Issues on Building T++, a Tool for Web Application Development with C++

Antonio Soares de Azevedo Terceiro
terceiro@im.ufba.br

Departamento de Ciência da Computação.
Instituto de Matemática – Universidade Federal da Bahia.
Campus de Ondina – Av. Ademar de Barros, Ondina – 40170-110.
Salvador, Bahia, Brazil

ABSTRACT

As the demand for web applications grows, so does the demand for tools that support them. As a general rule, such tools extend general purpose programming languages, like Servlets/JSP [2] does for Java [4], or define their own programming language, like PHP [3]. But there is no established engine for web applications written with C++. This work presents technical challenges that were faced when developing T++, an engine that supports web application development with C++.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features—*Frameworks, Polymorphism, Concurrent programming structures, Classe sand objects*

General Terms

Design, Languages

Keywords

Web Application Development, C++, Object Orientation

1. INTRODUCTION

The demand for web applications is getting more and more intense. They require less resources from clients, since business logic resides in application servers. This characteristic also facilitates application upgrades: they can be done simply by replacing the code that resides in the application server.

Two kinds of tools have been used for web application development: programming languages designed specifically for web applications, like PHP [3]; and extensions of general purpose programming languages, like Servlets/JSP [2], a Java [4] extension. The latter – general purpose programming language extensions – can be used to port applications originally designed for desktop systems to the web environment.

The C++ language [9] has been used in the development of many types of applications, from simple tools to complex information systems, including scientific and industrial applications. However, up until now, there is no established engine for web application development in C++. The lack of such a tool complicates the development of new C++ applications for the web environment; moreover, it hampers the deployment of a solid bulk of C++ applications on the web. This work briefly presents T++, an engine designed for running web applications written in C++. T++ comprises two parts: a programming language that works as an interface for web application development in C++, and an execution engine. In the present work we focus on the execution engine implementation, while aspects related to the programming language interface are detailed in [5]. Specially, we discuss some challenging issues regarding object-oriented implementation in C++, that have been subject of research during T++ development.

OOPSLA'03, October 26–30, 2003, Anaheim, California, USA.
ACM 1-58113-751-6/03/0010.

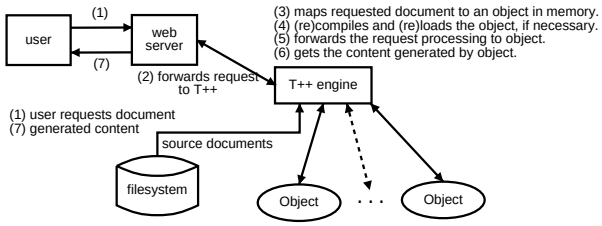

Figure 1: The processing of a request to a T++ document

2. TOOL DESCRIPTION

T++ is a tool for web application development in C++. It is free software, licensed under the GNU General Public License [1]. T++ comprises a programming language and an execution engine. Running T++ requires a GNU/Linux system, an Apache web server, and GNU Compiler Collection (`gcc`) with support to C++.

Figure 1 shows a high-level representation of the behavior of a T++ application. From the users' point of view, T++ documents are ordinary ones and are available for users request through their web browsers (1). Users send HTTP requests, and receive arbitrary content as result.

The web server forwards the request to the T++ engine (2). The T++ engine, then, maps the requested document to an object in memory (3). If it is is necessary, this object is (re)built (4) before it is used to process the request (5). Finally, output generated by this object (6), often in HTML format, is sent back to the user (7).

The T++ programming language supports writing T++ documents, mixing up static content - often HTML code – and C++ statements and expressions. C++ code is embedded in special language blocks, surrounded by special delimiters. There are blocks for code execution, expression evaluation and others.

3. TECHNICAL ASPECTS

C++ code is compiled into native code, unlike other languages commonly used for web application development. Java, for instance, is compiled into intermediate code, while PHP is purely interpreted (but there are optional features for caching precompiled

intermediate code).

Restrictions related to working with native code, and the absence of a virtual machine to support portability raise several issues that must be handled when developing a tool like T++. In the next subsections, we discuss some important issues and describe the proposed solutions to deal with them.

3.1 Automatic C++ source code generation and compilation

The objects shown in figure 1 are dynamically created, based on T++ documents that contain static content (usually HTML code), C++ statements, and other types of C++ code (classes, methods, and preprocessor directive declarations, for example). Their translation mixes both a template skeleton and the content read from the source document, creating a class that represents the T++ document.

The generated class extends an abstract `Page` class, which provides the standard interface for T++ documents. The translated content generates the body of a method called **service()** in the target class, for answering users' requests.

After the class is generated, it is compiled into a shared library (`.so` in UNIX systems), which will contain the class definition and other resources required for dynamic loading.

3.2 Dynamic class loading

T++ needs a dynamic loading mechanism, because users can change document source code after T++ and web server are already running, and restarting web server at each document update is a very bad idea. In this case, the class has to be generated once again, recompiled, and reloaded in order to make the objects in memory correspond to the code in source documents. Since C++ is compiled, dynamic loading of classes is not as easy to implement as it could be in interpreted or even semi-interpreted programming languages.

T++ dynamic loading mechanism is based on a technique for dynamic class loading in C++ proposed by Norton [8]. This technique is presented below.

Each class is compiled into its own shared library. Native library calls are used to load object code from the shared library file and to get pointers to named symbols inside the module loaded.

In order to process the request for a document, T++ needs to get an instance of the corresponding class, defined in the shared library. However, the only available information is that such class is a subclass of the `Page` class. Furthermore, it is not possible to refer to types at runtime in C++. These restrictions lead to the following implementation:

- The generated class extends the `Page` class, that defines the standard interface that all document classes must conform to.
- The generated source file has a public function, with a predefined name, let us say, *getInstance()*, that returns a pointer to an instance of the class, implementing the Singleton design pattern.
- After loading the library, the dynamic loading mechanism gets a pointer to *getInstance()*, and calls the function. The returned pointer is then used to forward the request to the object it points to.

3.3 Shared memory allocation

Web servers are designed to be robust and scalable, since they can host applications used by hundred, or even thousands of users. To achieve these requirements, web servers are by nature multiprogrammed: the main web server has several child processes that deal with users requests.

Document instances must be shared between all server's subprocesses, so that all users will be served by the same instance of the class corresponding to the requested document. Another reason for sharing unique instances between subprocesses, is that application programmers may wish to keep information between successive requests for some document in the document's object internal state. These requirements would pose unnecessary additional complexity if there were several instances of each document in memory.

To implement such functionality, we must:

- Provide a shared memory allocation mechanism, and some means to instantiate objects in shared memory.
- Provide a synchronization mechanism, since there will be a lot of race conditions using unique instances, specially if they keep internal state between requests.
- Provide a programming library for using shared memory, like shared memory versions of C++ containers and shared memory allocation operations.

An interesting approach for memory management was proposed by Meyers [7], using memory heaps and overloading C++'s `new` and `delete` operators. A technique that can be used for attaching this approach to C++ standard template library containers, redefining their memory allocation strategy, was described by Austern [6]. In T++, the shared memory allocation problem is solved by combining these two techniques, wrapping shared memory allocation with memory heaps and defining C++ allocators that are used when collections of objects are needed.

4. CONCLUSIONS

T++ is a tool that helps web application development, either for creating new applications from scratch, or for reusing existing C++ source code for providing a web interface for existing, conventional, applications. T++ represents an important contribution, since, as far as we know, there is no established engine for web application development in C++.

The reported design and implementation issues are interesting by themselves, and putting them to work together has been really challenging. The resulting knowledge will certainly benefit the design of other types of applications that have some requirements in common with a web application server mechanism, like *on demand plugin* loading mechanisms, component architectures with dynamic component loading, memory sharing components, among others.

Further information about T++ can be found at [5].

5. REFERENCES

[1] The GNU general public license. `http://www.gnu.org/licenses/gpl.html`.

[2] Javaserver pages technology. `http://java.sun.com/products/jsp`.

[3] PHP Hypertext Preprocessor. `http://www.php.net`.

[4] The source for Java technology. `http://java.sun.com`.

[5] T++ website. `http://tplusplus.sourceforge.net`.

[6] M. Austern. What are Allocators good for? *The C/C++ Users Journal*, Dec. 2002.

[7] N. C. Mayers. Memory management in C++. *C++ report*, July 1993.

[8] J. Norton. Dynamic class loading for C++ on Linux. *Linux Journal*, May 2000.

[9] B. Stroustrup. The C++ programming language. `http://www.research.att.com/~bs/C++.html`.