Adapting and Personalizing Web Content by Extending Web Proxies with Agents

Željko Obrenovic

Abstract. A proxy approach has often been used to automatically transform and adapt Web content as it naturally addresses the heterogeneity problem of clients and servers. However, existing proxy servers are still of limited flexibility and scope. In this blue note we propose a mechanism for extending proxy servers with mobile agents. The basic idea of our approach is to enable that proxy behaviour be completely or in part defined by software agents, e.g. as a multi-agent system. Various transformation and adaptation processing can be added, removed or changed by adding, removing or reconfiguring the agents. In this way, we could easily extend existing environments, enriching them with flexibility and robustness of agent systems. Transformation process can also be dynamically adapted, personalized and reconfigured, based on real-time data. We also show how proposed approach has been implemented by integrating Java-based proxy servers with Aglets - a popular Java-based agent platform.

1 Introduction

The richness in Web content and increasing heterogeneity of networks and user devices is making design of solutions for efficient and flexible online communications difficult. Many of the recent directions in Web design and multimedia, attack the problem of designing and transforming the content for multitude of devices, users, and usage scenarios [1, 2]. For example, content repurposing is taking content designed for a particular scenario, trying to automatically repurpose and adapt it to fit another scenario. However, providing real-time automated content adaptation and transformation is still not easy.

To address some of these problems, a proxies approach has often been used to automatically transform and adapt Web content, as it naturally addresses the heterogeneity problem of clients and servers [3]. However, existing proxy servers are still of limited flexibility and scope. To this end, in this blue note we propose a mechanism for extending proxy servers with transformation agents. The basic idea of our approach is enabling that proxy behaviour be completely or in part defined by software agents, e.g. as a multi-agent system. Agents would encapsulate various adaptation and transformation techniques, where various processing can be added, removed or changed by adding, removing or reconfiguring the agents. In this way, we could easily extend existing environments, enriching them with flexibility and robustness of agent systems.

In next section we briefly present some of the existing solutions. Then we portray the basic idea of our approach. After that, we show how proposed approach has been implemented using Pro-active Web Filter (PAW) and Aglets, a Java-based web filtering and agent platforms. In the end, we give conclusion and discussion.

2 Existing Solutions

There are various adaptation solutions that transform an input document into one or a number of fractional documents according to information about the client device, the network, and user preferences. According to the location where adaptation process can

take place, we can classify these adaptation approaches into server-side, intermediate (proxy), and client-side. In case of server-side adaptation, the server directly provides adapted content. Intermediate and client-side adaptations use and transform the content provided by servers, where client-side adaptation occurs in the client device, while intermediate adaptation usually take place on some other machine.

Proxies approach applies intermediate adaptations, naturally addressing the heterogeneity problem of clients and servers, where web content can be adapted based on the received data, or on the basis of author-provided metadata or adaptation hints. Web proxy systems such as Paw (pro-active webfilter) [4], and Muffin WWW filtering system [5] allow proactive filtering, with plug-in integration of custom filters and handlers. Various other adaptation proxies also exist. For example, Lakko and Hiltunen have developed the adaptation proxy to enable mobile users to access Web content that has not directly aimed for usage on mobile devices [3]. Hwang et al proposed a solution based on heuristics considering a Web page's structure and the relative importance of its components [6]. Lum and Lau provide content adaptation in an intermediary proxy server with a decision engine unit which automatically negotiates the suitable content adaptation choices [7]. Gupta et al developed the content extractor implemented within a proxy, which introduces a content-extraction framework based on the Document Object Model (DOM), where the DOM tree is navigated recursively, using customizable filters to remove and modify particular nodes [8]. Schilit et al developed the middleware proxy system to support navigation and action in independent interfaces. However, although these systems enable introduction of various complex processing and transformation methods, it is not easy to dynamically change or add these adaptation modules, nor it is possible to easily personalize their execution for individual users. Some of the systems can be used as personal proxy servers, but in this way the content has to be firstly downloaded to the user machine, and then transformed.

There are also various agent based adaptation and personalization solutions. For example, Web Browser Intelligence (WBI) [9] organizes agents on a user's workstation to observe user actions, proactively offer assistance, modify web documents, and perform new functions. WBI has been implemented as a proxy server that can run either on the user's client workstation or on a server that many users access. Albayrak et al. developed an agent-based personal information system called PIA for collecting, filtering, and integrating information at a common point, in order to support intelligent and personalized filtering of available information [10]. However, these solutions are primarily aimed to support Web browsing and filtering experience, but not transformations and adaptation of rich multimedia content that is now available at the Internet.

3 Extending Proxies with Agents

The basic idea of our approach is to extend content-filtering systems such as web proxy servers by enabling that their behaviour be completely or in part defined by software agents. Instead of developing many complex filters or handlers, we propose a generic approach where transformation and adaptation functionality of the proxy server is defined as a multi-agent system. The focus of our work is not on adaptation algorithms themselves, but on the mechanism for integration and organizing processing at the proxy server. Figure 1 illustrates this idea.

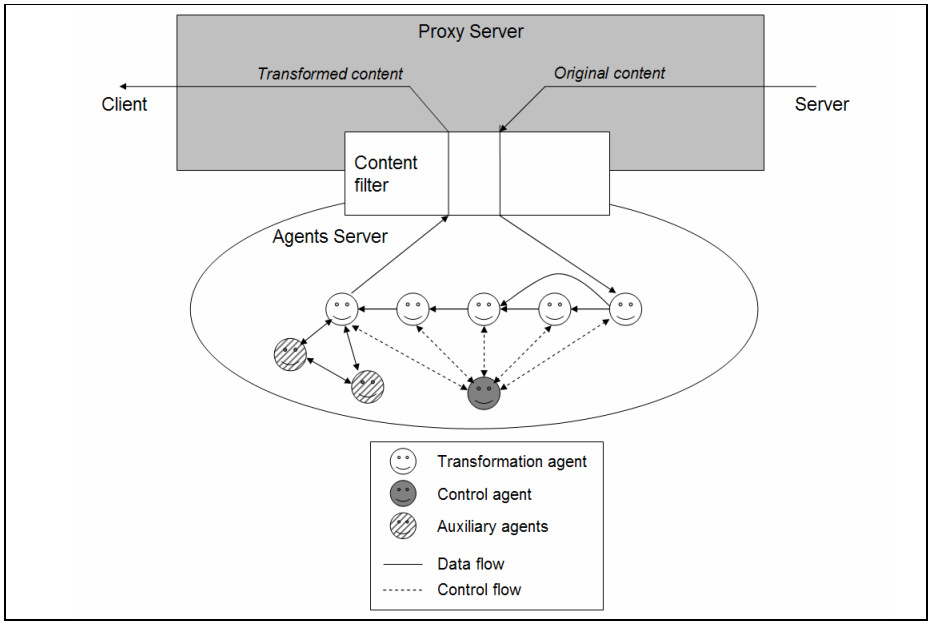


Figure 1. The basic idea of our approach.

A proxy server runs a generic agent content filter, which encapsulates the agent server, and provide the agents with the interface to the requested content. Agents encapsulate various adaptation and transformation algorithms, enabling their flexible integration in adaptation process. The server receives agents from different users, and put them in the processing pipe. When there are no agents, the original functionality of the system in unaffected. Agents can be sent by users from their machine or, for example, the administrator can install some agents for all users from his machine.

3.1 Agents

Figure 2 shows a simplified model of agent types in our system. We have defined three basic types of agents:

- *Transformation agents*, which perform content transformations, classified as:
  - *Public transformation agents*, which are common for all the users, and transform the content regardless of the user who requested the content. These agents are usually installed by the administrator.
  - *Private transformation agents*, which are private for each user, or a user group, e.g., each user or user group has its own colony of agents, and these agents do not affect content transformations of other users.
- *Control agents*, do not directly transform the content, but monitor the transformation process and the bandwidth, and change transformation parameters to make them optimal for particular users and situations, according to given criteria.
- *Auxiliary agens*, which do not directly participate in the transformation process, but are used by transformation and/or control agents. With auxiliary agents it is possible to have more complex agent organization, not just a simple pipeline.

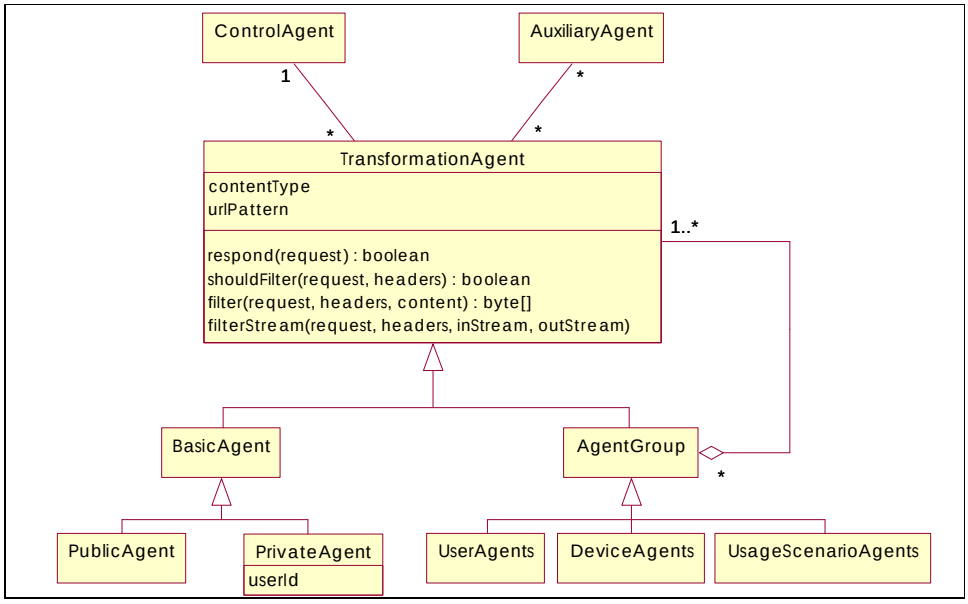


Figure 2. A simplified model of agent types in our system.

Every transformation agent is defined by *the content type* that it process, and optionally with *the list of URL patterns* for which the agent should be used. If there are no such patterns, then the agent is used for all URLs. Private agents have to additionally specify *the IDs of the users*, defined by the current Internet addresses of the users. Each agent implements an interface with several methods called by the proxy. The method `shouldFilter` is called in order to see if the agent should be used for concrete request. Here, agents can check the type of the content, address of the user, or content size. The method `respond` is called before the proxy contact the requested server. This method can be used to return the content without contacting the server, for example, from the cache. There are two types of methods that actual perform transformations. The method `filter` gets complete requested content in a buffer, and should return a buffer with transformed content. The method `filterStream` receives the references to the input and output streams, using them to read and write data. An agent should implement both the methods. Which method will be called depends on the proxy server type.

When the request for some content arrives, the handler firstly calls all the public agents registered for the type of the requested content. Then, based on the identification of the client request, private agents are called. If there are many users who use the same type of the agent, it is possible that they share the same agent, while the agent would maintain its list of user. Agents can be organized into groups, where agent could work together to achieve desired effects.

We propose organizing agents into several groups:

- *User agents*, where an individual user can create its own private agent group,
- *Device agents*, where agents can organize processing to adapt the content to device characteristics,

- *Usage scenario agents*, are dedicated to particular usage situations, such as driving a car, or visiting a museum.

When using the system, the users could define their profile, where they can specify their preferences, devices that they use, and special situations that they will be working in. Agents do not need to be send by the user; the user can just give his profile, profile of his devices and platforms, and the system could automatically create and connect the agents that could repurpose existing content so that it better suit given parameters.

4 Implementation Details

We have tested our approach extending several open-source Java proxy servers, such as Paw (pro-active webfilter) [4], and Muffin WWW filtering system [5]. These proxies allow proactive filtering, with plug-in integration of custom filters and handlers. In this section, as a proof of concept, we will describe in more details how we have extended the Paw Web filtering system by integrating it with the Aglets agent platform.

PAW (pro-active webfilter) is an open-source filtering HTTP proxy based on the Brazil Framework provided as an open-source project by Sun. Aglets is a Java mobile agent platform and library that facilitates the development of agent based applications, originally developed at the IBM Tokio Research Laboratory [11]. Aglets technology is now hosted at sourceforge.net as open source project, and is distributed under the IBM Public License (see http://aglets.sourceforge.net/).

4.1 Components

Our system consists of three types of components:

- *Agent Filter*, which is integrated in the Web proxy server, and provides agents with the interface toward the content,
- *Agent Server*, that is started and controlled by the filter, and hosts and executes the agents, and
- *Agents*, which perform or control transformation process.

In the PAW server, filters are defined as Java classes that extend predefined `Filter` class. This class defines interface that enable integration of customized code in the Web proxy. Other filtering proxy servers have similar extension mechanisms. The agent filter has several methods, including: `init()`, `request()`, `shouldFilter()`, and `filter()`. The `init()` method is called by the proxy server during initialization, and before the server starts to process requests. In our case, the main action that the filter does during initialization is starting the agent server. The `filter()`method actually performs the transformation of the content. It receives parameters of the user request, MIME headers of the request, and the content itself. The method returns a byte array with the transformed data. The filter maintains the list of agents, as well as a reference to the *control agent*. When request for content comes, the filter first determines which agents should process the content by calling agets' `shouldFilter` methods. If there is a control agent in the system, the filter gets the list of agents form it. Otherwise, it uses the list of all agents registered at the system. When it gets the list, it then calls the agents to perform processing. In each step, we

measure time of processing and compression rate, passing these data to the control agent. If there are no agents in the system, the filter bypass described logics, and returns the reference to the original content.

4.1.2 Agent Server

Aglets platform enable development of customized extensions of its agent server. The key component for integration of custom code is the `ContextAdapter` class, which has to be registered with the agent server instance. When an agent arrives at the server or when it is created, it calls the `registerAgent()` method, which registers all the agents that implement our `FilterAgent` or `ControlAgent` interface with the filter. Here is the simplified Java code showing how we have extended this adapter class to enable integration of agents with our filter class:

```
class AgentContextAdapter extends ContextAdapter {
    private AgentFilter filter;
    AgentContextAdapter(AgentFilter filter) {
       this.filter = filter;
    }
    ....
    public void agletArrived(ContextEvent ev) {
       Aglet agent = ev.getAgletProxy().getAglet();
       if (agent instanceof FilterAgent) {
           this.filter.agents.addElement( agent );
       } else if (agent instanceof ControlAgent) {
           this.filter.controlAgent = agent;
       }
    }
 }
```

4.1.3 Agents

In Aglets, every agent has to extend the `Aglet` glass. Our agents additionally have to implement our `TransformationAgent` or `ControlAgent` interface. The same agents can run with different proxy server, provided that they run the Aglet server. We have developed various transformation agents, such as:

- *Generic XSLT transformation agent*, performs XSLT transformations of content. It requires XML input.
- *TidyAgent*, based on HTML Tidy library (http://sourceforge.net/projects/jtidy), which we use to transforms HTML content into corrects XHTML format. After TidyAgent process the content, we can use agents that use XHTML scheme as input to perform further processing.
- *JPEG compression and resize agent*, provides JPEG compression and resizing based on given size, compression quality, and compression size threshold. It is based on Java Advanced Imaging API.
- *MP3 compression agent*, provide MP3 compression based on given compression quality, and compression size threshold. It is based on Java Media API.

5 Using the Agents

When proxy server starts, it runs the agents server, and can receive the agents that transform and adapt the content. Figure 3 shows two simple examples of transformations generated by sending agents on the proxy server.

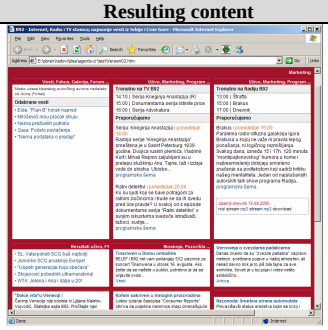
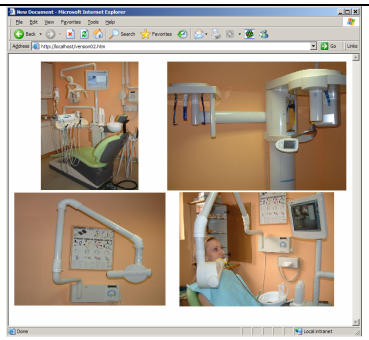
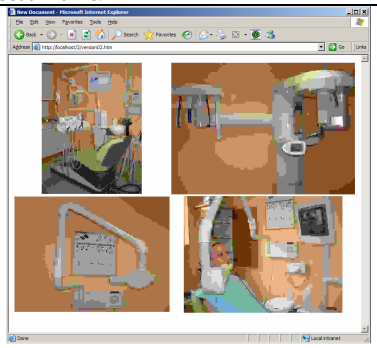
Original content	Resulting content
	
Original Web page, with total size of this page was 166545 bytes in 41 files (including image and CSS files).	Web page after sending Image and Object remover agents. Total size of this page was 41589 bytes in 3 files. Processing overhead was about 110 ms.
	
Original Web page, with total size is 167017 bytes in five files (one HTML file and four images).	Web page after sending JPEG compression agent with maximal compression settings. Total size is 26766 bytes in five files.

Figure 3. Results of usage of some agents with the Web proxy.

5.1 Creating agents

There are several ways how agents can be created and dispatched to run on the server. Agent environments usually offer standalone applications and servers, which allow users to create, configure, dispatch, retract or dispose the agent. These tools can be used directly by user or administrator to create agents, and to send them on the proxy server. The user could have its own collection of agent which he can create and send to the proxy server when want to apply some content transformation or adaptation. These

standalone applications also keep track of agents that have already been sent, allowing users to retract the agents which they do not want to use any more. This way of interaction with the proposed system can be very useful for developers, as them allows them to easily test new agents and components in real environment, without building any special tools.

Creating agents can also be integrated as a part of various applications. For example, we have developed a simple application that can help the user to create and manage its agent groups, creating agent profiles that contain a list of agents with predefined parameters. In this way, users can predefine groups of agents, and send them all automatically.

5.2 Organizing agents

In its simplest form, agents at the proxy server are organized in a pipeline, processing data in a sequence defined as they arrived at the system. Here agents use as its input the output of the previous agent, except for the first agent which process original content. However, it is also possible to organize agents in a hierarchical fashion, so that some agents provide intermediate processing. This hierarchical organization reflects model-drive translations, enabling processing of data at different levels of abstraction. For example, in perceptual content repurposing, instead of directly transforming content from one platform to another, we firstly create intermediate perceptual representation of the content, and then use transformation that aim to keep the perceptual properties of the content on the target platform [12]. Hierarchical organization of agents opens a space for various optimizations of processing. For example, we introduced an option where each agent could cache final or intermediate results of processing, and when there is the same request, it can bypass the processing, and return the results from file or memory.

5.3 Adaptive content transformations

We also enable automated creation and changing of agents configurations based on processing characteristics of the agents and bandwidth. The key to adaptive content transformations are the control agents. Currently, our control agents maintain two tables:

- *Agents performance table*, which keeps data about performance of every agents, describing average performance overhead of agents,
- *User link bandwidth table*, where control agent maintains data about bandwidth of users that have been connected. It is based on the IP address of the client request.

Control agent can also track relations among other parameters, so that it can set parameters, such as compression rate to produce optimal performance with given constraints. If you want to apply different control policy, the system can be easily changed by sending a new control agent.

6 Conclusion

We have described an approach to extending proxy servers with transformation agents, where proxy transformation process can be flexibly defined, and easily changed by adding, removing or reorganizing the agents.

Proposed way of extending proxies with transformation agents can bring several advantages. Firstly, transformation process can be flexibly defined and easily changed. The functionality of transformation filter can be changed by adding, removing or reorganizing the agents, which can be sent by the user or by the administrator from different machines. Existing filtering proxy servers require that the new filter be installed on the proxy server, usually by administrator, while the server often has to be restarted. In addition, each user can define its private set of agents, which simplifies the problem of state management and user management and configuration. Also, transformation process can be dynamically adapted and reconfigured, based on real-time data, as agents can execute asynchronously and autonomously [13]. Control agents can monitor the transformation process, and change agent parameters as needed. Processing can also be distributed through the network, as agents can move or communicate with agents and systems on other machines, supporting the system of cooperative proxy servers [14]. Proposed approach also enables easy integration of various content transformation and adaptation solutions into existing environments, using agents as glue. In addition, neither the client nor the server do not have to be aware of the agents present in the system, as their interface to the proxy server is unaffected.

This approach could also enable rapid-prototyping of various adaptation solutions, as it is possible to change proxy behavior without recompiling or restarting the system. Even if you do not want to use agent in the final version of the system, the proposed system can be used in development, as it is easy to test various processing components by adding or removing agents, or by reconfiguring them, in order to see which components are optimal for various situations.

In our future work, we will work on the developed of tools that can automatically create processing agent configuration based on profiles of user, devices and environment. To achieve this, we plan to enable that each agent can be meta-described with data about effects which it produce on the content, in terms of size, speed, perceptual effects, and/or performance requirements.

References

[1] Christian Timmerer, Hermann Hellwagner. "Interoperable Adaptive Multimedia Communication," IEEE MultiMedia, vol. 12, no. 1, pp. 74-79, January/March 2005.

[2] Gurminder Singh, "Content Repurposing", IEEE Multimedia, Vol. 11, No. 1, January-March 2004, pp. 20-21.

[3] Timo Laakko, Tapio Hiltunen. "Adapting Web Content to Mobile User Agents", IEEE Internet Computing, vol. 09, no. 2, pp. 46-53, March/April 2005.

[4] Paw (pro-active webfilter) Web page, http://paw-project.sourceforge.net/, last visited February 2, 2006.

[5] Muffin WWW filtering system Web page, http://muffin.doit.org/, last visited February 2, 2006.

[6] Y. Hwang , J. Kim and E. Seo , "Structure-Aware Web Transcoding for Mobile Devices," IEEE Internet Computing, vol. 7, no. 5, 2003,pp. 1421.

[7] W.Y. Lum and F.C.M. Lau , "A Context-Aware Decision Engine for Content Adaptation," IEEE Pervasive Computing, vol. 1, no. 3, 2002,pp. 4149.

[8] S. Gupta , et al., "DOM-based Content Extraction of HTML Documents," Proc. 12th Int'l World Wide Web Conf. (WWW 2003), ACM Press, 2003,pp. 207214.

[9] Rob Barrett Paul P. Maglio Daniel C. Kellem, "How to Personalize the Web", Proc. Of the CHI'97, pp. 75 - 82.

[10] Sahin Albayrak et al., "Agent technology for personalized information filtering: the PIA-system", Proceedings of the 2005 ACM symposium on Applied computing, Santa Fe, New Mexico, pp. 54 - 59, 2005.

[11] Hideki Tai, Kazuya Kosaka, "The Aglets Project", Comm. of the ACM, Vol. 42, No. 3, March 1999, pp. 100-101.

[12] Zeljko Obrenovic, Dusan Starcevic, Bran Selic, "A Model-Driven Approach to Content Repurposing", IEEE Multimedia, Special Issue on Content Repurposing, Vol. 11, No. 1, January-March 2004, pp. 62-71.

[13] Danny B. Lange, Mitsuru Ochima, "Seven Good Reasons for Mobile Agents", Comm. of the ACM, Vol. 42, No. 3, March 1999, pp. 88-89.

[14] C. Canali et al., “Cooperative Architectures and Algorithms for Discovery and Transcoding of Multi-Version Content,” Proc. 8th Web Content Caching and Distribution Workshop (WCW 03), Kluwer, 2003; http://2003.iwcw.org/papers/canali.pdf.