Mobile Telescript Agents and the Web

Peter Dömel[x]
General Magic, Inc., 420 North Mary Avenue, Sunnyvale, CA 94087

Abstract

Mobile agents are software programs which can move themselves around in computer networks to "do their job". Telescript is a technology specifically designed as a platform for such agents. This paper describes how Telescript agents can be created and controlled via ordinary web browsers and how they can interact with Java applets.

Keywords: *Mobile agents, software agents, interactive agents, mobile code, WWW, Telescript, Java, web services, electronic marketplace.*

1. Introduction

Today's computer networks—especially the Internet—offer an overwhelming volume of information and services that are constantly changing. Because of this, it is advantageous to have *agents*, support programs that have special knowledge about available information and services that can execute tedious or repetitive tasks on behalf of their users, or that can perform compound tasks based on existing services. Since, by there very nature, data and services are distributed, the ability of these agents to move may help to reduce network bandwidth requirements, improve efficiency, and allow users to initiate offline activities in the network. Mobile agents are especially useful to support small mobile devices (personal intelligent communicators (PICs) and personal digital assistants (PDAs)), which only have limited memory and slow network connectivity.

With these goals in mind, the development of Telescript [1, 2] started in 1991 at General Magic, Inc. At that time the World-Wide Web was practically unknown and the Internet was built and used nearly exclusively by the research community. Today, as Internet and World-Wide Web are dominating the electronic world, it is even more desirable for users to be supported by mobile agents.

Other languages besides Telescript that may also be suitable for the programming of mobile software agents include: Java [3], Tcl [4] (which now includes the Safe-Tcl concepts [5]), Oblique [6, 8], Python [9] and M0 [10]. A language independent proposal for an agent transport and interaction framework is described in [11].

[x]E-mail: peter_doemel@genmagic.com or doemel@informatik.uni-frankfurt.de

2. Telescript

Telescript is an object-oriented programming language that was designed to facilitate the development of mobile programs in computer networks. In Telescript, these mobile programs are refered to as *mobile agents*, reflecting the intention behind the invention of the language. Telescript was designed to provide an infrastructure for building electronic marketplaces, in which programs (agents) act autonomously to carry out the wishes of their users. For example, their task could consist of finding some information in the network [12], locating an electronic shop offering a specific product, or watching for a specific event, like a stock price falling below a certain limit.

In many cases, the mobility of these agents—which enables them to autonomously move between different computer systems in the network—may help to reduce network traffic and enable efficient access to the services and electronic shops of many different providers in a computer network. In contrast to more conventional remote procedure calls (RPC), this concept is called *remote programming* because agents are programs that are executed on the server on behalf of their clients. In cases where a sequence of operations must be called to produce a desired result for a user, and especially when the intermediate results require significant amounts of memory, it is often more efficient to use remote programming instead of RPC.

Of course, the remote programming paradigm raises many new questions regarding security (think, for example, of viruses). Security issues of mobile agents are addressed in [13]. To provide a secure environment for electronic commerce, the Telescript language also introduces other high level concepts like places, tickets, authorities, permits, meetings, petitions and object ownership.

Telescript enables processes to migrate by suspending themselves and resuming execution in a different computer in the network by simply executing a single instruction: *go*. *Places* are inhabited by agents. Like an agent, a place is a process, but unlike agents (which may move from place to place) places are not mobile. Places implement services which are used by agents. In order to go to a place, an agent needs a *ticket* which contains the address of the place and defines how the agent will travel (i.e., the communication protocol). Every process (agent or place) has its *authority*, that is, a unique identifier which represents a real life person or organization which is responsible and accountable for the activity of this process. A *permit* defines the kind and amount of the resources a process may consume. In order for agents to interact, there is the concept of a *meeting*. To meet another agent, a valid *petition* has to be presented by the agent initiating the meeting. Each object is *owned* by a process. Whenever an agent moves, it takes all the objects it ownes and its process state with it. Ownership may be transferred.

Some of the characteristics of the Telescript language are:

- purely object-oriented and strongly typed
- interpreted
- inherent multi-threading support
- object persistence
- process synchronization mechanisms
- fine graned resource consumption control
- automatic memory management
- mix-in classes

Mix-in classes are abstract base classes whose only purpose is to be added into a class definition. They avoid several general problems resulting from multiple inheritance, but allow programmers to benefit from merging the functionality of more than one base class.

Telescript development is supported by the Telescript Development Environment (TDE). The TDE provides:

- A project management tool
- An interactive graphical class and library browser
- A source level debugger which allows debugging of mobile processes in a distributed environment and multiple threads simultanously
- A compiler and build process control tool
- Online hypertext documentation

3. Telescript Agents and the Web

As noted above, Telescript was designed to provide an infrastructure for electronic commerce in computer networks. While Telescript was under development, the World-Wide Web (WWW)came into existence. The WWW not only provides a widely used infrastructure based on the Internet, but is rapidly becoming a de-facto standard for electronic commerce. This section describes how the WWW may benefit from Telescript's mechanisms with the help of the *Telescript active web tools*, specifically

- how Telescript agents may be created and controlled using ordinary web browsers and standard web protocols,
- and how Telescript agents themselves may access the WWW in behalf of their users.

In the current implementation of the Telescript active web tools, a Telescript engine may act as a back end to a normal HTTP server process. The services offered by the engine are currently accessed using the common gateway interface (CGI). This approach was chosen to benefit from the future enhancements of web server software. However, to improve performance, an HTTP interface to the engine itself could be provided. Telescript agents can be created and accessed by submitting special HTML forms or simply following hyperlinks. In either case, normal URLs are used to control the agent's behaviour.

The example in Fig. 1 shows the different components and how they work together.[1] We often refer to a Telescript platform as a *cloud*. Inside a cloud, most communication is accomplished using agents that go between places.

Assume that there is a Telescript place (in Fig. 1 called `UserHomePlace`), where personalized web pages for different users are stored and maintained. These pages may, for example, contain hyperlinks to access the services selected by the user from a previous session. They also may be updated by agents while the users are offline. When a user looks the next time for his or her pages, they might contain some more information or references to data or even additional pages created by agents and stored in other places. Such pages may also be used as entry points for the user to manage active agents or pending requests which are being processed by the agents of a service.

3.1. Accessing Telescript from the Web

Telescript makes it straightforward to use a standard web browser to create an interactive agent, send it to the `UserHomePlace`, and have it return an updated "home

[1]Dashed arrows show agents 'go', solid arrows the communication.

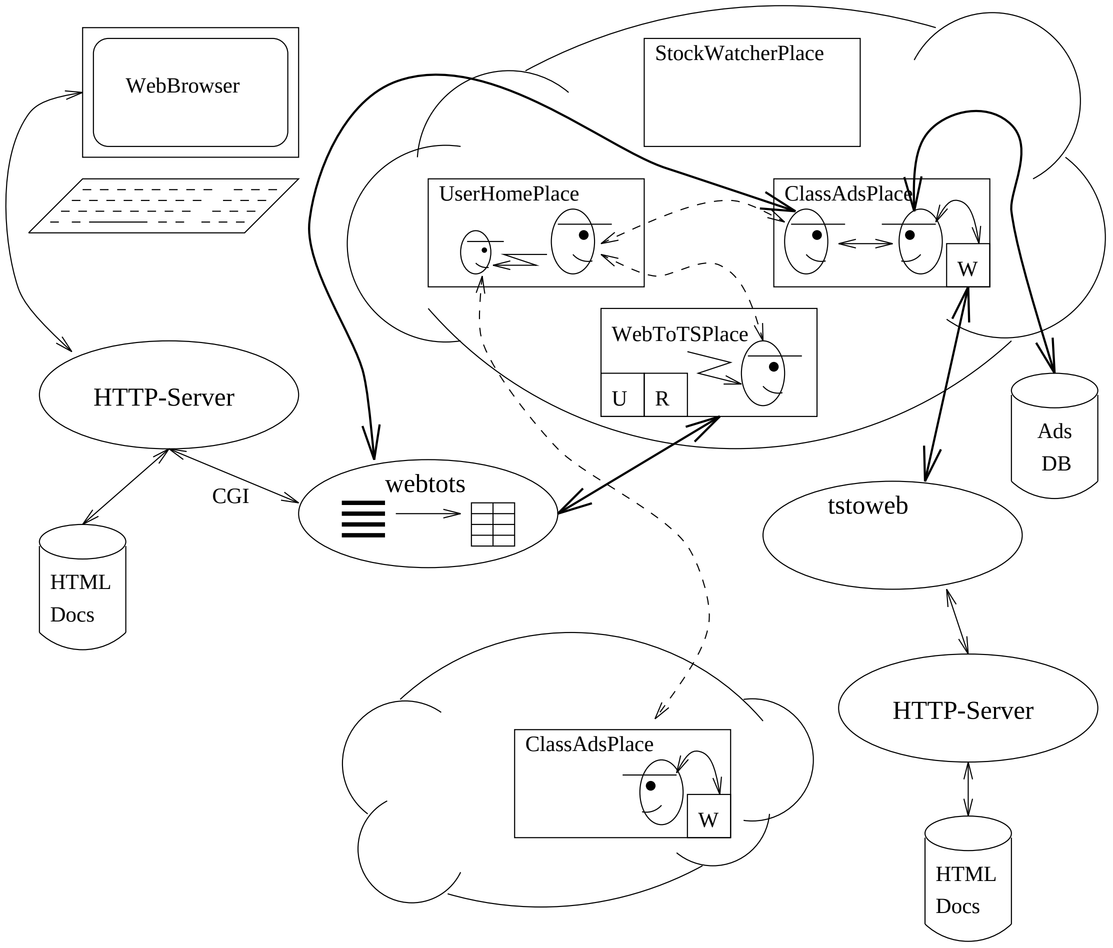

Figure 1. The Telescript WebTools structure.

page". The user simply submits a special form which contains some additional information about what the user wants to do inside the cloud (e.g., the class of the agent to be created). Submitting the form will send a special URL request to the HTTP server. Assuming, that the HTTP server runs on the host `hugo` and uses the non-standard port 3000, the URL for our example might look like this:

http://hugo:3000/cgi-bin/telescript/UserHome/getHomePage

After receiving this URL the HTTP server will create the CGI process `webtots` which will generate a Telescript procedure from its input data and send it to the `WebToTSPlace`. This Telescript procedure will be executed by the `WebToTSPlace` and usually creates a special Telescript object: a dictionary containing all the CGI environment variables and input data from HTML forms. However, it is also possible to inject new Telescript agent code which may have been stored inside a HTML document.[2]

TCP connections into the cloud or back to the `webtots` process are currently protected by generated unique random strings (so called *magic cookies*). The technique used is similar to the one provided by X window servers to protect displays from unauthorized access: A connection is only accepted if the correct magic cookie has been presented. The cookie handling itself is implemented transparently for the user and the Telescript services and agents.

In the current implementation, the `WebToTSPlace` is implemented using two special mix-in super classes:

- The Service Registry (`R` in Fig. 1)
- The User Profiles (`U` in Fig. 1)

The Registry mix-in is part of a set of classes to handle URL resolution. Our example URL will be resolved the following way. The first parts up to and including `/telescript` resulted in a connection to the `WebToTSPlace`. The part `/UserHome` will be mapped by the Registry to the class name of the place implementing the service (here it is the `UserHomePlace`). The Registry also provides Telescript tickets for agents to go to this

[2]Currently, several restrictions apply in this case, because not all security mechanisms are implemented yet.

place. The URL resolution classes automatically direct the newly created agent to the service place specified in the URL. However, before the agent starts his journey to the `UserHomePlace`, it will pick up the user's profile from the User Profile mix-in. The user profile contains some personal information about the user, provided that user had previously authenticated using the password login sequence and provided this information during an ealier registration.

The last part of our example URL will tell our agent to invoke the operation `getHomePage` which is offered by the `UserHomePlace`. After calling this operation, the agent will reconnect to the still waiting external `webtots` process and deliver this page. It is possible for services to overload this default behaviour of method invocation and provide their own mechanims to interpret remaining parts of URLs.

After delivering the user's personal page to the browser, the `webtots` process dies. But with the submission of the received HTML page, the user may—with the help of another `webtots` process—connect to his or her still living interactive agent, which is now inside the UserHomePlace. He or she may tell it, for example, to use a special distributed classified advertisment service to look for special car. To do so, the agent could create a non-interactive agent and even send it to another cloud, where it would interact with a place belonging to this distributed service. Then it might go to the `ClassAdsPlace` inside the same engine and ask there for the car (e.g., a *red BMW 525i*). The place will query its database and generate a results page containing all available *red BMW 525i* offers.

3.2. Accessing the Web in Telescript

The place might not only query a local database. It may also make use of yet another Telescript mix-in called `WebAccessing` (`W` in Fig. 1). Using this mix-in, an agent (or place) may behave exactly like a human user surfing the web. This mix-in provides a high-level Telescript API (application programmers interface) to access the web with the help of the external process `tstoweb` which uses the `libwww` HTTP client library ([14], developed at CERN) to access the web.

Knowing the exact HTML page format used by some existing non-Telescript classified web services, the `ClassAdsPlace` may parse these pages and extract, e.g., the offered *red BMW 525i*'s, thus providing a value added service by leveraging existing web services.

3.3. HTML Document Generation

To make it easy for agents and places to generate and manipulate HTML documents, a Telescript HTML API is provided. HTML pages are composed by creating a hierarchy of Telescript objects. The following Telescript code example defines a place which provides a simple HTML page.

```
HelloPlace: class (WebPlace) =
(
  public
    initialize: op(...) = {
      ^;
      *.setDoc(INDEXKEY, *.createPage());
    };

    createPage: op()  HTMLParent|Nil = {
      page := HTMLPage("Hello World");
      HTMLString(page,nil,"Hello World",
        HTML_TITLE,1,HTML_CENTER,true);
      HTMLRule(page);
      HTMLString(page,nil,
        "A Telescript place created this page.");
      return page;
    };
);
```

A user may load the page (which was generated by invocation of the method `createPage`) into his browser by simply using the following URL[3].

```
http://hugo:3000/cgi-bin/telescript/HelloPlace
```

The HTML code of the generated page itself looks like this:

```
<HTML>
<!DOCTYPE HTML PUBLIC '-//W30/DTD/ WWW HTML 2.0//EN'>
<HEAD><TITLE>Hello World</TITLE></HEAD>
<BODY ><CENTER><H1>Hello World</H1></CENTER>
<HR>This page was returned from a Telescript place.
</BODY>
</HTML>
```

At the first glance this might look somewhat complicated, but it has a high benefit from a programmers point of view: an agent may easily modify a part of a page without having to deal with HTML parsing and complicated string manipulations. Also, a page fetched from another web server using the operations of the WebAccessing mix-in can be parsed and made available as a hierarchy of Telescript objects. This way, it is easy for an agent to get a page from the web, make a little modification, and deliver it to its user (or add it to the user's personalized page in the `UserHomePlace`, where the user may fetch it later).

3.4. Watcher Agents

One area in which agents may easily assist users is to watch for the occurances of certain conditions and notify them when the event occurs, e.g., when

- a new offer is added to a classified advertisement service, which matches the requirements of a buyer's search,

[3] `hugo:3000` has to be replaced by the actual address of the web server which provides access to the Telescript service.

- the price of a certain stock exceeds a specific limit,
- a special page on the web has been modified by its author.

If a user wants to monitor something for an 'exotic' condition and the only way to do so is by polling, it is more efficient to use remote programming and send an agent directly to the server where the change is expected to happen. This agent then 'sleeps' inside this server and periodically wakes up to do its job locally, thus avoiding network traffic.

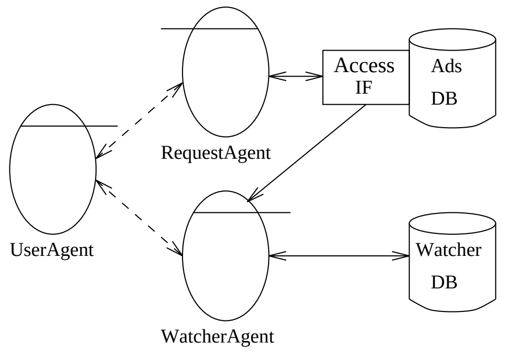

Figure 2. A watcher-agent scenario.

However, the examples given above are general enough to provide a more efficient way to do the watching. Instead of having each user send a new polling agent for each condition to be watched, a service could already provide a watching mechanism which takes the orders of many users and performs an optimized, event-triggered monitoring of data changes. Many modern databases provide triggers, and some of them (e.g., the Topic database [15]) already provide such an optimized watching mechanism.

In the case where a service doesn't use such sophisticated databases, the following simple Telescript solution may serve the same purpose (see Fig. 2). A special meeting agent (the `WatcherAgent`) takes the watching orders—which also contain the description of the means to notify the user about a change, like pager number or email address—from the user. This specialized watcher agent can pre-process and optimize all of the queries. Instead of polling for changes, it will receive and process events from the database access interface whenever some data modifications are performed (via the `RequestAgent`). It can then check immediately if the new changes match some of the watching orders and react accordingly.

4. Telescript and Java

Java is another object-oriented programming language that can support mobile code in computer networks [3]. Java is becoming accepted by the Internet community as the "applet programming language."

Like Telescript, Java provides multithreading support and automatic memory management. Compared to other interpreted languages, Java programs run very efficiently, but do not provide object persistence. Java programs (applets) may be automatically downloaded from network servers by web browsers and executed on client computers. Java is a C++-like language but does not provide Telescript's high level concepts (see section 2) for agent programming. Currently there is also no support in the language or its class libraries to allow sending, executing and controlling of programs between network servers. Java, however is well adapted to the WWW, and has library classes which allow the convenient usage of internet protocols.

At present, Java allows (subject to safety restrictions) remote programming of clients by their servers. Telescript allows clients to remotely program servers in the network. For complex tasks, either helps to minimize network load, because expensive computations can be performed locally, reducing the need for RPC.

This observation leads to the conclusion that the ideal scenario lies in the symbiotic use of both languages. Telescript agents may send Java applets to browsers on client machines. Java applets may create, control and interact with Telescript agents in network servers. Java may be used to provide a nice user interface to the normally invisible Telescript agents, and Telescript allows Java to initiate flexible offline activities in the network.

A proof-of-concept implemention demonstrating the usefulness of such a symbiosis was done by the author as part of the classified advertisement example mentioned in section 3.1. The agent that generates the main entry page to this service figures out if the user's browser is capable of executing Java applets. If it is not, the agent simply generates an HTML form. Otherwise, it returns a page containing HTML markup which causes the browser to download a user interface Java applet. The Telescript agent parameterizes the Java applet while generating the HTML markup, thus controlling its activity.

If the user's browser contains a Java interpreter (like Netscape's Navigator), the Telescript-generated HTML markup that loads the Java applet `KeynesInputDialog` in the user's browser looks like this:

```
<APPLET CODEBASE="/Keynes/classes"
    CODE="KeynesInputDialog.class" WIDTH=300 HEIGHT=50>
  <PARAM NAME=dialogTitle
         VALUE=\"Keynes User Role Input\">
  <PARAM NAME=dialogText
         VALUE=\"Please enter your role
                 [buyer/seller/groupinfo]:">
  <PARAM NAME=rectText
         VALUE="Click here to open the input dialog.">
  <PARAM NAME=autoOpenDialog VALUE="false">
  <PARAM NAME=agentURL
         VALUE="http://hugo:3000/cgi-bin/Keynes/
                        classads/customerRole">
</APPLET>
```

The Java applet, in turn, may control Telescript agents the same way a user would with a web browser: via HTTP, by simply creating and requesting special URL's. This is supported by the Java library's URL class. In the classifieds demo, the Java applet returns the user input to the Telescript agent as an HTML form would.

Two other types of interaction between Java applets and Telescript agents have been identified. These are more flexible and direct—but also require a special infrastructure (extensions to the Java and Telescript runtime environments in client browsers and Telescript engines). They are:

- Normal RPC, implemented using the available socket classes.
- Uploading of the Java applet into the Telescript engine where it could continue its execution inside a Java interpreter (which would have been linked into the Telescript engine using the engine's *external language interface* (ELI)).

For the first suggestion, some additional security has to be provided, e.g., the magic cookie solution (mentioned in section 3.1) in connection with SSL [16] protected communication. The second solution is technically 'expensive' and would require the ability of Java programs to migrate themselves.

5. Conclusion and Acknowledgements

In the research area of mobile agents, exciting things are currently going on. The symbiosis of Telescript and Java provides the foundation for a new breed of communicating applications. In order to increase the agent's capabilities, many approaches could be taken to make them more 'intelligent'. Also, a better infrastructure for routing these agents to the right destinations is needed, as well as a framework to let 'foreign' agents and services interact. Mobile code can also be helpful for the dynamic extension of applications during runtime (see [17]).

Telescript was built with contributions of many 'Magicians,' but primarily due to Jim White, Doug Steedman and Chris Helgeson. The *Telescript Active WebTools* are the fruit of the work of Chris Bloom, Adam Hertz and the rest of our Internet Group. Finally, I would like to thank Kate Greer and Joe Tardo for their help in preparing this paper.

References

[1] **White, James E. [1996]:** *Mobile Agents*; in *Software Agents* by Jeffery Bradshaw, AAAI Press / MIT Press; http://www.genmagic.com/Telescript/TDE/AGENTS.PDF

[2] **White, James E. [1994]:** *Mobile agents make a network an open platform for third-party developers*; Computer (IEEE Computer Society), 27(11): 89–90

[3] **Gosling, James; McGilton, Henry [1995]:** *The Java Language Environment: A White Paper*; http://java.sun.com/whitePaper/java-whitepaper-1.html

[4] **Ousterhout, John [1994]:** *Tcl and the Tk Toolkit*; Addison-Wesley Publishing Company (ISBN 0-201-63337-X); ftp://ftp.aud.alcatel.com/tcl/ftp.smli.com

[5] **Borenstein, Nathaniel S.; Rose, Marshall T. [1994]:** *EMail With A Mind of Its Own: The Safe-Tcl Language for Enabled Mail*; ULPAA'94, Barcelona; ftp://ftp.fv.com/pub/code/other/

[6] **Cardelli, Luca [1995]:** *A Language with Distributed Scope*; Proc. of the 22nd Annual ACM Symposium on Principles of Programming Languages: 286–297; http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-122.html

[7] **Bharat, Krishna; Cardelli, Luca [1995]:** *Distributed Applications in a Hypermedia Setting*; http://www.cc.gatech.edu/gvu/people/Phd/Krishna/IWHD.html

[8] **Bharat, Krishna; Cardelli, Luca [1995]:** *Migratory Applications*; UIST'95; http://www.cc.gatech.edu/gvu/people/Phd/Krishna/Migration.ps.Z

[9] **van Rossum, Guido [1995]:** *The Python Language Home Page*; http://www.python.org/

[10] **Tschudin, Christian F. [1995]:** *Protokollimplementierung mit Kommunikationsboten*; in *Kommunikation in Verteilten Systemen*: 475–489, Springer Verlag (ISBN 3-540-58960-0), Germany; ftp://cui.unige.ch/pub/m0/

[11] **Lingnau, Anselm; Drobnik, Oswald; Dömel, Peter [1995]:** *An HTTP-based Infrastructure for Mobile Agents*; WWW Fall'95, Boston; http://www.tm.informatik.uni-frankfurt.de/ma/www4-paper.html

[12] **Daigle, Leslie et al [1995]:** *URAs - Uniform Resource Agents*; CIKM'95 Intelligent Information Agents Workshop; http://services.bunyip.com:8000/products/silk/silktree/uraintro.html

[13] **Tardo, Joe; Valente, Luis [1996]:** *Mobile Agent Security and Telescript*; IEEE Conference COMPCON'96; see also http://www.genmagic.com/Telescript/TDE/security.html

[14] **Frystyk Nielsen, Henrik; Berners-Lee, Tim; Lie, Hakon W.; Groff, Jean-Francois [1995]:** *The W3C Reference Library*; http://www.w3.org/pub/WWW/Library/

[15] *Topic*; Verity, Inc.; http://www.verity.com/products/family.html

[16] **Hickman, K., and El Gamal, T. [1995]:** *The SSL Protocol*; Netscape Corporation, Mountain View, CA; http://home.netscape.com/newsref/std/SSL.html

[17] **Dömel, Peter [1995]:** *WebMap - A Graphical Hypertext Navigation Tool*; WWW Fall'94; Computer Networks and ISDN Systems, Vol. 28: 85–97, Elsevier Science B.V., Netherlands; http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/doemel/www-fall94.html