The I*Tea$^{\text{TM}}$ Application Server: An Overview

João Luís *Jorge Nunes*
Luís Miguel Campos

Dev.Web
Rua de Borges Carneiro, 61-A
1200 Lisboa
jpsl@devweb.pt, jfn@devweb.pt, lcampos@devweb.pt

June 2000

Abstract

I*Tea is a Web application server entirely written in Java, scalable, and small in size. It supports the **Tea** scripting language, allowing fast prototyping and development of web applications which can evolve up to large, scalable and robust applications. It supplies a few pre-built client context and session management policies giving the application programmer the choice to choose between client-server programming models, or explicit client context management. **I*Tea** can generate any kind of output the programmer desires (HTML, WML, XML, etc...) and comes with object-oriented libraries for HTML rendering and form processing. Applications typically range from small scripts embedded within HTML pages to large business management applications on Enterprise JavaBeans[1]technology.

1 Introduction

CGI applications usually require the programmer to explicitly manage application state storing outsite of the mechanisms provided by the programming language chosen. Web clients are simpling following links (HTTP GET URL requests) or submitting forms (HTTP GET or POST requests). The programmer has to identify each one of them client and explicitly manage

[1]Java 2 Plataform Enterprise Edition, Java Servlet API, Java Server Pages, and Enterprise JavaBeans are registered trademark of Sun Microsystems inc.

the data structures which store the application state, save them on hidden form fields, or, serialized/de-serialize them into/from from disk files, etc... A **web application server** simplifies most of this job.

There are several **web application** programming frameworks and commercial products which can help the programmer on this. **I*Tea** is one of them, for which we will try to briefly describe its features and characteristics.

2 Software Architecture

In this paper we will describe the **I*Tea** version which interfaces with browsers through **CGI** technology.[2]

An operating **I*Tea** application site usually involves the following software:

- *Web Client* - A *Client* is usually a user's browser which performs HTTP GET or POST requests.

- *Web Server* - A server process which receives HTTP requests and takes appropriate action.

- *cgi2itea*- A small executable **CGI** program wich interfaces the *I*Tea Server* process with the *Web Server* .

- *I*Tea Server* - A standalone process, which receives requests from the *cgi2itea*, executes them, and passes the response back to *cgi2itea*.

I*Tea is entirely written in **Java** , inheriting all the advantages of such a technological choice. Its runtime environment occupies a surprisingly low disk space (approx. 1MB, including the **Tea** runtime environment).

The *I*Tea Server* is a multi-threaded **Java** process which can be modelled as a set of software components (data and threads) which communicate between them:

- *Client Data* - A simple data storage hastable, private to each *Web Client* . It can only be accessed through a *set/get key value* hashtable like API. The values store in this hashtable can be simple **Tea** data objects (integers, strings, etc...)

[2]Integration into a web server as a **servlet** is under study, but meanwhile, the **CGI** version is compatible with most web servers, and very well tested in large full production environments.

More complex data structures have to be serialized/de-serialized if to be stored in such a way. For those situations we suggest using a separate *Tea Interpreter Context* for each *Web Client* .

- *Tea Interpreter Context* - Object binding context for executing **Tea** code (ie. the bindings between variables/values for **Tea** scripts executed under the *I*Tea Server*).

  Recall that the **Tea** language supports integers, floats, strings, symbols, lists, vectors, hashtables, global variable bindings, nested code blocks (code with local variable bindings), functions as first class objects (`lambdas`), programming modules (files defining public and private functions and variable bindings), and object oriented programming (class based with single inheritance). It has a simple syntax and powerful semantics that allow a programmer to extend the language based on the language itself. As such, a program written in the **Tea** language can have very complicated object structures. (See documentation on the **Tea** language for more information).

- *Main Server Thread* - The *Main Server Thread* accepts connections from the *cgi2itea*and places them in the request queue. (The number of pending requests is configurable by the programmer.)

- *Thread Pool* - The *Thread Pool* contains a configurable number of idle *threads* waiting for requests to be placed in the *Request Queue.* When an idle *thread* acquires a pending request, it:

  1. Identifies the *Web Client* by the HTTP cookie provided. If the *Web Client* has no cookie, then it is a new *Web Client* and a new *Client Context* is created. A configurable operating mode parameter specifies if the *Client Context* is given a new *Tea Interpreter Context*, or to reuse one. (Configurable parameters limit the number of *Client Contexts* allowed and their idle lifetime.)
  2. Parses a document specified by the URL and configuration parameters. This document can be a single **Tea** script (wich is executed), or a textual document (HTML for example) with embedded **Tea** script blocks. Each **Tea** script block runs under the appropriate *Tea Interpreter Context* and has access to the *Client Data* (private for this *Web Client*) through the **I*Tea** API.
  3. Flushes the data outputed during document parsing back to the calling *cgi2itea.*

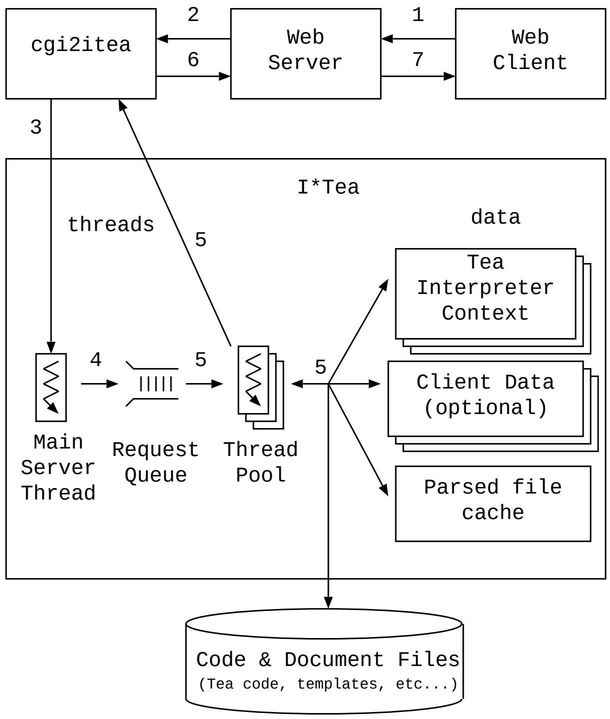

Figure 1: HTTP request interaction

The *I*Tea Server* is dynamically linked with the **Tea** interpreter code (also written entirely in **Java**) and with other **Java** class libraries which give it access to whatever the application programmer might need (JDBC for relational database access, TCP/IP APIs, etc...)

A typical interaction with a *Web Client* goes like described in figure 1.

1. The *Web Client* issues an HTTP request to the *Web Server* (by following a link or submitting a form).

2. The *Web Server* receives the HTTP request, and, if the URL requested corresponds to a *cgi2itea*invocation, spawns the process and passes it the information of the request (as defined by the **CGI** specification).

3. The *cgi2itea*spawned by the web server reads the HTTP request information supplied by the *Web Server* , opens a TCP/IP connection to the *I*Tea Server* and passes it this information.

4. The *Main Server Thread* from the *I*Tea Server* accepts the connec-

tion from *cgi2itea*, reads the request parameters and places them in a request queue.

5. An idle *thread* picks up the request, selects the appropriate *Client Data* and *Tea Interpreter Context* for this *Web Client* , and parses the document specified by the URL, executing the whole document as **Tea** script, or expanding text embeded **Tea** script blocks.

   The **Tea** script blocks can access the request parameters, access the *Web Client*'s private *Client Data* and manipulate its own variables in its *Tea Interpreter Context* and dynamically import and use **Tea** or **Java** libraries. Requests are serialized for each *Web Client* , so there is usually no need to worry about concurrency when manipulating these resources.

   At any time, the script can output data through **I*Tea** API function calls and libraries to be sent back to the *Web Client* .

6. The *cgi2itea*reads the output from the *I*Tea Server* , and passes it back to the *Web Server* , closes its connections and dies.

7. The *Web Server* passes the *cgi2itea*output back to the browser (which should render it on the user's browser window), closes its HTTP connection, and it is ready for another request.

3 Client Session Management

One of the key strengths of **I*Tea** lies in its pre-defined automated policies of *Web Client* context management.

I*Tea automatically identifies every browser with an HTTP cookie key (setting it only once at the beginning of the client's session), and from there on it manages its internal context data pools for each HTTP request automatically. The programmer has no need to worry about identifying separate client sessions.

I*Tea provides a simple *get/set key value* API for storing simple *Client Data* state information. In some cases, this is enough to easily store client dependent application state. In such cases, the programmer can configure the *I*Tea Server* to have a limited pool of *Tea Interpreter Context* , which can be used to cache data and code common to all users.

In the cases where the effort for serializing individual *Web Client* application state is considered high, it is recommended that the programmer configures the *I*Tea Server* to allocate a private *Tea Interpreter Context* for each *Web Client* session. In this case, the programmer sees a full client-server programming model, where, each *Web Client* appears to have a single server process just to itself.

As a simple example of **I*Tea** code for a simple HTML counter. Tme implementation is exptrmely simple. Just a library file `counter.tea`

```
define counter 0
# Like C's "static int counter = 0;"

global incrementAndPrint () {
   set! counter [+ $counter 1]
   doc-print $counter
}
# Like C's "void incrementAndPrint() {
#    counter = counter + 1;
#    printf("%d", counter);
# }"
```

and the server side parsed HTML file `counter.html`

```
<HTML><BODY>
Counter value is: <!--@
    import "counter.tea"
    incrementAndPrint
--></BODY></HTML>
```

Each *Web Client* sees its own counter, and every time they attempt to read `counter.html` **I*Tea** identifies the *Web Client* and chooses apropriate context for the variable `counter`.

In this latest case, as **Tea** and **Java** libraries are dynamically imported for each *Web Client* session, for large software projects the time spent on importing vast library definitions can become high. **Tea** supports a library *import-on-demand* feature to ease such problem, but, besides that, **I*Tea** supports an operating mode where each *Tea Interpreter Context* can be reused for another *Web Client* when the previous client has releases the session (though logout or timeout). Such a new session already has all the libraries from the previous *Web Client* imported.

4 Software Libraries

I*Tea is supplied with object oriented libraries written in **Tea** which allow

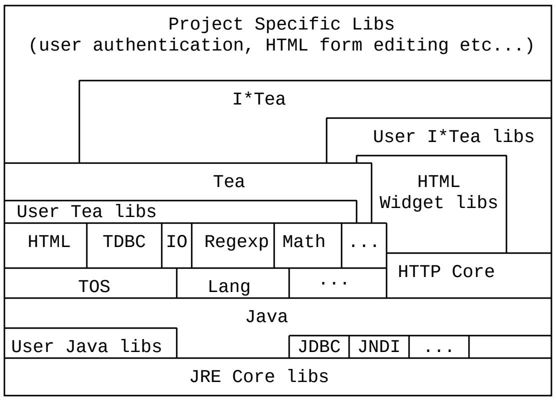

Figure 2: I*Tea Software Libraries

representation of HTML documents (including form elements) and processing of HTTP requests as event trigger functions or object method callbacks.

These libraries can be extended (for example, using inheritance and delegation programming patterns) allowing the programmer to easily build problem specific object oriented models, adding problem specific logic. Several object-oriented programming patterns have already been developed to help editing, validating and displaying form data. Also, HTTP file upload, *on-the-fly* generation of graphic images and seamless integration with **Java Applets** and client side **Java Script** has been done with full success.

On the backend side of the application, the full power of the **Java** language (existing libraries and standard APIs) is available through the **Tea** language. (Writing of mapping APIs onto the **Tea** language might be required).

5 Fast Development and Maintenance

Tea is an interpreted scripting language recommended for fast prototyping, and supports many programming paradigms which allow it to scale well for larger software projects. An *I*Tea Server* parses documents dynamically, so there is no need to recompile anything after changing **Tea** source code.

I*Tea has a policy of caching parsed documents, only re-interpreting them if the modification date has changed. Careful use of this feature allows expert site administrators (which also know the software architecture of the application and how **Tea** caches libraries) to change code/patch on line applications without any visible disturbance for on-line clients.

6 Performance and Scalability

Running over JDK 1.2.1 on Solaris 2.6 on a SPARC CPU at 300Mhz, an *I*Tea Server* version 1.4 takes aprox. 40 miliseconds to process an HTTP request, rendering simple HTML pages through its HTML library. This value can be taken as a reference value of **I*Tea** overhead for each request. For a more complex page, the time will increase with the number of HTML objects rendered and backend processing required.

A single *I*Tea Server* based on a CPU at 400Mhz has been used in call-center helpdesk situations with more than 40 simultaneous clients, with an average of 3 HTTP requests per second (typical request frequency ranges from 1 to 5 requests per second).

It performs acceptably well, and it can scale as well if needed. Besides brute-force CPU power increase, **I*Tea** applications can achieve scalability through:

- Vertical Scalability - If the **Java Virtual Machine** and native operating system support multi-processing through **Java** threads, then the performance of the *I*Tea Server* scales through symmetric multi-processing processing power increase.

- Horizontal Scalability - Replication of an *I*Tea Server* running identical copies of the same application accessing the same backend (a relational database) has been successfully tested over a network of cheap PCs. The load balancing intelligence has been coded in the *cgi2itea*.

7 The Future into J2EE[1]

I*Tea was conceived in early 1997 to replace **CGI** programming and other resource consuming application server alternatives available at the time.

Today, the standardization of the Java Servlet API and Java Server Pages into the Java 2 Platform Enterprise Edition by Sun Microsystems point the way to go for large multi-tiered applications, with middle tiers on Enterprise Java Beans. Nevertheless we feel clearly the need for a fast proototyping scripting language like **Tea** , specially for developping Web front-ends.

For this reason, **Tea** is being extended for writing Enterprise Java Beans client applications, and running existing **I*Tea** application code under a Servlet/Java Server Pages environment is under study (already demonstrated experimentally).

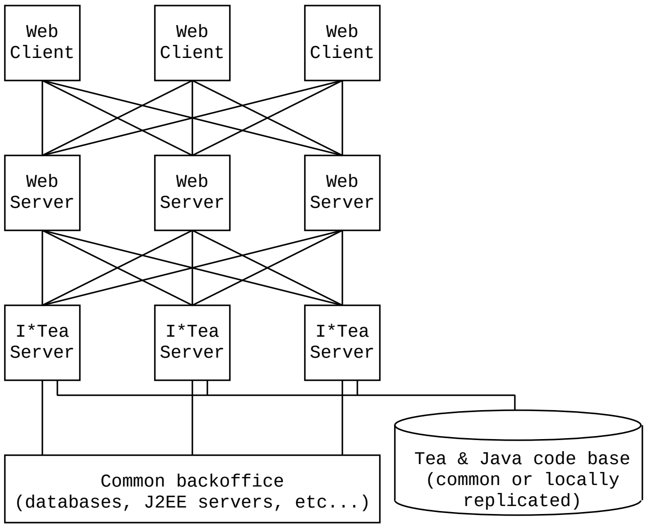

Figure 3: I*Tea Horizontal Scaleable Architecture

8 Summary

I*Tea is a low resource consumption (disk space) application server for fast development of Web Applications using the simple but powerful **Tea** scripting language.

The power of the built-in web session management policies of **I*Tea** , combined with the power of the **Tea** language and being entirely based on the **Java** platform, allows quick web prototypes evolve as commercial quality production applications. It has been successfully tested in large projects (more then one 300000 lines of code in a single project) such as:

- Web HomeBanking
- Internet Service Provider Management (including both customer self-care, call-center, and enterprise management interfaces)
- On-Line Brokerage
- Portal management
- Custom Enterprise Management Applications (Customer/service database interface, billing, call-center web support, etc...)