Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

The Changeling Web Server

Ashley McClenaghan

Abstract

This document is an introduction and user guide to Changeling.

Changeling is a prototype, script-based Web server which supports an extensible set of methods.Methods are implemented as Safe-Tcl scripts. Changeling clients may install new methods, or inspect, invoke or remove existing methods via the HTTP protocol while the Changeling server is running.

Changeling has been developed as part of the ANSA B3 Workpackage on “Scripts and Agents”.

APM.1453.01 **Approved** 21st September 1995

Briefing Note

Distribution:
Supersedes:
Superseded by:

The Changeling Web Server

The Changeling Web Server

Ashley McClenaghan

APM.1453.01

21st September 1995

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

TELEPHONE UK	(01223) 515010
INTERNATIONAL	+44 1223 515010
FAX	+44 1223 359779
E-MAIL	apm@ansa.co.uk

Contents

1 Introduction

1.1 Objective

- Objective [McClengahan]: To prototype a script-based Web server which supports an extensible set of methods.

The aim of the ANSA B3 Workpackage “Scripts and Agents” [ANSA] is to promote the evolution of open mobile scripting technology for programming the Internet. This is the context for the work reported in this document.

1.2 A Web server which supports an extensible set of methods

The HyperText Transfer Protocol (HTTP/1.0) [HTTP] defines how Web clients and servers communicate. HTTP uses the RPC-like request-response model where:

1. a Web client sends a request to a Web server
2. the request invokes a method on the Web server
3. the result of the method is sent back to the client as a response.

HTTP is designed to support an extensible set of methods. At present most Web servers recognise only a basic set of methods (eg. `GET`, `HEAD`, `POST`, etc.) and provide no support for extending this method set. However, our new Changeling Web Server supports the extensibility feature by using Tcl scripting technology [Tcl/Tka][Tcl/Tkb][MIME/Safe-Tcl][Safe-Tcl][Email/Safe-Tcla][Email/Safe-Tclb]

1.3 Basic idea

The basic idea is that when the Changeling server is initialized it supports only a core set of ‘kernel’ methods (`INSTALL`, `REMOVE`, `INFO`, etc.). These kernel methods are used to bootstrap and manage new methods including a set of ‘standard library’ methods (`GET`, `HEAD`, `POST`, etc.) that are loaded automatically. Users may `INSTALL` their own methods which will persist across server process reincarnations until removed by a `REMOVE` request or a re-configuration of the server.

1.4 Why

The HTTP/1.0 definition allows for a server with an extensible set of methods but, until now, no-one has implemented such a server (to the best of my knowledge). Changeling implements such a server.

With respect to the ANSA Workprogramme, Changeling was developed as part of an investigation into mobile scripting technology. Specifically, the Changeling work has contributed to our understanding of (Safe-)Tcl, safe

interpretation, in-service (on-line) software modification, and the use of the Web as a delivery and execution context for scripts. This work relates to ANSA's Web-CORBA [Edwards][ReesEdwards] work and Web Meta-Data work [Madsen95].

2 Architecture

2.1 The main components

Figure §2.1 shows the structure of the Changeling server. The functions of the main components of the server are outlined below.

2.1.1 The connection handler

The main loop of the server is a Tcl script (with Tcl-DP, TclX[1] and Safe-Tcl extensions) which listens for new HTTP connections at the server port.

A new Tcl interpreter is spawned to handle each connection[2].

2.1.2 Parsing the request

The spawned interpreter handling an HTTP connection will parse the header information in the incoming request. The results of the parse will be placed into a data structure which will later be made available to the requested, executing method.

2.1.3 The restricted interpreter

The (unrestricted) Tcl interpreter creates a (restricted) Safe-Tcl interpreter to execute the requested method script. The restricted interpreter executes method scripts inside a containment environment which prevents the method scripts from directly accessing any outside resources. The requested method script is loaded into the restricted interpreter and executed (interpreted).

2.1.4 Access to local resources

A method script (executing within the restricted interpreter) may indirectly access the local resources using a set of well controlled resource access procedures (see §4.1) made available through the unrestricted interpreter. These procedures provide facilities for:

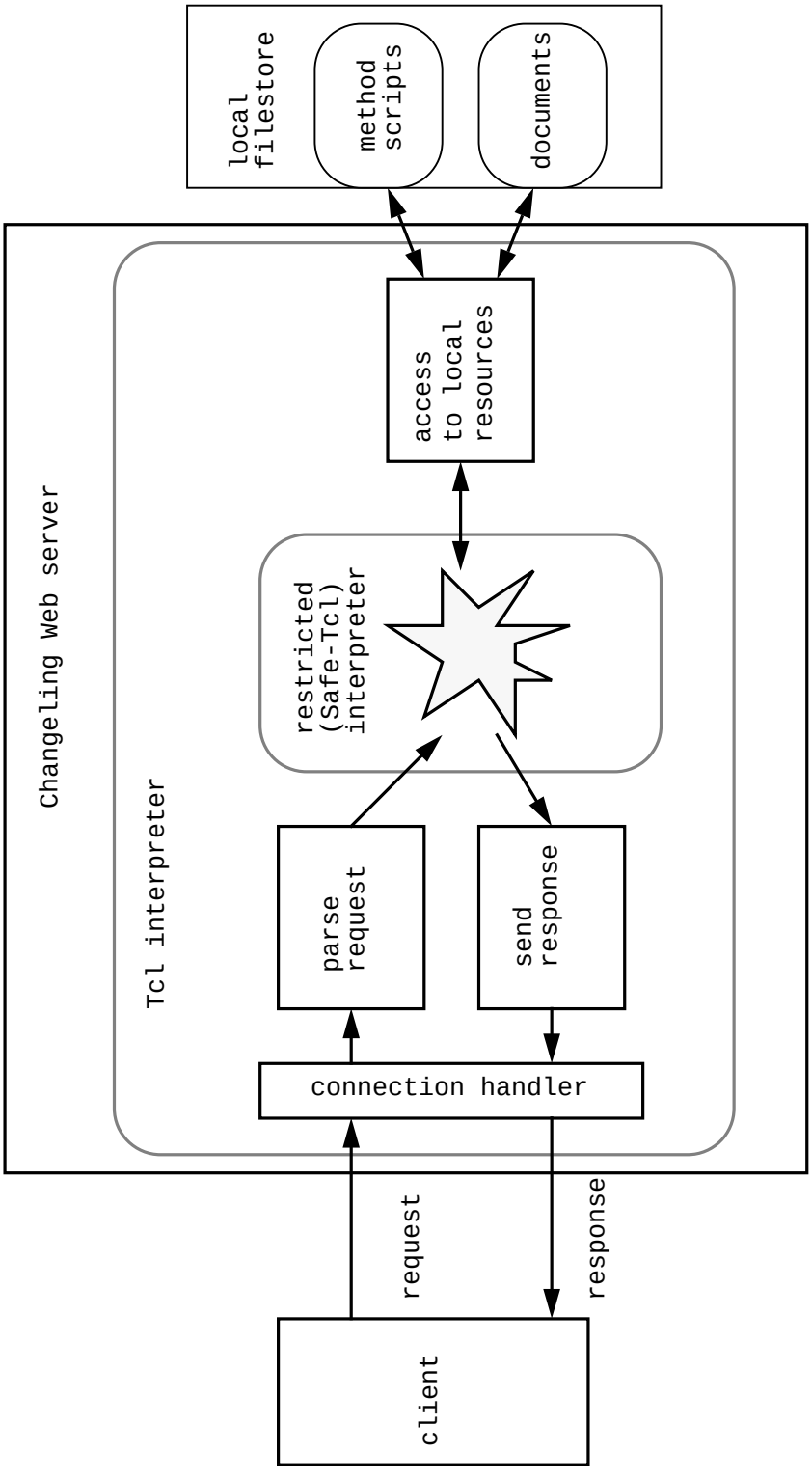
- accessing the information obtained from parsing the HTTP request headers
- reading and writing local resources
- installing and removing methods.

---

1. I would like to acknowledge the work done by Lindsay Marshall. I have learnt lots from his TclX-based Web server called Jungle [Marshall].

2. Which means that in this initial design stateful client-server interactions will have to be implemented using the local filestore to store state information between HTTP connections. Later designs might spawn and maintain a Tcl interpreter for each stateful interaction.

Figure 2.1: Architecture of the Changeling Web Server



2.1.5 Authorization

The server supports two classes of users: authorized/registered and unauthorized/unregistered.

- Both classes of users can read anything in the local resource space[1].

- Unauthorized users can write to only the temporary scratchpad area in the resource space whose contents are automatically deleted by a Changeling `cron` job after some expiry period.
- Authorized users can write to both the temporary scratchpad area and their own private area in the resource space. Authorized users can also install and remove their own methods.

To provide accountability, only authorized/registered users are allowed to write documents into the resource space or to install methods.

Any user may become authorized/registered by invoking the `REGISTER` method. At present, authentication is based on simply maintaining username/IP address pairs. A request is authorized if the username supplied in the request headers and the IP connection address matches a username/IP address pair in the database of registered users.

The present authentication scheme is rather weak. Possible security enhancements might use PGP public-key security [Zimmermann] or incorporate the Secure-HTTP scheme.

1. The local resource space is mapped to the local file space. Changeling users can access a resource if Changeling allows them access to that resource and if the file to which that resource is mapped has the appropriate file permissions.

3 Methods

This chapter introduces the 'kernel' and 'standard library' methods supported by Changeling. (We assume that the reader is familiar with the HTTP protocol and the (Safe-)Tcl language.)

3.1 Kernel methods

These are ever-present methods for managing the registration of users and the installation of new methods.

3.1.1 REGISTER

A client can invoke the `REGISTER` method to ask the Changeling server to register it as an authorized user. Example invocation:

```
REGISTER / HTTP/1.0
Authorization: ChangelingIP am
```

Where `am` in the `Authorization` header is username of the user to be registered. When Changeling registers a user it pairs the given username with the IP address of the user's client. Changeling will recognise a request from a registered user as an 'authorized request' if the username in the `Authorization` header and the IP address of the connected client match a registered username/IP address pair.

3.1.2 UNREGISTER

A client can invoke the `UNREGISTER` method to ask the Changeling server to unregister it as an authorized user. Example invocation:

```
UNREGISTER / HTTP/1.0
Authorization: ChangelingIP am
```

This request must be an 'authorized request'. When a user is unregistered any methods that they have installed are removed, and any documents that they have posted are deleted.

3.1.3 INFO

A client can invoke the `INFO` method to ask the Changeling server to return details about the environment in which a method script will be interpreted. Example invocation:

```
INFO / HTTP/1.0
Content-Length: 8

methods
```

Where methods in the Object-Body of the request is an actual parameter to the INFO method. The INFO method recognizes the follows actual parameters:

- userAdmin — returns user registration information as a list of username/IP address pairs.
- methodAdmin — returns administration information about all the presently installed methods. The information is formatted as a list of method name/method path/author username/author IP address tuples.
- methods — returns a list of method names of all the installed methods.
- procs — returns a list of procedures available to a method script.
- globals— returns a list of the global variables available to a method script.
- commands — returns a list of the commands available to a method script.

3.1.4 INSTALL

A client can invoke the INSTALL method to ask the Changeling server to record a script as a callable method. Example invocation:

```
INSTALL /methods/am/DOUBLE.tcl HTTP/1.0
Authorization: ChangelingIP am
Content-Length: 164


proc DOUBLE {} {
   set number [string trim [chgl_lookup Object-Body]]
   set result [expr 2 * $number]
   return [STDMSG_essence OK 200 “Ok” “Result = $result”]
}
```

Where the URI /methods/am/DOUBLE.tcl specifies where, in the resource space, the method is to be installed. This path must begin with methods, followed by the username (am in this example) of the registered user installing the method (am in this example), followed by the name of the method (DOUBLE in this example) with a .tcl extension.

In the current version of Changeling all method scripts are installed in the same space. Therefore the name of a new method must be different from the name of any existing method. Also, a given method script may use procedures in other already installed method scripts and, in-kind, the procedures in the given script will be made available for other method scripts to use.

The script which implements the method to be installed, DOUBLE in this example, is supplied as the Object-Body.

3.1.5 REMOVE

A client can invoke the REMOVE method to ask the Changeling server to remove a method that it has previously INSTALLed. Example invocation:

```
REMOVE /methods/am/DOUBLE.tcl HTTP/1.0
Authorization: ChangelingIP am
```

Where the URI /methods/am/DOUBLE.tcl specifies the method to be removed. The username, specified in the Authorization header (am in this example), must be the same registered username that INSTALLed the method.

3.1.6 STDMDG

There is a kernel method called STDMSG (standard message) which can be invoked by a client but this is not the intended use. Instead, this method is used by Changeling as a fallback method, eg. whenever a client requests the invocation of a non-existent method, Changeling defaults to using the STDMSG method to return an error message to the client.

Also, other method scripts may use the STDMSG_essence procedure from the SDTMSG method script as a convenient way of generating a response message — the DOUBLE script in §3.1.4 uses STDMSG_essence in this way.

Aside: I have implemented several of the method scripts as two procedures: METHOD and METHOD_essence. Where METHOD_essence performs the core work of the method and is called with arguments like an ordinary Tcl procedure. While METHOD acts like an server-side stub which unpacks the arguments from the HTTP request and then calls METHOD_essence with these unpacked arguments to perform the core functionality of the method. The idea is that method scripts written in this way, like STDMSG, may be used, in part, by other method scripts. (There maybe some scope for developing an IDL based stub compiler which would wrap Changleing methods in CORBA IDL interface descriptions, see [Edwards].)

3.2 Standard library methods

These are methods that are suggested in the HTTP/1.0 Internet-Draft. They are automatically installed when the Changeling server starts-up.

3.2.1 GET

A client can invoke the GET method to ask the Changeling server to return a resource document. The HTTP/0.9 version of a GET request looks like:

```
GET http://ahost.ansa.co.uk:8081/documents/index.html
```

The HTTP/1.0 version of a GET request looks like:

```
GET http://ahost.ansa.co.uk:8081/documents/index.html HTTP/1.0
```

3.2.2 HEAD

A client can invoke the HEAD method to ask the Changeling server to return header information about a resource. Example invocation:

```
HEAD http://ahost.ansa.co.uk:8081/documents/index.html HTTP/1.0
```

3.2.3 POST

A client can invoke the POST method to ask the Changeling server to save the given Object-Body as a document resource. Example invocation:

```
POST /documents/am/hello.txt HTTP/1.0
Authorization: ChangelingIP am
Content-Length: 13


Hello world!
```

Where the URI /documents/am/hello.txt specifies where, in the document resource space, the document is to be installed. This is an example of an 'authorized request' where a registered user (am) is POSTing a document (hello.txt) into his own document space (/documents/am/). Non-registered users may post documents only to the temporary scratchpad space /tmp/. Posting to /tmp/ does not require authorization.

3.3 User methods

These are any methods installed by Changeling users.

4 Writing Methods Scripts

This chapter indicates how to write method scripts for Changeling. We begin by describing the execution environment of methods and then provide an example of how to author, install and use a method script. See the actual Changeling program code for more details.

4.1 The method interpretation environment

Method scripts are interpreted by a restricted, Safe-Tcl interpreter. Method scripts may use:

- Any the commands that are available in Safe-Tcl minus the commands related to Enabled-Mail.
- Any of the procedures from the other installed method scripts.
- The additional commands: `chgl_lookup`, `chgl_resource`, `chgl_installMethod`, `chgl_removeMethod`, `chgl_registerUser`, `chgl_unregisterUser`, `chgl_unlink`, `chgl_open`, `chgl_read`, `chgl_puts`, `chgl_gets`, `chgl_docIndex`, `close`, `getclock`, `pid`.

  The `chgl_` commands are explained below.

4.1.1 chgl_lookup

Given the name of a variable created from parsing the HTTP request, `chgl_lookup` returns the value of that variable. Many of the variables have the same names as the non-terminals found in the BNF rules defining an HTTP request. The most important of these variables are:

- `hostname`, `port` — the host name and port of the Changeling server
- `clientNetAddr` — the IP address of the connected client
- `Status_Code`, `Reason_Phrase`, `reasonDetails` — the status after parsing the HTTP request, eg. if some kind of parse error occurred this is detailed in these variable values
- `Method`, `URI`, `HTTP-Version` — parsed strings from the from the `Request-Line`
- `HTTP-Header-<non-terminal name>` – `HTTP-Header field-values`, eg. `chgl_lookup(HTTP-Header-Content-Length)` would return `164` after the request from §3.1.4 was parsed
- `Object-Body` — the whole `Object-Body` of the HTTP request

4.1.2 chgl_resource

Given a command option and the name of a local resource, `chgl_resource` returns information about that resource. `chgl_resource` understands the following command options: `readable`, `extension`, `size`, `lastModified`, `mimeType`.

4.1.3 chgl_installMethod

Given a method resource name and a string containing a method script, `chgl_installMethod` installs the method. `chgl_installMethod` fails if the resource name is invalid or of the request is not authorized. Installing a new method with the same resource name as an existing method will overwrite the existing method.

4.1.4 chgl_removeMethod

Given a method resource name, `chgl_removeMethod` removes specified method. `chgl_removeMethod` fails if the resource name is invalid or of the request is not authorized.

4.1.5 chgl_registerUser

`chgl_registerUser` extracts a username from the `Authorization` header in the HTTP request, establishes the IP address of the user, and registers the user as a username/IP address pair. `chgl_registerUser` also creates a document resource space `/documents/username/` and a method resource space `/methods/username/` for the user to `POST` and `INSTALL` into. `chgl_registerUser` fails if the username is already registered.

4.1.6 chgl_unregisterUser

`chgl_unregisterUser` extracts a username from the `Authorization` header in the HTTP request, and unregisters the user. `chgl_unregisterUser` also deletes the document and method resource spaces associated with the username. `chgl_unregisterUser` fails if the request is not authorized.

4.1.7 chgl_open, chgl_unlink

`chgl_open` and `chgl_unlink` are similar to the standard Tcl `open` and `unlink` except that they take resource names as arguments instead of file names. They fail if the request is unauthorized, eg. a user attempts to write to a document in another user's resource space.

4.1.8 chgl_puts, chgl_gets, chgl_read

`chgl_puts`, `chgl_gets` and `chgl_read` are similar to the standard Tcl `puts`, `gets` and `read` except that they operate on resources opened using the `chgl_open` command.

4.1.9 chgl_docIndex

`chgl_docIndex` returns a list of resource names of all the documents in Changeling's resource space, ie. all resources whose names begin `/documents/`.

4.2 Mapping resource space to local file space

Changeling handles requests for access to resources — document resources and method resources. Changeling maps the resource space to the local file space, ie. maps resource names to local filestore names. This mapping is hidden from Changeling clients. Figure §4.1shows the structure of the

resource space and figure §4.1 shows the structure of a corresponding file space — this mapping is somewhat configurable.

Figure 4.1: Structure of the Resource Space

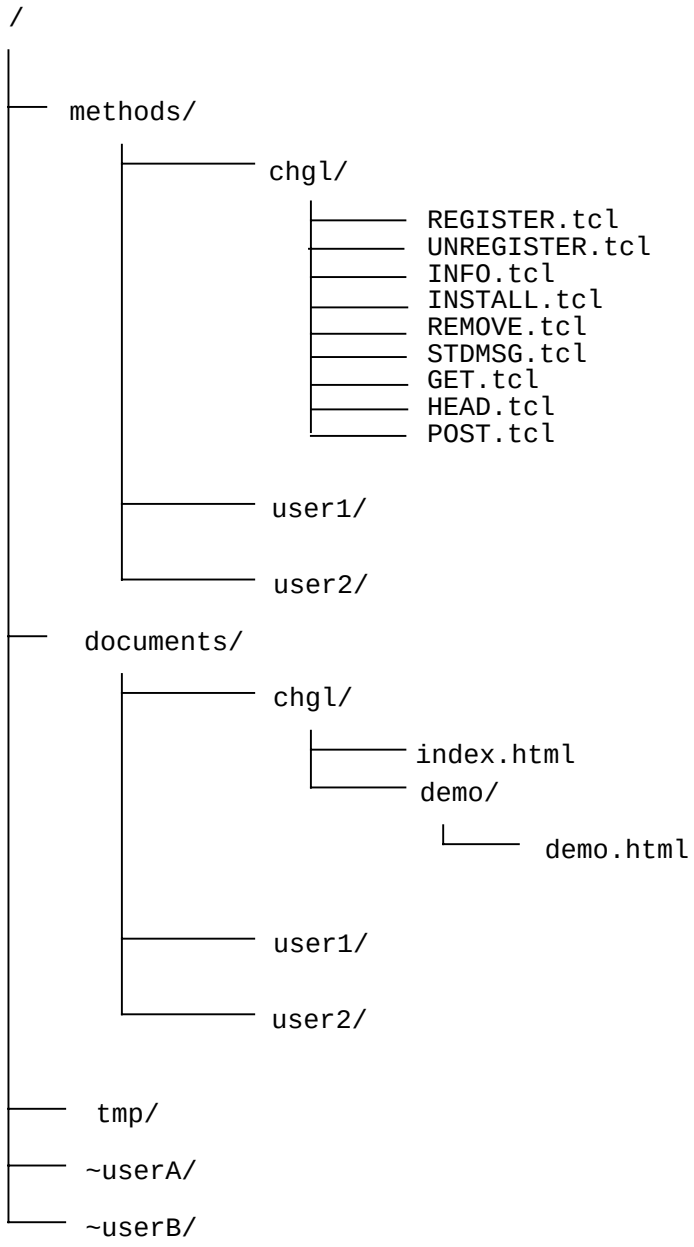
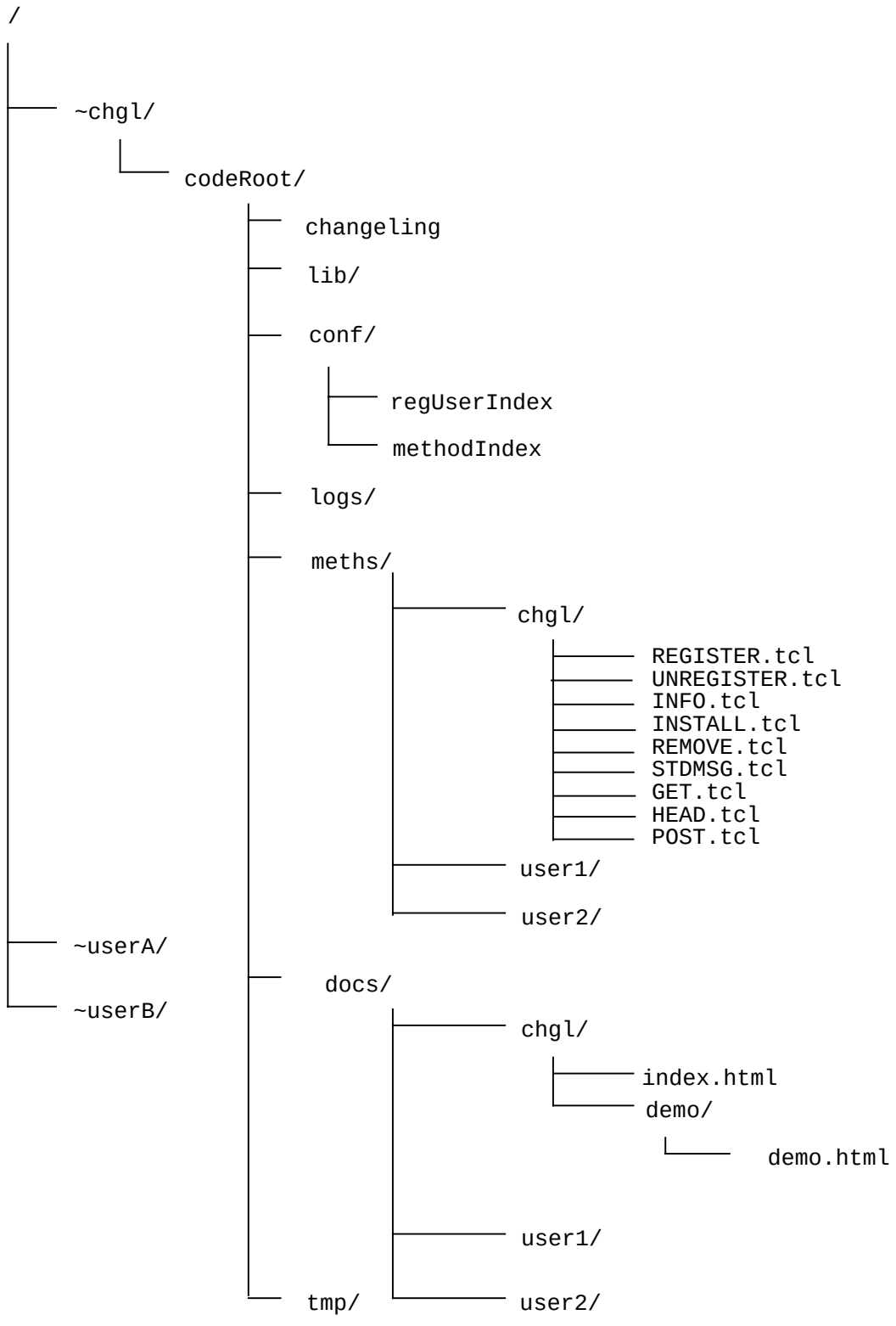


Figure 4.2: Structure of the File Space



4.3 Example lifecycle of a method script

This subsection describes how to write, install and invoke a simple Changeling method script.

4.3.1 Writing the method script

By way of an example, let us develop a method `WEBGREP` that returns a list of URIs of all the documents in Changeling's resource space that contain a given string. This method could be used to create a meta-data index to the documents on Changeling servers. Below we walk through the `WEBGREP` method script in stages.

```
proc WEBGREP {} {

 # Construct the URI prefix...
 set prefix “http://[chgl_lookup hostname]:[chgl_lookup port]”

 # Get the pattern to be matched from the HTTP request’s body...
 set pattern [string trim [chgl_lookup Object-Body]]
```

- A Changeling method definition should be implemented as a Safe-Tcl[1] procedure which has the same name as the method (`WEBGREP` in this example) and which takes no arguments.
- To preconstruct part of a URI, the `chgl_lookup` command is used to establish the `hostname` and `port` of the server.
- `chgl_lookup` is also used to read the `Object-Body` of the request. In coding this particular method we have decided to pass the method invocation argument – in this case, the pattern to be grep'd for – as a string in the `Object-Body`. Alternatively we could have decided to pass the argument as value in an HTTP-Header field.

```
# Ask Changeling for a list of all the documents in it’s resource space.
 set resourceList [chgl_docIndex]

 # Initialize the URI list to be returned...
 set uriList {}
```

1. To be exact, Safe-Tcl modified as described in §4.1.

- chgl_docIndex is used to build a list of resource names of the documents kept by this server.

```
# Loop through the document list...
foreach entry $resourceList {
   # Open document for reading...
   if [catch { chgl_open $entry r } fileId] {
      # Open failed --- ignore it
   } else {
      set contents [chgl_read $fileId] ;# Read the contents
      close $fileId
      # Does the pattern match the contents...
      if [regexp $pattern $contents x] {
         lappend uriList "$prefix$entry" ;# Yes, so note the URI
      }
   }
}
```

- We loop through the resourceList to process the listed documents.
- chgl_open is used to open the document for reading.
- If chgl_open throws an exception then it is caught and simply ignored.
- If the document is opened then the contents are read (chgl_read).
- If the specified pattern occurs in the document contents then an URI to the document is appended to the uriList.

```
# Create a temporary resource to hold the body of the HTTP response...
set tmp "/tmp/[pid][getclock]" ;# The name of the temporary resource
if [catch { chgl_open $tmp w } fileId] {
   # Failed to open temporary resource --- throw an error
   error "WEBGREP: chgl_open failed with $fileId"
}

# Write the URI list to the resource...
chgl_puts $fileId [join $uriList \n]
close $fileId
```

- Changeling expects the return value form a method to be a two element list. The first element is a string containing the complete header data for the HTTP response. The second element is the name of a resource which contains the Object-Body for the HTTP response.
- The code fragment above creates a resource (called $tmp) to contain the Object-Body for the HTTP response. (We use pid and getclock to create a unique resource name.) This resource is to be used only as a scratchpad for building the response body, and so it is created in the temporary (/tmp/) resource space (to be automatically deleted by a Changeling cron job after some expiry period).
- chgl_puts is used to write the list of URIs – which point to documents containing the specified pattern – to the $tmp resource.

```
# Create a string to hold the head of the HTTP response...
set head "HTTP/1.0 200 Ok
set head "${head}Server: Changeling/1.0
set head "${head}Content-Length: [chgl_resource size $tmp]
set head "${head}Content-Type: text/plain
set head "${head}Last-Modified: [chgl_resource lastModified $tmp]
```

- The code fragment above creates a string (called head) to hold the header data for the HTTP response.
- chgl_resource is used to establish the size and timestamp of the Object-Body data (stored in tmp).

```
  # Return from the method handing Changeling back the head and body of
  # the HTTP response to be sent back to the client...
   return [list $head $tmp]
}
```

- A Safe-Tcl procedure implementing a Changeling method definition should return a two element list. The first element (head in this example) is a string containing the complete header data for the HTTP response. The second element (tmp in this example) is the name of a resource containing the Object-Body for the HTTP response.

4.3.2 Installing the method script

The WEBGREP method script is installed by sending Changeling the following HTTP request.

```
INSTALL http://ahost.ansa.co.uk:8081/methods/am/WEBGREP.tcl HTTP/1.0
Authorization: ChangelingIP am
Content-Length: 1899

method script for WEBGREP
```

- The URI specifies the method is to be installed under Changeling's resource space / under the method space methods/ under the user space am/ with the method name WEBGREP with the .tcl extension.
- The Authorization header indicates the user am has sent the request. This username must correspond to the user space am/ in the URI. Also, this user must have been REGISTERed and the request must have been sent from the REGISTERed IP address (see §3.1.1 on how to REGISTER).
- Content-Length indicates the length of the HTTP request's Object-Body. The Object-Body contains the Safe-Tcl script implementing the WEBGREP method.

If everything goes okay the HTTP response will be:

```
HTTP/1.0 200 Ok
Server: changeling/1.0
Content-Length: 173
Content-Type: text/html
Last-Modified: Wed, 5 Apr 1995 13:37:21

<HTML>
<HEAD>
<TITLE>OK: 200</TITLE>
</HEAD>
<BODY>
<H1>OK: 200</H1>
Ok <P><HR>
Ok, the new method script /methods/am/WEBGREP.tcl has been installed <P><H
</BODY>
</HTML>
```

4.3.3 Invoking the method script

An HTTP request to invoke the installed WEBGREP method script looks like:

```
WEBGREP / HTTP/1.0
Content-Length: 8

Welcome
```

- Where `Welcome` is the pattern to be grep'd for.

The corresponding HTTP response looks like:

```
HTTP/1.0 200 Ok
Server: Changeling/1.0
Content-Length: 57
Content-Type: text/plain
Last-Modified: Wed, 5 Apr 1995 13:42:03

http://ahost.ansa.co.uk:8081/documents/chgl/index.html
```

- Where the URI in the `Object-Body` of the response points to a document which contains the pattern `Welcome`.

4.3.4 Removing the method script

The `WEBGREP` method script is removed by sending Changeling the following HTTP request.

```
REMOVE http://ahost.ansa.co.uk:8081/methods/am/WEBGREP.tcl HTTP/1.0
Authorization: ChangelingIP am
```

4.3.5 Errors

Tcl's usual exception catching mechanism `catch` can be used within Safe-Tcl scripts for catching errors. Changeling itself encloses each method invocation inside a `catch` statement. This means that if the method script fails to catch it's own errors then Changeling will catch them and return an HTTP response containing a plain text error report.

5 Associated Programs

This chapter introduces two programs – Changeling Client and Changeling Method Caller – which can be used with the Changeling Web Server.

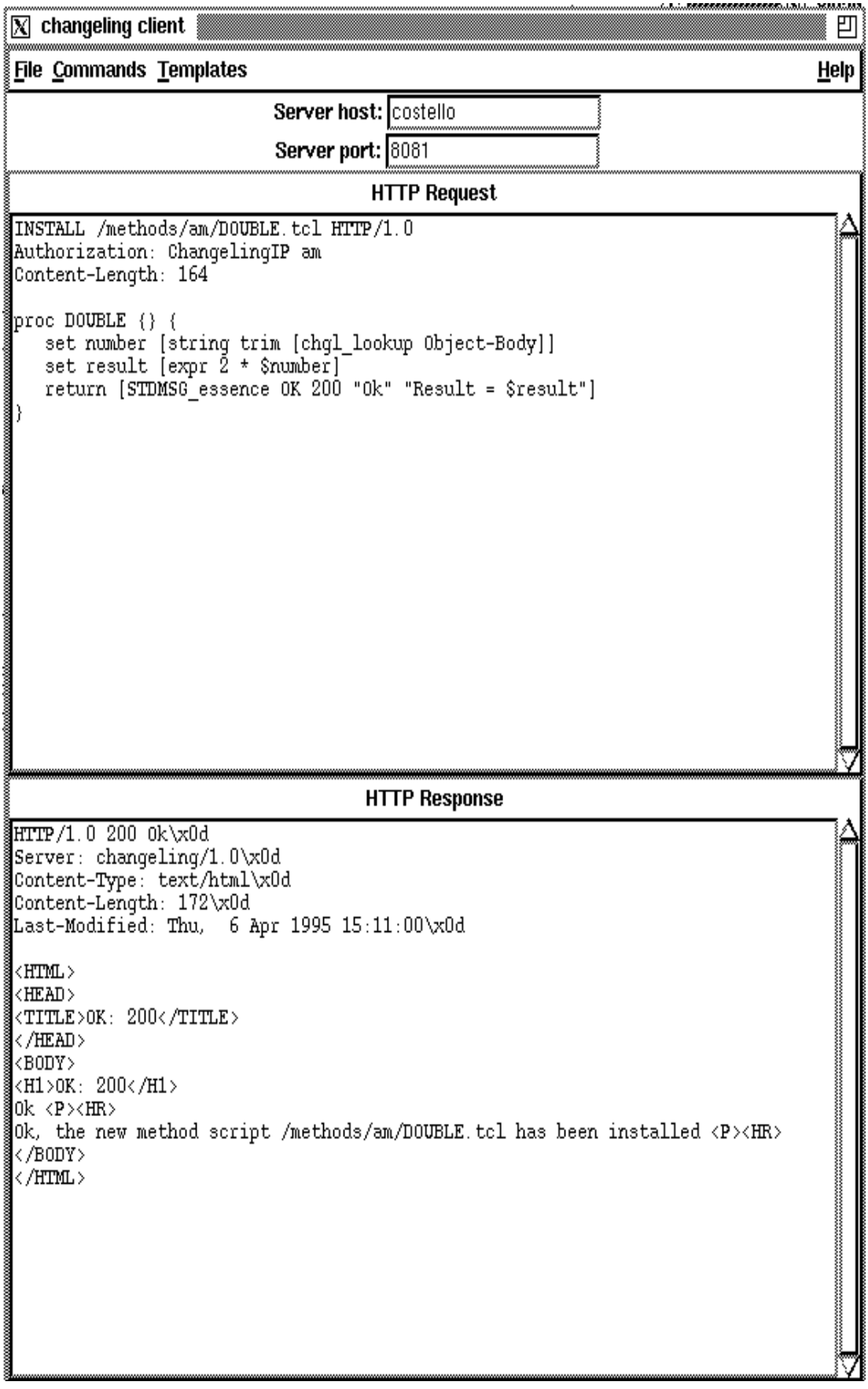
5.1 Changeling Client

Changeling Client is a simple, general-purpose client for Changeling Web Servers. Changeling Client is a Tcl-DP program (`chgl_client.tcl`) which can be used for constructing and sending HTTP requests to a Changeling Web Server, and for reading the resulting HTTP responses. Figure §4.1 shows the Changeling Client in action.

- HTTP requests are constructed in the upper text area.
- The user may choose from the `Templates` menu, any one of a number of predefined HTTP request templates (eg. there are templates for `REGISTER`, `INSTALL`, `GET`, `POST` etc. requests). The template text will be inserted into the HTTP request text area.
- The user may edit the (template) text in the HTTP request area.
- The `Import file` option in the `File` menu will import the contents of a file as the `Object-Body` of the HTTP request being constructed.
- The `Size request Object-Body` option in the `Commands` menu will size the `Object-Body` of the HTTP request and will, if a `Content-Length:` header line is present, write this size after the `Content-Length:` text.
- The `Send HTTP request` option in the `Commands` menu will send the text in the HTTP request text area to the specified Web server as an HTTP request.
- The `Server host` and `Server port` entry widgets under the main menu bar specify the fully qualified domain name and port of the target Web server.
- HTTP responses are displayed in the lower text area.

In figure §4.1 the Changeling Client has been used to `INSTALL` a new method `DOUBLE` on a Changeling Web Server. The upper text area shows the `INSTALL` request, and the lower text area shows the subsequent `OK` response.

Figure 5.1: chgl_client



5.2 Changeling Method Caller

Changeling Method Caller is program which makes method calls to a Changeling Web Server on behalf of a Mosaic Web browser. Changeling Method Caller is a Tcl-DP program (`chgl_methodCaller.tcl`) which interacts with Mosaic 2.5 through it's CCI interface. By itself Mosaic can only make `GET` and `POST` method calls, but in association with the Changeling Method Caller we can, in effect, have Mosaic initiate a method call to any Changeling method. The event-sequence diagram in figure §4.1 depicts how this can be done.

To recreate the sequence show in figure §4.1:

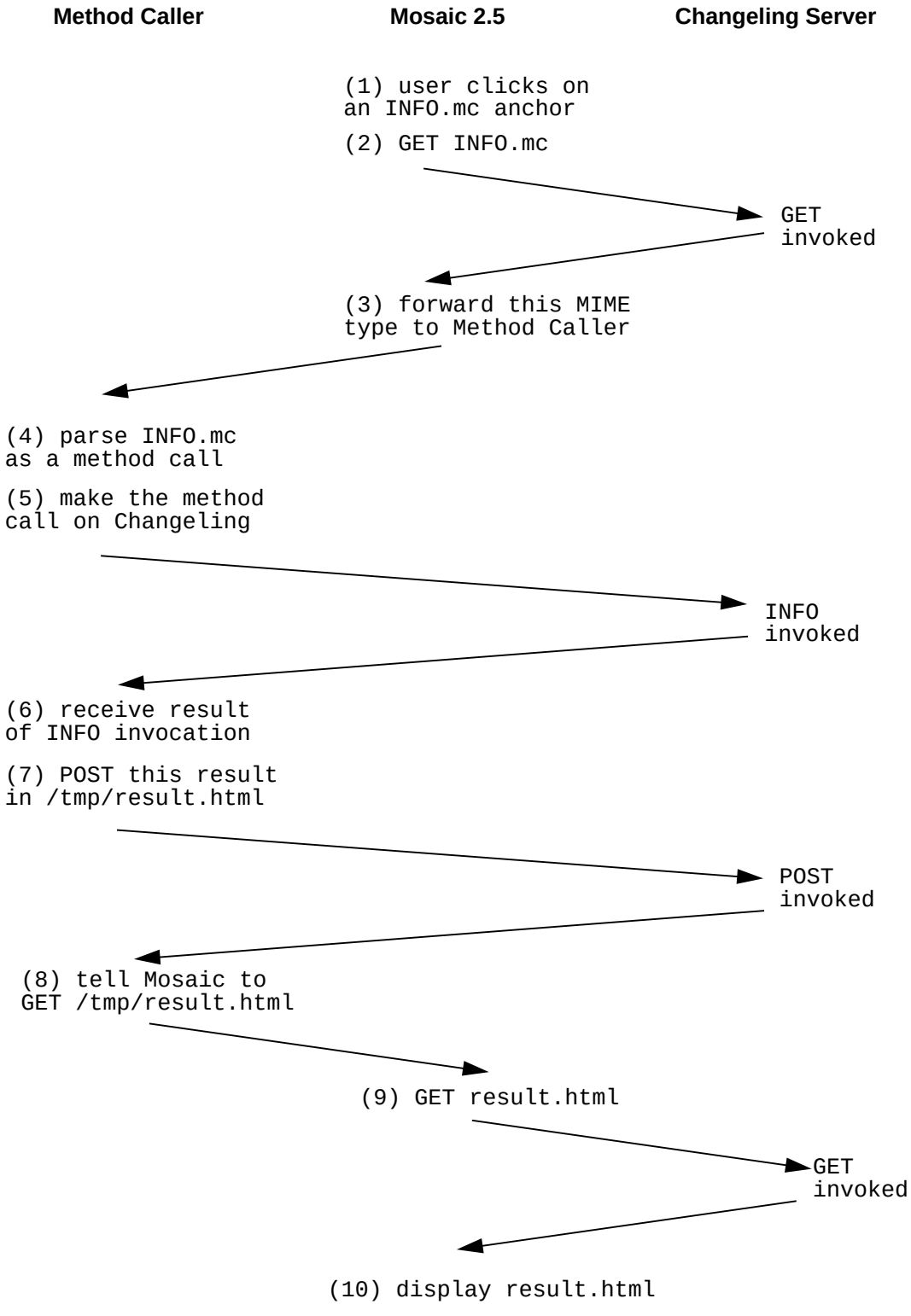
- the Changeling Web Server should be running
- Mosaic 2.5 should be running with CCI enables and set to port 8024 (a default expected by the Method Caller)
- then the Method Caller program should be started.

Figure §4.1 shows how Method Caller can be used to make it appear that Mosaic has called a Changeling method – `INFO` in this example. This is what happens:

1. The Mosaic user clicks on an anchor which points at the resource `INFO.mc` on a Changeling Web Server.
2. Mosaic `GET`s this document from the Changeling server.
3. The `.mc` document is mapped to an `x-methodCall` MIME type. Method Caller has asked Mosaic to forward any `x-methodCall` type documents, so this document is forwarded to Method Caller.
4. The `INFO.mc` document contains an `INFO` method invocation. The Method Caller program parses the `INFO.mc` document to extract the method invocation.
5. Method Caller then makes this method invocation on the Changeling server.
6. Method Caller receives the result of the invocation of `INFO`. (Unfortunately the present CCI interface does not provide any direct means of passing Mosaic a document to display, so at this stage Method Caller cannot simply pass Mosaic a document containing the results of the `INFO` invocation, instead...)
7. Method Caller asks Changeling to `POST` the results in a temporary document `/tmp/results.html`.
8. Method Caller asks Mosaic to `GET` the results document.
9. Mosaic `GET`s the results document.
10. Mosaic displays the results document, ie. displays the results of the `INFO` method invocation.

In contrast to the convoluted way that we do it for Mosaic, the new Sun Web/Net client HotJava[HotJava] promises the kind of extensibility needed to directly support method calls to arbitrary methods on extensible Web servers like Changeling.

Figure 5.2: Using Changeling Method Caller



A derivative of the Changeling Method Caller has been used to prototype resolution and lookup indirection when accessing Web documents. This is an aspect of the `URN->URC->URL` resolution and lookup work being done by Mark Madsen [Madsen95].

6 Acknowledgements

The author would like to thank the other members of the ANSAweb team, Nigel Edwards, Mark Madsen and Owen Rees, for their help.

References

[McClengahan]

Ashley McClenaghan, *Scripts and Agents: Introduction and Work Proposal*, 1434.00, APM Ltd., Cambridge UK, 1995

[Tcl/Tka]

John S. Ousterhout, Tcl and the Tk Toolkit, Addison-Wesley, 1994

[Tcl/Tkb]

Brent Welch, P*ractical Programming in Tcl and Tk*, Draft, Xerox-PARC, 3333 Coyote Hill Road, Palo Alto CA, 94304, USA, August 1994

[MIME/Safe-Tcl]

Nathaniel Borenstein, Marshall T. Rose, M*IME Extensions for Mail-Enabled Applications: application/Safe-Tcl and multipart/enabled-mail*, November 1993

[Safe-Tcl]

Nathaniel Borenstein, Marshall T. Rose, *(Safe-Tcl) swish man page*, October 1993

[Email/Safe-Tcla]

Nathaniel Borenstein, E*Mail With A Mind of Its Own: The Safe-Tcl Language for Enabled Mail*, First Virtual Holdings, Inc., 25 Washington Avenue, Morristown, NJ 07960, USA

[Email/Safe-Tclb]

Nathaniel Borenstein, Marshall T. Rose,*A Model for Enabled Mail*, July 1994

[Madsen95]

Mark Madsen, *Meta-Informtion Management Overview*, 1414.01, APM Ltd., Cambridge UK, 1995

[Marshall]

Lindsay Marshall, *Jungle*, <URL:http://catless.ncl.ac.uk/Programs/Jungle/>, 1995

[Edwards]

Nigel Edwards, *A CORBA IDL Compiler for the World Wide Web*, 1419.01, APM Ltd., Cambridge UK, 1995

[ANSA]

ANSA, *1994-1996 ANSA Workplan*, 1275.02, APM Ltd., Cambridge UK, September 1994

[ReesEdwards]

Owen Rees, Nigel Edwards, *An Overview of the Information Services Framework*, 1306.00.09, APM Ltd., Cambridge UK,February 1995

[Zimmermann]

Philip Zimmermann, *PGP User's Guide, Volume 1: Essential Topics*, Boulder Software Engineering, 3021 Eleventh Street, Boulder, Colorado 80304, USA, October 1994

[HTTP]

Tim Berners-Lee, R T Fielding, H Frystyk Nielsen, *Hypertext Transfer Protocol --- HTTP/1.0*, Internet Draft, December 1994

[HotJava]

Sun Microsystems Labs., *HotJava Home Page*, <URL:http://java.sun.com/ >,March 1995