time when load balancing has been delayed

scale systems and services. Computer Networks 35(4): 473-497 (2001)

3. Dey, A.K., Salber, D., Abowd, G.D. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications In Moran, T.P. and Dourish, P. (eds.) Context-Aware Computing: A Special Triple Issue of Human-Computer Interaction. Lawrence-Erlbaum, March 2002.
4. Rakesh Mohan, John R. Smith, Chung-Sheng Li: Adapting Multimedia Internet Content for Universal Access. IEEE Transactions on Multimedia 1(1): 104-114 (1999)
5. Keller, A and Ludwig, H. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. IBM research report, July 2002.
6. H. Ludwig, A. Keller, A. Dan, R. Franck, and R.P. King. Web Service Level Agreement (WSLA) Language Specification. IBM Corporation, July 2002.
7. WS-Agreement Specification site http://www.gridforum.org/Meetings/GGF11/Documents7draft-ggf-graap-agreement.pdf
8. C.Wang, G. Wang, A. Chen, and al. A policy-based approach for qos specification and enforcement in distributed service-oriented architecture. In *Proceedings of the 2005 IEEE International Conference on Services Computing*. IEEE Press, 2005.
9. I. O. for Standardization. Iso/iec. international standard 13236 technology - quality of service: Framework, Dec. 1998.
10. Z. Chen, C. Liang-Tien, B. Silverajan, and L. Bu-Sung. Ux an architecture providing qosaware and federated support for uddi. In *Proceedings of the 2003 International Conference on Web Services*, 2003.
11. A. S. Ali, O. F. Rana, R. Al-Ali, and D. W. Walker. Uddie: An extended registry for web services. In *Proceedings of the Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference*. IEEE Press, 2003.
12. A. Kumar, A. El-Geniedy, and S. Agarwal. A generalized framework for providing qos based registry in service oriented architecture. In *Proceedings of the 2005 IEEE International Conference on Services Computing*. IEEE Press, 2005.
13. A. Keller and H. Ludwig. The wsla framework: Specifying and monitoring of service level agreements for web services. Research report RC22456, IBM, 2002.
14. D. D. Lamanna, J. Skene, and W. Emmerich. Slang: A language for defining service level agreements. In *Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems*. IEEE Press, 2003.
15. L. Taher, R. Basha, and H. E. Khatib. Establishing association between qos properties in service oriented architecture. In *Proceedings of the Workshop on Service Oriented Computing: Models,*

Architectures and Applications at SAINT Conference. IEEE Press, 2003.

16. V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian. Supporting qos based selection in service oriented architecture. In *Proceedings of the International Conference on Next Generation Web Services Practices.* IEEE Press, 2006.
17. G. Wang, A.Chen, C. Wang, C. Fung, and S. Uczekaj. Integrated quality of service (qos) management in service-oriented enterprise architectures. In *Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf.* IEEE Press, 2004.
18. R. Aversa, B. Di Martino, N. Mazzocca, S. Venticinque , MAGDA: A Mobile Agent based Grid Architecture, Journal of Grid Computing, Springer Netherlands, ISSN 1570-7873 (Print) 1572-9814 (Online), pp 395-412, December 2006, Vol.4 Num. 4
19. W3C consortium, Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0, W3C Recommendation 15 January 2004 http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/
20. D. Lorenzoli, D. Tosi, S. Venticinque, R. A. Micillo, "Designing Multi-Layers Self-Adaptive Complex Applications", to be published in Fourth International Workshop on Software Quality Assurance (SOQUA 2007) Dubrovnik, Croatia, September 3-4, 2007