



An approach for using the Web as a Mobile Agent infrastructure

Marius Feldmann

Department of Computer Science
Dresden University of Technology
Dresden 01062, Germany
feldmann@rn.inf.tu-dresden.de

Abstract. The paper describes an approach for establishing a mobile agent architecture in the Web. This architecture makes use of currently much discussed interaction principles and the document centric constitution of the Web. It builds out an abstraction layer useful for mobile agent execution and allows realizing various feasible Web applications following the mobile agent principle.

1 Introduction

After the idea of mobile software agents had spread in the middle of the 1990s it was assumed that they would be one of the major design principles for future distributed systems. Due to their characteristics several advantages over client-server and further architectures have been pointed out [1]. This optimistic opinion changed at the beginning of the new century rapidly. There were even statements that “the only widespread incarnation of mobile software agents is malware” [2]. Beside the lack of common implementations the main reasons for the pessimistic view on mobile agents especially regarding their success in the Internet are their technological complexity, their need for special security considerations and the absence for a widespread execution infrastructure. [3]

Despite all these arguments against mobile agent systems nowadays they have assured their position in various dedicated application areas. Even though their success regarding the Internet is still missing, though there are various imaginable application scenarios realizable by the mobile agent approach in a very straight forward way. Areas of these applications include especially information retrieval and aggregation, monitoring information sources or proxy functionalities.

The approaches that have been tried to establish a mobile agent infrastructure in the Internet (e.g. Java Aglets) were mainly designed by top down approaches. After analyzing the technical needs and creating the isolated, often monolithic system it was intended to be installed on various servers in the Internet to bring the vision of mobile agents to reality. This approach failed due to the non-appearance of spreading of the created infrastructure and its too high complexity.

A more promising approach is the use of as much already existing infrastructure as possible. For this the World Wide Web offers an ideal starting basis. It provides well-known principles, protocols and languages. They can be used as the building block for a mobile agent based architecture. By this it is possible to avoid reinventing the wheel and use various available tools. Due to its document-centric nature the Web

enables a very abstract view on technical details and thus reduces complexity of the overall agent infrastructure.

2 General idea and Related Work

The general idea behind using the Web as an execution environment for mobile agents results from the powerful tool set provided by it. The Web comes with easily understandable document description formats (mainly based on XML), a global addressing scheme (URI) [6] and the application level request-reply protocol HTTP. The potentials of Web documents open a wide range of application areas. They are enrichable with logic that can be interpreted and executed on Web servers and on Web browsers. For this purpose XSLT [9] might be considered, but generally ECMAScript (JavaScript) is used. Furthermore Web documents are addressable by URIs leading to the fact, that code contained in a document is natively accessible through URI fragment identifiers. The third advantage of Web documents is their interpretation within the scope of the Document Object Model (DOM). This model has been defined by the W3C for dynamic manipulation and information extraction of tree-structured XML based Web documents.

In nutshell Web documents can be considered as

- logic and data encapsulating,
- dynamically modifiable,
- migrateable,
- reactive or even proactive

entities. With these characteristics they fulfill all requirements for mobile agents e.g. defined in [4]. Everything what is required is based on generally available standards and widespread mechanisms. By using well-known Web standards the system gains a high degree of interoperability with all types of technologies in the Web and makes an integration of Web accessible information possible. The separation of code and data is disposed. Both are components of documents and thus are part of the same domain of technology.

The idea of using documents for representing software agents is commonly known. It has already been described in various approaches. Especially XML has been used to define languages for agent description. The available approaches are often summarized under the term Active Documents. They for example occur in systems like Adlets [7] and have been extended for utilization in combination with the World Wide Web [8]. These systems define additional semantics of own code constructs included into the documents and interpreted by a specialized runtime environment. By this specialized runtime environment they follow rather the above mentioned top down approach than first checking for reusability of principals and techniques.

Apart from using documents as full-featured mobile agents they have also been utilized for distributing separated binary represented code. The most famous technology in this category is the Java Applets mechanism. Though Applets are considerably widespread, they underachieved. They have been designed as separate technology that did not evolve out of regular Web description formats, which might be a major reason for failure. Beside other mechanisms, Applets are a very good example that due to the slightly innovation resistant character of the World Wide Web new technologies have to evolve by stepwise modification of existing ones. By trying

to establish a document centric mobile agent system that is based on JavaScript, the DOM and HTTP this evolutionary condition is fulfilled. The currently often used HTTP interaction mechanisms Asynchronous JavaScript and XML (Ajax, [5]) and HTTP streaming enlarge the possibilities of such a system.

By using HTTP as communication protocol many predefined and widespread security mechanisms defined for this protocol can be used. These include especially authentication mechanisms and the extension HTTPS [11] for additional communication encryption. Based on these technologies a system model representing the mobile agent's execution environment has been developed that can be implemented in arbitrary languages. The model might be understood as an extension to the Representational State Transfer (REST) model [10].

3 Fundamental System Model

The system model is based on two components:

1. the Pneuma: the document that encapsulates data together with a execution state and logic (= the mobile agent)
2. the Soma: the execution environment that is available on Web servers and Web browsers

Document based mobile agents are intended to run both on Web servers and on Web browsers. Due to the different abilities of these two host types different requirements for the two resulting types of Somata have been formulated. For instance Web browsers are not natively addressable leading to the need for a workaround to place a mobile agent on one specific browser.

3.1 Considerations for the Pneuma

The Pneuma is based on regular Web standards, especially on JavaScript. Beside this, logic might even been formulated in XSLT. The essential fact is the wide range of possibilities that avoids fixation on one language.

A crucial extension that has to be introduced is a mechanism for distinguishing regular Web documents from those conforming to this model. To achieve this specificity a HTTP header extension has been introduced. By using the header's field-name "Agent-Host" the origin of the agent can be indicated and implicitly the distinctiveness of the HTTP request is pointed out. The extension can be seen in .

After a Pneuma has moved to the place of execution it is able to use functionality offered by the Soma. Some of the fundamental functionalities are described below. Before a document based agent migrates to one host it can check if the target host offers all necessary functionalities needed for the execution. This is done by a HTTP GET request to a predefined URI. The reply to this request contains an XML description of the host's properties including all offered abilities. It especially covers the language and method support. When a Pneuma has executed it is not able to change the state of the execution environment but only to change itself or further accessible Web resources.

After checking for necessary runtime support the document agent can migrate by using standard HTTP methods – especially the PUT method for placing a resource at

a given URI is used for that purpose. It can be used at arbitrary points of execution for relocating itself. The new location can be communicated back to the server of origin by a POST request.

```

PUT /cgi/agentinterface#321 HTTP/1.1\r\n
Host: www.someexample.ws\r\n
Connection: Keep-Alive\r\n
Accept-Encoding: gzip\r\n
Accept: text/xml,application/xml,text/html,*/*\r\n
Accept-Language: en-gb\r\n
User-Agent: Mozilla/5.0\r\n
Content-type: text/javascript\r\n
Content-length: 541\r\n
Agent-Host: www.examplehost.ws\r\n
\r\n
function aggregateDocuments(){
...
}

```

Fig. 1. Example for an HTTP request containing a mobile agent

3.2 Considerations for the Soma

The Soma is used on a host to bring the code inside the Pneuma to execution. There are two different types of these execution environments:

1. Permanent available Soma
2. Dynamical loaded temporary Soma

The permanent available Soma is located on Web servers. Beside representing the execution environment for mobile agents it manages communication with Web browser that have been bound to the server through its permanent Soma. It is accessible by a constant URI. If this URI is concatenated with the predefined fragment identifier “soma” an associated downloadable temporary Soma is addressed. For example if the Web server's execution environment is available at <http://example.com/place> a Web browser can request the temporary Soma through <http://example.com/place#soma>. After requesting the temporary Soma it is deployed on the Web browser and can realize execution of mobile agents.

A very important mechanism is the binding process between a permanent Soma and various temporary ones. When a temporary execution environment is requested by a Web browser the associated original Soma on the Web server generates a so-called temporary URI (tURI) before the request is answered. A tURI consists out of its associated URI plus a unique ID attached as fragment identifier. The unique ID can be generated by an arbitrary algorithm.

For communicating the generated tURI to the associated Web browser a further HTTP header extension has been introduced: “T-Uri”. When the request for a Soma is answered, this header-field is included to the reply message indicating the particular

tURI. All requests that are addressed to this tURI are forwarded by the Web server to the bound Web browser. Furthermore the Web server operates as proxy for all bound Web browsers. These two requirements – the addressability by tURI and the proxy functionality - enable a direct browser to browser communication and therefore the exchange of mobile agents between them. It is possible to generate or modify a Pneuma on one Web browser and transmit it for execution on a further Web browser. Out of a technical point of view the interaction between client and server is based on the non-blocking interaction mechanism Ajax or on HTTP streaming. Due to this, data that has been transmitted to the tURI by a third instance, which is requested in fixed intervals by polling or is directly forwarded to the Web browser if it is attached to the server via HTTP streaming. The overall mechanism of fetching a temporary Soma and starting communication is shown in Fig. 2.

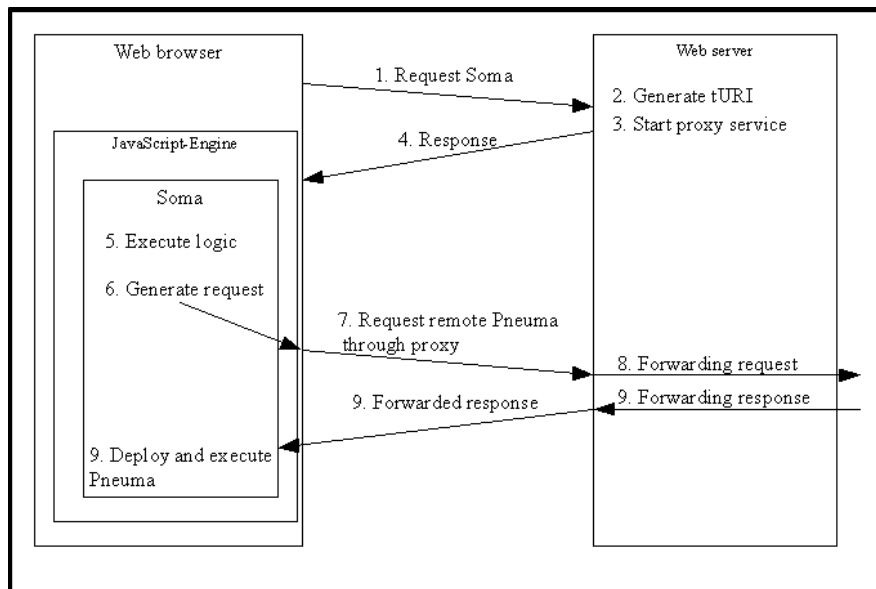


Fig. 2. Fetching a temporary Soma and request a Pneuma

When the functionalities offered by the Soma are discussed, it has to be differentiated between the permanent and the temporary one. The temporary execution environment has to be formulated in JavaScript thus making it deployable on a JavaScript-enabled Web browser. Hence the temporary Soma offers all general purpose features available in regular JavaScript programs run on Web browsers. This includes DOM manipulation and initiating HTTP based communication with remote resources.

The permanent execution environment is more complex and has to fulfill a broader range of functionalities. It does not only have the task of executing Pneumas transmitted to it, it also has to manage the temporary Somata and attached tURIs. Moreover it is responsible for realizing the proxy functionality. The Soma needs to make available a special XML description for indicating its capabilities. It is published under the URI of the execution environment plus the fragment identifier “capability”. Furthermore it has to offer the possibility for realizing HTTP interaction

with further Web servers. The permanent execution environment can be implemented in arbitrary languages and on basis of various technologies that support the realization of requirements. These include Common Gateway Interfaces, Java Server Pages or even Enterprise Java Beans.

3 Application Examples

The model has been validated and proven by various application scenarios based on its implementation. Direct browser to browser communication and exchange of data and logic between them makes a remote DOM manipulation possible which may be used in applications for remote control of Web browsers. By this a user can control the view and page presented to another user. This is suitable for support and educational purposes .

Beside direct Web browser interaction important application areas are Web proxy functionalities and document aggregation. In the case of proxy functionality a Pneuma can be located to a Soma where it functions as proxy for forwarding requests to its destination and delivering the replies to the requesting instance. By this it is feasible to first transmit such a Pneuma to a Web browser where fundamental parameters such as Quality of Service (QoS) parameters are specified. After this it migrates to an appropriate host and fulfills its work. It permanently checks the compliance of QoS parameters and if they are not accomplished the Pneuma migrates to an other host and fulfills its work from there. This dynamic proxy functionality might be useful to bypass location dependent adaptations of Web applications such as returning a set of pages of a Web search depending if the requesting instance is located in Poland or in the USA.

The last considered application scenario is the aggregation of Web documents on remote hosts. A Pneuma can be instructed by a Web browser to migrate to a Web server or another Web browser and to request information from different sources. These information are aggregated to one document and delivered to the initiating Web browser. Furthermore it is possible that the Pneuma does some filtering of content or is instructed to search for and monitor information sources by itself. This application scenario is especially useful if the Web browser is running on a host with only low bandwidth available. It is particularly feasible when requesting different Web feeds for example in Atom format. This shifts the information and feed aggregation away from the edges of the Web (the Web browsers) to its center (the Web servers).

4 Conclusion and Future Work

The work has shown that it is possible to develop a mobile agent infrastructure based on Web technology. For this minimal extensions have been defined. They all can be realized with standard Web principles and mechanisms. Particularly the expendable protocol HTTP is a powerful basis for mobile agent interaction and migration.

The model that has been developed relies on a high degree of interoperability with other Web applications and thus allows integration of them. The use of document based mobile agents allows the implementation of various feasible applications. These applications can be used by regular JavaScript enabled browser. Due to the

document centric view a high level of abstraction is achieved. Furthermore the user profits from a high degree of transparency. Especially the mentioned remote control and the proxy functionality can be realized without additional software installation and without expensive configuration steps. The extension of mobile agents' principles to the document centric World Wide Web may be one of the building blocks for realization the vision described by the buzzword 'the Web as platform'. Future work will focus more intensively on security mechanisms. Especially validating an agent's origin has to be considered in more detail. Additionally it should be possible to confine the URIs which a particular agent might access. Furthermore an XML format for indicating the capabilities of a Soma has to be specified. After completing this work an efficiency analysis for different application examples will be provided.

References

1. Lange D. B., Oshima M.: *Seven good reasons for mobile agents*, Commun. ACM, Vol. 42. ACM Press (1999) 88-89.
2. Roth V.: *Obstacles to the adoption of mobile agents*, 5th IEEE International Conference on Mobile Data Management (MDM 2004), Berkeley, CA, USA (2004) 296-297.
3. Vigna G.: *Mobile agents: Ten reasons for failure*, In Proceedings of MDM Berkeley, CA (2004) 298-299.
4. Gray R. S.: *Agent Tcl: A flexible and secure mobile-agent system*, Fourth Annual {Tcl/Tk} Workshop (1996).
5. Garrett J. J.: *Ajax: A New Approach to Web Applications*. Adaptive Path. Online article (2005)
<http://www.adaptivepath.com/publications/essays/archives/000385.php>
6. Berners-Lee T., Fielding R., Masinter L.: *Uniform Resource Identifier (URI): Generic Syntax*, RFC 3986 (2005).
7. Chang S.-K., Znati T.: *Adlet: An Active Document Abstraction for Multimedia Information Fusion*, IEEE Transactions on Knowledge and Data Engineering, Vol. 13, no. 1 (2001) 112-123.
8. Bompani L., Ciancarini P., Vitali F.: *Active Documents in XML*, ACM SigWeb Newsletter 8 (1) (1999) 27-32.
9. Novatchev D.: *Functional programming in XSLT using the FXSL library*, Extreme Markup Languages 2003 (Montréal, Québec)
10. Fielding R. T.: *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California, Irvine (2000).
11. Rescola E.: *HTTP Over TLS*, RFC 2818 (2000).