TOME Component Model

The TOME Team
Curtis "Fjord" Hawthorne
Craig Miller
Clint Olson
fREW Schmidt

October 28, 2006

Contents

List of Figures

1 Introduction

1.1 Purpose of Document

The purpose of this document is to specify the TOME system and the component model it uses.

1.2 Background

In December of 2003, several students on Dorm 41 started a system called TOME. The basic idea is that at the end of the semester, instead of everyone selling their books back to the bookstore, they all donate them to central repository. Anyone on the floor can then check out whatever books they need free of charge for a semester.

The advantage of having a computer-based system to keep track of all those books is easy to see, and one has been under development ever since the start of TOME. Since its humble beginnings as a quick solution over Christmas break, the system has grown to well over 3,000 lines of Perl code as well as HTML templates, a well-planned database schema, and significant documentation. The system not only has comprehensive facilities for tracking books and patrons, but also keeps tabs on what books are used for what classes and other alternatives to purchasing new books.

1.3 Context

This component model is a part of the larger suite of documentation for the TOME project. This document will describe the component architecture in detail. For the application model, please reference the Application Model Document. Overall project information can be found in the Project Brief and Project Plan. A later document describing the test architecture will be made available soon.

All current documentation will be available in the Trac environment (See Section 1.4).

1.4 References

All project data will be stored in a combination Subversion repository and Trac environment. All of this will be made viewable at the following URL:

`http://enosh.letnet.net/trac/tome`

References will be made to a number of standard Perl modules such as CGI::Application, Template Toolkit, and many others. Documentation for these modules is available at the following URL:

`http://search.cpan.org/`

2 Component Specifications

2.1 admin.pl

2.1.1 Role

The role of admin.pl is to load the system and serve as a starting point for the webserver

2.1.2 Responsibilities

admin.pl needs only to load TOME::Interface and hand control over to the CGI::Application framework.

2.1.3 Exclusions

admin.pl has no intrinsic understanding of anything, really. It only hands control over to the other modules.

2.1.4 Collaborators

TOME::Interface collaborates very closely with admin.pl. admin.pl loads TOME::Interface and then relinquishes control.

2.1.5 Key Scenarios

admin.pl is called by the web server admin.pl will load, and, in turn, load TOME::Interface.

2.1.6 Properties

admin.pl will be a standard text file. It will run everything under Perl's taint mode. It will use strict. It will use warnings.

2.1.7 Creation, Existence, and Management

admin.pl will create a single instantiation of the TOME::Interface class. The web server (usually Apache) will be responsible for calling admin.pl. Multiple instantiations of admin.pl (and therefore TOME::Interface and any other classes) can exist only if Apache runs multiple copies, which is expected.

2.1.8 Resource Usage/Management

No specific requirements exist for this component. Resources will be limited only by the web server.

2.1.9 State/Session/Context Management

This component has no particular knowledge of states, sessions, or contexts.

2.1.10 Data Storage

This component has no intrinsic need for its own data storage.

2.1.11 Performance

This component does not have any specific performance or speed requirements. Faster is better, but that is determined entirely by the machine on which it is run. The only impact of faster or slower performance is the quality of the end user's experience.

2.1.12 Packaging

admin.pl is packaged as a single, executable Perl file.

2.2 TOME::Interface

2.2.1 Role

The role of TOME::Interface is to control almost all interaction between the user and TOME. Nearly every method is a CGI::Application runmode, with the exception of a few helper methods

2.2.2 Responsibilities

TOME::Interface is responsible for some of the set up of the CGI::Application framework. It has all of the runmode methods that CGI::Application uses to dispatch requests served by the web server through admin.pl. It should check all user input using the CGI::Application::Plugin::ValidateRM module. It should also use ValidateRM's features to give good user feedback if something is wrong with their input. ValidateRM can nicely refill the form the way it was before and provide error messages.

2.2.3 Exclusions

TOME::Interface shouldn't know any details about the internal database structure or what things look like when they're displayed by the template.

2.2.4 Collaborators

TOME::Interface collaborates very closely with admin.pl. admin.pl loads TOME::Interface and then relinquishes control. TOME::Interface also interacts with CGI::Application::Plugin::ValidateRM to validate user input, CGI::Application::Plugin::Forward to transfer control from one runmode to another when necessary, and Crypt::PasswdMD5 to provide secure password authentication. It also interacts with the TOME module in that it is a base class of TOME.

2.2.5 Key Scenarios

admin.pl loads TOME::Interface and calls the run method Since the run method is not actually a part of TOME::Interface, it has no direct responsibilities in this case. However, the run method sets in motion a chain of events controlled by CGI::Application including calling the setup and cgiapp_prerun methods, which are implemented by TOME::Interface.

A runmode dispatch occurs TOME::Interface contains all of the runmode subs that CGI::Application will dispatch to. A runmode dispatch will simply run one of these methods. When the method runs, its usual course of action will be to acquire information from the user's CGI query, make appropriate calls into the TOME core to obtain or modify database information,

format that information to be used in a template, and then make the call to Template Toolkit with the appropriate information.

2.2.6 Properties

TOME::Interface will be a standard text file. It will run everything under Perl's taint mode. It will use strict. It will use warnings.

2.2.7 Creation, Existence, and Management

TOME::Interface will exist as a single instantiation created by admin.pl. The web server (usually Apache) will be responsible for calling admin.pl. Multiple instantiations of admin.pl (and therefore TOME::Interface and any other classes) can exist only if Apache runs multiple copies, which is expected.

2.2.8 Resource Usage/Management

No specific requirements exist for this component. Resources will be limited only by the web server.

2.2.9 State/Session/Context Management

This component has no particular knowledge of states, sessions, or contexts.

2.2.10 Data Storage

This component has no intrinsic need for its own data storage. It will interact heavily with the database, but only through the TOME module. It has no internal knowledge of anything storage-related.

2.2.11 Performance

This component does not have any specific performance or speed requirements. Faster is better, but that is determined entirely by the machine on which it is run. The only impact of faster or slower performance is the quality of the end user's experience.

2.2.12 Packaging

TOME::Interface is packaged as a single Perl module name Interface.pm in the TOME directory.

2.3 TOME

2.3.1 Role

The role of the TOME module is to provide database connectivity and all utility functions used by other parts of the program.

2.3.2 Responsibilities

TOME needs to validate all subs with the Params::Validate module. Even if the data has already been validated inside TOME::Interface, it needs to do it again. Sooner or later, someone will make a call with unvalidated data, and catching it is critical.

2.3.3 Exclusions

TOME shouldn't know anything about users and especially nothing about templates.

2.3.4 Collaborators

TOME collaborates very closely with TOME::Interface. TOME is the base class of TOME::Interface. TOME is in turn implemented as a subclass of CGI::Interface. TOME also interacts closely with several external modules such as Template Toolkit, DateTime, Params::Validate, SQL::Interpolate, and MIME::Lite. It also uses the TOME::TemplateCallbacks modules to make dynamic callbacks available to templates.

2.3.5 Key Scenarios

admin.pl loads TOME::Interface and calls the run method Since the run method is not actually a part of TOME::Interface, it has no direct responsibilities in this case. However, the run method sets in motion a chain of events controlled by CGI::Application including calling the setup and cgiapp_prerun methods, which are implemented by TOME::Interface.

A runmode dispatch occurs TOME::Interface contains all of the runmode subs that CGI::Application will dispatch to. A runmode dispatch will simply run one of these methods. When the method runs, its usual course of action will be to acquire information from the user's CGI query, make appropriate calls into the TOME core to obtain or modify database information, format that information to be used in a template, and then make the call to Template Toolkit with the appropriate information.

2.3.6 Properties

TOME::Interface will be a standard text file. It will run everything under Perl's taint mode. It will use strict. It will use warnings.

2.3.7 Creation, Existence, and Management

TOME::Interface will exist as a single instantiation created by admin.pl. The web server (usually Apache) will be responsible for calling admin.pl. Multiple instantiations of admin.pl (and therefore TOME::Interface and any other classes) can exist only if Apache runs multiple copies, which is expected.

2.3.8 Resource Usage/Management

No specific requirements exist for this component. Resources will be limited only by the web server.

2.3.9 State/Session/Context Management

This component has no particular knowledge of states, sessions, or contexts.

2.3.10 Data Storage

This component has no intrinsic need for its own data storage. It will interact heavily with the database, but only through the TOME module. It has no internal knowledge of anything storage-related.

2.3.11 Performance

This component does not have any specific performance or speed requirements. Faster is better, but that is determined entirely by the machine on

which it is run. The only impact of faster or slower performance is the quality of the end user's experience.

2.3.12 Packaging

TOME::Interface is packaged as a single Perl module name Interface.pm in the TOME directory.

2.4 TOME::TemplateCallbacks

2.5 Templates

2.6 Static Content