TOME Application Model

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

October 5, 2006

Contents

1 Introduction **4**
- 1.1 Purpose of Document . . . 4
- 1.2 Background . . . 4
- 1.3 References . . . 4

2 Application Overview **5**
- 2.1 Scope . . . 5
- 2.2 Context . . . 5
- 2.3 Technical Environment . . . 5

3 Actors **6**
- 3.1 Actor Diagram . . . 6
- 3.2 Actor Definitions . . . 6
  - 3.2.1 Patron . . . 6
  - 3.2.2 TOMEkeeper . . . 7

4 Business Use Cases **7**
- 4.1 Use Case Listing . . . 7
- 4.2 Graphical Use Case Diagram . . . 7
- 4.3 Business Use Cases . . . 7
  - 4.3.1 UC01: Book Added . . . 8
  - 4.3.2 UC02: Book Reserved . . . 9
  - 4.3.3 UC03: Book Checked Out . . . 10
  - 4.3.4 UC04: Class Added . . . 11
  - 4.3.5 UC05: Book Added To Class . . . 12

5 Business Domain Model **13**
- 5.1 Business Class Diagram . . . 13
- 5.2 Business Object Definitions . . . 14
  - 5.2.1 TOME . . . 14
  - 5.2.2 Database . . . 15
  - 5.2.3 Web Interface . . . 16
  - 5.2.4 Templates . . . 17

6 User Interface Requirements Specification **18**
- 6.1 User Interface . . . 18
- 6.2 Navigation Model . . . 18

6.3 Screens . . . 18
6.4 View Definitions . . . 19
6.4.1 Home Page . . . 19
6.4.2 Patron Page . . . 20
6.4.3 Modify Class . . . 20
6.4.4 Reserve Books . . . 20
6.4.5 Add Book . . . 21
6.4.6 Add Class . . . 21
6.4.7 Check Books Out . . . 21

7 Non-Functional Requirements Specification 22
7.1 Overview . . . 22
7.2 Enabling Technologies . . . 22
7.2.1 Target Hardware and Hardware Interfaces . . . 22
7.2.2 Target Development Environment . . . 22
7.2.3 System Interfaces . . . 22
7.3 Capacity Planning . . . 23
7.3.1 Traffic Volumes over Time . . . 23
7.3.2 User Populations and Locations . . . 23
7.3.3 Permanent Storage . . . 23
7.4 Printing . . . 23
7.5 Network . . . 23
7.6 Workstations . . . 23
7.7 Operational Parameters . . . 24
7.7.1 Usability . . . 24
7.7.2 Reliability . . . 24
7.7.3 Maintainability . . . 24
7.7.4 Portability . . . 24

8 Design Approach 25
8.1 Key Drivers . . . 25
8.2 Issues . . . 25
8.3 Solution Approach . . . 25

9 Design 25
9.1 Overview . . . 25
9.2 Assumptions . . . 25
9.3 Component Diagram . . . 26

9.4 Component Descriptions 26
9.4.1 TOME 26
9.4.2 TOME::Interface 27
9.4.3 TOME::TemplateCallbacks 27

10 Interfaces **27**
10.1 Offered Services 27
10.2 Internal Support Services 27
10.3 External Services Required 27
10.4 External Libraries Required 28

11 Test Strategy **29**

12 Database Structure **29**

List of Figures

1 Actor Diagram 6
2 Use Case Diagram 8
3 Business Class Diagram 14
4 TOME Sequence Diagram 15
5 Database Sequence Diagram 16
6 Web Interface Sequence Diagram 17
7 Template Sequence Diagram 18
8 Template Sequence Diagram 19
9 Component Diagram 26
10 Database Diagram 30

1 Introduction

1.1 Purpose of Document

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

An application is defined as having business logic and presentation elements while an individual component is defined as providing a particular set of services or presentation—but not both. This approach allows for the structure of a modern component based distributed system.

Delivery scheduling and testing issues are also dealt with to ensure that other related applications and components can be developed in parallel.

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 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`

2 Application Overview

2.1 Scope

The TOME system is responsible for managing:

- Books
- Book information
- Class information
- Class-to-book mappings
- Book reservations
- Book checkouts

The TOME system is **not** responsible for managing:

- Book acquisition
- Book disposal
- Selling old books
- Any financial activity

2.2 Context

TOME exists as an independent system with no ties to other databases. Information from other databases, such as class listings, book information, and book-to-class associations will be used, but not in an automated fashion.

2.3 Technical Environment

TOME is a web-based Perl application. It is intended to be run under the Apache webserver, but any webserver that supports CGI should be capable of running the system. Template::Toolkit is used to process HTML templates. PostgreSQL is used as the database backend. CGI::Application is used as the framework for the system as a whole.

3 Actors

3.1 Actor Diagram

Figure 1 shows an overview of the actors involved in the TOME system and how they interact with each other. Patrons talk to TOMEkeepers who use the TOME web interface which is based on the TOME database.

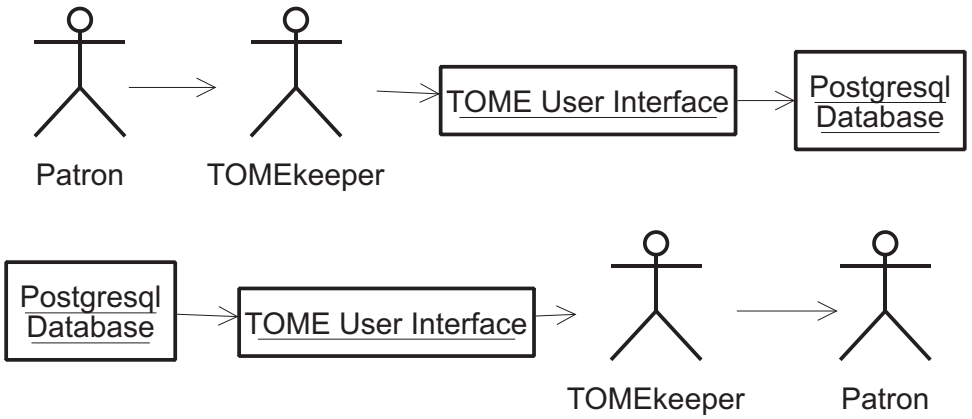

Figure 1: Actor Diagram

3.2 Actor Definitions

3.2.1 Patron

Description	The Patron is any student who wishes to use the TOME system. Only students that reside on floors with an active TOME system can become Patrons. Patrons never interact directly with TOME, only through TOMEkeepers.
Aliases	Student.
Inherits	None.
Actor Type	Passive - Person.

3.2.2 TOMEkeeper

Description	The TOMEkeeper is the primary user of the TOME system. They are responsible for all interactions with Patrons, all system administration, and all TOME activity.
Aliases	None.
Inherits	None.
Actor Type	Active - Person.

4 Business Use Cases

4.1 Use Case Listing

ID	Use Case Name	Comments
UC01	Book Added	When a book is donated to the TOME system
UC02	Book Reserved	When a patron requests that book be reserved for a particular semester
UC03	Book Checked Out	When a patron checks out a physical book
UC04	Class Added	When a TOMEkeeper adds a class to the system
UC05	Book Added To Class	When a TOMEkeeper adds a book to a particular class

4.2 Graphical Use Case Diagram

This section represents the business use cases of the TOME system in graphical form in Figure 2.

4.3 Business Use Cases

This section documents the complete business scenarios within the scope of this project.

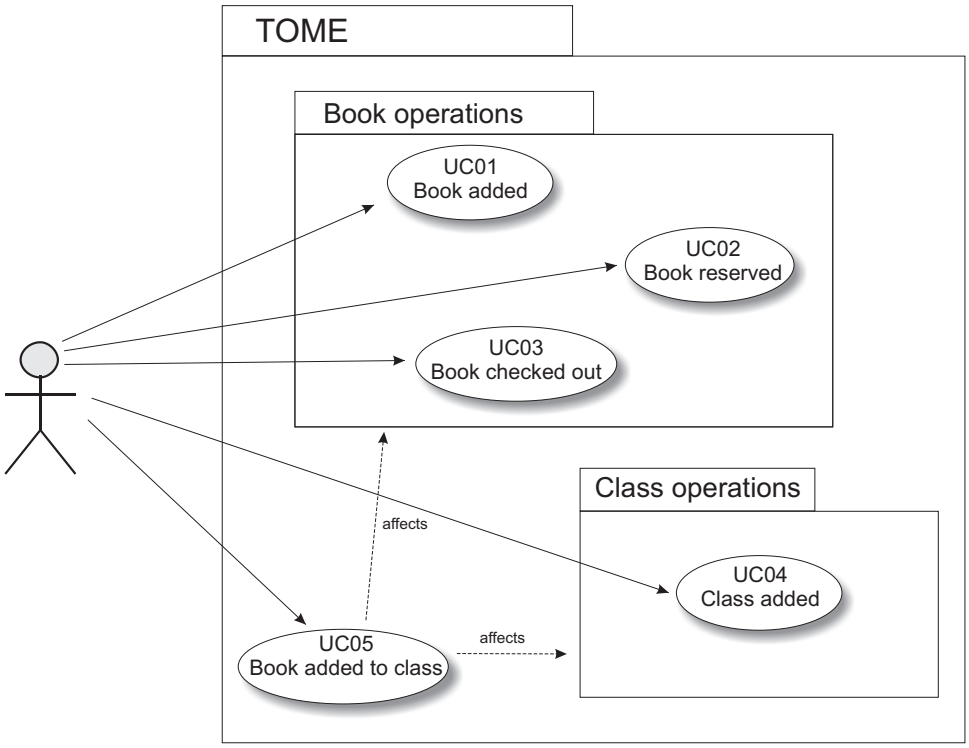

Figure 2: Use Case Diagram

4.3.1 UC01: Book Added

Description This scenario happens when a patron donates a book to the TOME on their floor. The book information must be entered in the system, and the book must be assigned a unique ID number by the system.

Actors TOMEkeeper and (indirectly) Patron

Preconditions

1. The TOMEkeeper must have an account
2. The TOMEkeeper must be logged in

Use Case Text

1. Patron requests to donate a book
2. TOMEkeeper enters the book information into the system
3. TOME assigns a book ID and presents the book information page

Alternate Courses None.

Extends None.

User Interfaces ID0005.

Constraints None.

Questions None.

Notes None.

Source Documents None.

4.3.2 UC02: Book Reserved

Description This scenario happens when a patron requests that a book be reserved for them for a particular semester.

Actors TOMEkeeper and (indirectly) Patron

Preconditions

1. The TOMEkeeper must have an account
2. The TOMEkeeper must be logged in

Use Case Text

1. Patron sends their class list to the TOMEkeeper
2. TOMEkeeper enters the class information into the system
3. TOME determines what book reservations can be made, makes them, and notifies the patron via email

Alternate Courses None.

Extends None.

User Interfaces ID0004.

Constraints None.

Questions None.

Notes None.

Source Documents None.

4.3.3 UC03: Book Checked Out

Description This scenario happens when a patron requests that a book be checked out for a semester

Actors TOMEkeeper and (indirectly) Patron

Preconditions

1. The TOMEkeeper must have an account
2. The TOMEkeeper must be logged in

Use Case Text

1. Patron requests that the book be checked out. This may have already been done in the form of a reservation, or the patron can make the request without a previous reservation
2. The TOMEkeeper checks a specific instance of the book out to the patron, and the system logs that checkout

Alternate Courses None.

Extends None.

User Interfaces ID0007.

Constraints None.

Questions None.

Notes None.

Source Documents None.

4.3.4 UC04: Class Added

Description This scenario happens when a TOMEkeeper adds a class to the system

Actors TOMEkeeper

Preconditions

1. The TOMEkeeper must have an account
2. The TOMEkeeper must be logged in

Use Case Text

1. The TOMEkeeper finds a class that needs to be added to the system
2. The TOMEkeeper adds the class to the system

Alternate Courses None.

Extends None.

User Interfaces ID0006.

Constraints None.

Questions None.

Notes None.

Source Documents None.

4.3.5 UC05: Book Added To Class

Description This scenario happens when a TOMEkeeper adds a book to a class that is already listed in the system.

Actors TOMEkeeper

Preconditions

1. The TOMEkeeper must have an account
2. The TOMEkeeper must be logged in
3. The class to be added to must already exist within the system

Use Case Text

1. The TOMEkeeper finds a book that needs to be added to a particular class
2. The TOMEkeeper navigates to the class information page
3. The TOMEkeeper adds the book to the class

Alternate Courses None.

Extends None.

User Interfaces ID0003.

Constraints None.

Questions None.

Notes None.

Source Documents None.

5 Business Domain Model

The business domain model provides a listing of the modules contained in the project and how they work together. This framework gives the project team a high-level guide to use in developing the project.

5.1 Business Class Diagram

Figure 3 shows a graphical representation of the business classes involved in the TOME system.

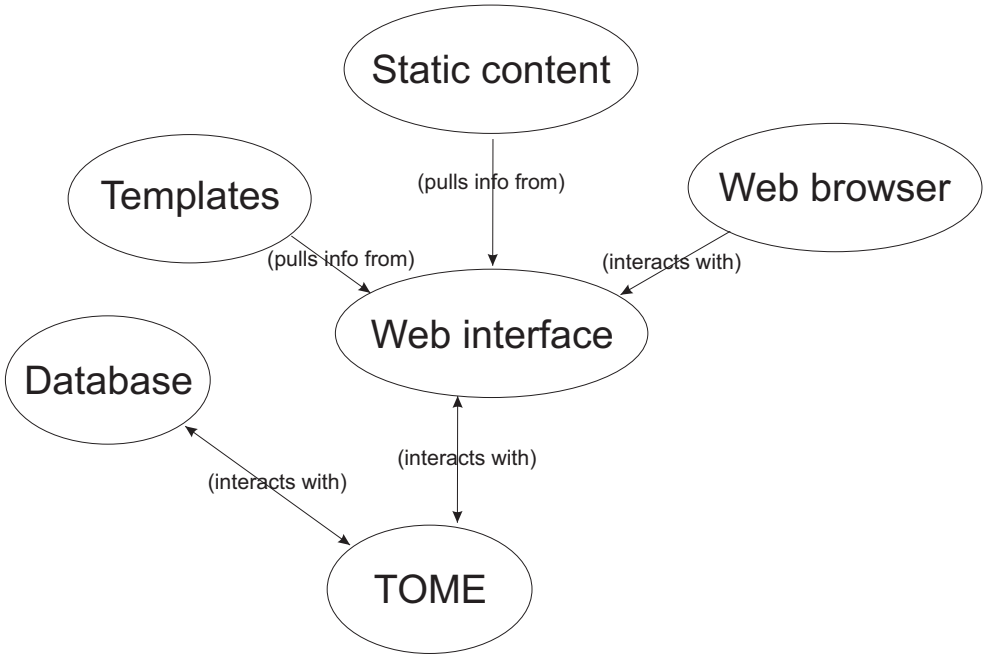

Figure 3: Business Class Diagram

5.2 Business Object Definitions

5.2.1 TOME

Description	TOME is the core module that processes database requests from the Web Interface.
Attributes	TOME contains all the methods used for interacting with the database as well as processing template files.
Responsibilities	TOME is responsible for all database communication, all template processing, all error handling, and any other "core" functionality

Business Rules	TOME has dependencies on many Perl modules (listed in the install document that can be located by referencing Section 1.3). These must be installed before TOME will function properly.

Figure 4 is an example sequence of events involving TOME.

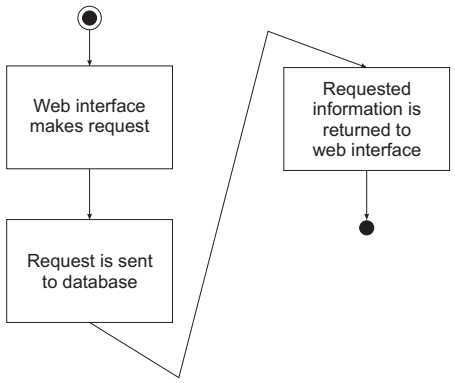

Figure 4: TOME Sequence Diagram

5.2.2 Database

Description	The database is where all the information for TOME is stored. The current implementation uses PostgreSQL for the database and DBI for all database interactions.
Attributes	All TOME data is stored in the database. See Section 12 for more details. TOME communicates with the database to retrieve, insert, and update all data for the system.
Responsibilities	The database is responsible for maintaining a consistent state for all data at all times.
Business Rules	PostgreSQL must be installed before the system will be functional.

Figure 5 is an example sequence of events involving the database.

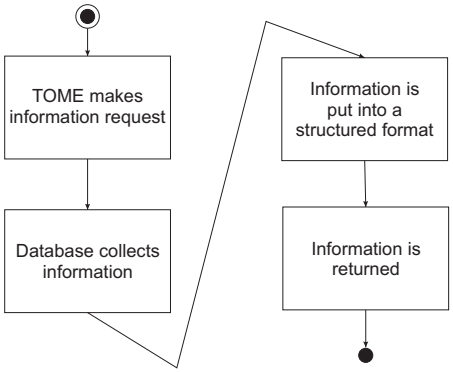

Figure 5: Database Sequence Diagram

5.2.3 Web Interface

Description	The Web Interface forms the connection between the Database and the Templates. It processes information that TOME retrieves from the Database and sends it to the templates. It also processes user requests and sends them to TOME.
Attributes	The Web Interface contains all methods related to interacting with the user via CGI.
Responsibilities	The Web Interface is responsible for taking any data the user submits, verifying its validity, and submitting it to TOME to be run through the Database. It is also responsible for taking information that TOME retrieved from the Database and preparing it for the templates to use.
Business Rules	The Web Interface depends on a number of Perl modules (see the Install document referenced in Section 1.3) and these must be installed before it will be functional.

Figure 6 is an example sequence of events involving the Web Interface.

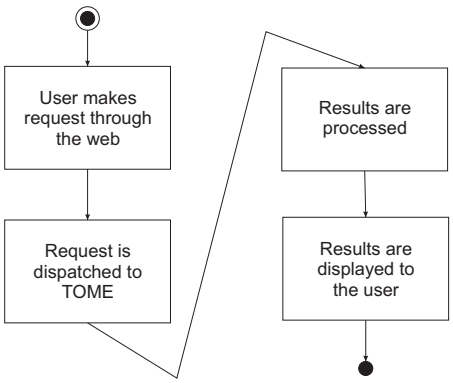

Figure 6: Web Interface Sequence Diagram

5.2.4 Templates

Description	Templates form the basis for the look and feel of the web interface. They do not give any functionality in and of themselves, but are a way to display information that is given by the Web Interface
Attributes	Templates have no methods, they only contain the data that determines how information is formatted and displayed.
Responsibilities	The templates must take information that is given to them by the Web Interface and format it. In the process of formatting that information, they may request more information through a callback mechanism.
Business Rules	The templates need to be XHTML and CSS compliant.

Figure 7 is an example sequence of events involving template generation.

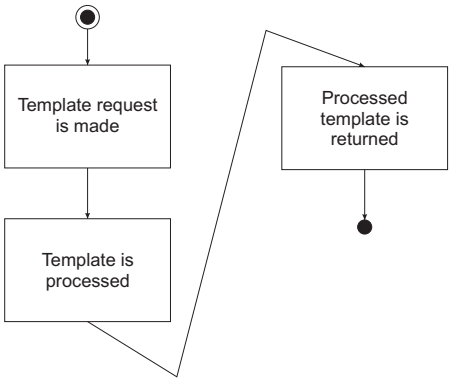

Figure 7: Template Sequence Diagram

6 User Interface Requirements Specification

6.1 User Interface

The interface is entirely web-based. Interaction with the Web Interface, which is displayed through templates accesses the TOME system which interacts with the Database.

6.2 Navigation Model

Figure 8 graphically displays how the user interacts with TOME through the Web Interface.

6.3 Screens

Screen ID	**Description**
ID0001	Home Page
ID0002	Patron Page
ID0003	Modify Class
ID0004	Reserve Books
ID0005	Add Book
ID0006	Add Class
ID0007	Check Books Out

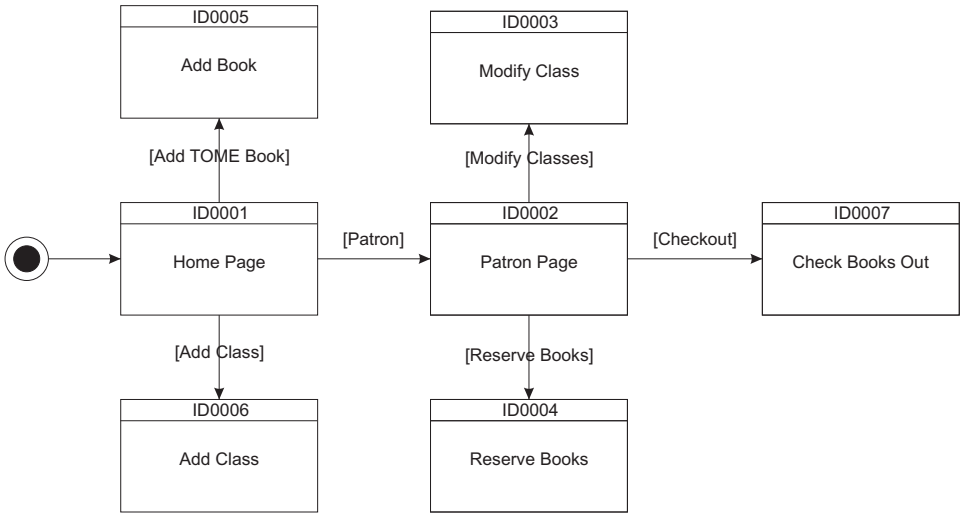

Figure 8: Template Sequence Diagram

6.4 View Definitions

6.4.1 Home Page

View Title	TOME
Number	ID0001
Description	This is the main home page for TOME and contains links to all major functions. It also displays basic information about the status of the system such as the current semester and the current user that is logged in.
Scenario References	None.

6.4.2 Patron Page

View Title	Patron Page
Number	ID0002
Description	This page displays information about the currently selected patron such as books currently checked out, books reserved, and classes for the current semester. It serves as a starting point for many of the patron-related tasks.
Scenario References	None.

6.4.3 Modify Class

View Title	Class Information Page
Number	ID0003
Description	From the class information page, information can be modified about a class. Comments can be made, books for the class can be added, deleted, and modified, and other maintenance work can be done.
Scenario References	UC05.

6.4.4 Reserve Books

View Title	Reserve Books
Number	ID0004
Description	From the patron information page, classes can be added for that patron. Once a class information table is built up, the TOMEkeeper can start to reserve books for that semester for the patron. This process will happen in steps where information about a class and its book associations is presented to the TOMEkeeper, and the TOMEkeeper makes the appropriate selections. After the selections are made, the books are reserved and the patron is emailed a notification of what all happened.
Scenario References	UC02.

6.4.5 Add Book

View Title	Add TOME Book
Number	ID0005
Description	When a patron desires to add a book, this is the interface that will be shown. From this page, the TOMEkeeper will enter basic information about the book such as the ISBN and which patron donated the book. If TOME is unaware of the ISBN, the TOMEkeeper will be presented with a form to fill out the rest of the information about the book.
Scenario References	UC01.

6.4.6 Add Class

View Title	Add Class
Number	ID0006
Description	If a TOMEkeeper desires to add a class to the system, they will use this interface. Adding a class is as simple as entering in the class ID and a name for the class. All class information is added through the interface defined in Section 6.4.3.
Scenario References	UC04.

6.4.7 Check Books Out

View Title	Check Book Out
Number	ID0007
Description	When it comes time to actually check a physical instance of a book out, this is the interface that will be used. The TOMEkeeper can either change an existing reservation into a checkout, or they can specify the exact book to be checked out. After this is accomplished, the Patron can take the book for the semester.

Scenario References	UC03.

7 Non-Functional Requirements Specification

The Business Scenario Model only represents part of the full set of requirements. Although the Business Scenario Model represents the functional requirements that are included in the project, there are many intangible requirements that havent been covered. The purpose of this section is to document the non-functional requirements of TOME.

7.1 Overview

The non-functional requirements outlined in this section include the technologies needed to allow TOME to run, capacity planning strategies, and operational parameters.

7.2 Enabling Technologies

7.2.1 Target Hardware and Hardware Interfaces

TOME is a reasonably lightweight application. It is targeted to run on any server-class computer. If the computer is fast enough to run Apache, PostgreSQL, and Perl, it is fast enough for TOME. Of course, the more users that access the system, the faster and bigger the server will have to be.

7.2.2 Target Development Environment

Development machines will utilize Vim, PostgreSQL, Subversion, and Perl on a Linux platform.

7.2.3 System Interfaces

TOME is a very high-level application, and as such will not have any direct interfaces to the system itself. Apache will handle all network communication with the user, and PostgreSQL will handle all data storage.

7.3 Capacity Planning

7.3.1 Traffic Volumes over Time

Because TOME is a web application, its traffic is very "bursty." TOME will have a high draw on system resources during the time a request is handled, but after the request has been handled there is no load on the system. So, TOME would go from a 0 when it isn't handling a request to a 1 when it is handling a request.

7.3.2 User Populations and Locations

TOME is intended to be used by TOMEkeepers. They will primarily access the system using their personal computers on the floors they live. However, during Christmas and Summer break, the TOMEkeepers will need to be able to access the system remotely.

7.3.3 Permanent Storage

The data that will be held is actually very small in volume. It is all little chunks of text stored in a database. A very large TOME installation will be on the order of tens of megabytes.

7.4 Printing

Any printing requirements will be handled by the user's web browser. TOME has no intrinsic printing mechanisms.

7.5 Network

TOME has extensive network requirements, however they will be handled entirely by Apache and the host operating system. TOME itself has no knowledge of network layout or transportation.

7.6 Workstations

Any workstation with a modern, standards-compliant web browser can be used to access TOME.

7.7 Operational Parameters

7.7.1 Usability

In order to be successful, TOME must be very user-friendly. Its target audience is limited, so it doesn't need to be completely intuitive, but the TOMEkeeper's technical skills will range from the extremely proficient to the beginner. To improve usability, the interface will be designed to make most operations as obvious as possible. Care has been taken to make the workflows logical. Also, training will be given to every new user.

7.7.2 Reliability

TOME must be extremely reliable. The database is especially important. Without the data, the system becomes useless, and without the system, hundreds of books could be misplaced. In order to ensure the reliability of the database, extensive checks have been put in place to make sure the data is consistent at all times (see Section 12 for details).

Nightly backups will be made of all the data in the system. Restoring a backup after a catastrophic failure is as simple as reinstalling the system, and using the PostgreSQL data recovery tool to reload the database.

7.7.3 Maintainability

Maintainability is a large concern with TOME. To improve maintainability, all code will be documented in both embedded POD and in overviews such as this one. Development will always be done in a Subversion environment to ensure traceability of all changes.

7.7.4 Portability

TOME itself contains no platform-dependent code. The install scripts do make assumptions about a Linux-based platform, but the install could be done manually on other platforms. The bulk of the code is written to interface to other extremely portable libraries and systems such as PostgreSQL and CGI.

8 Design Approach

8.1 Key Drivers

Continued development of TOME will follow the basic pattern that already exists. In some ways this is good because it will not need to be redesigned. However, in some ways it can be a hindrance because the original design does not use objects as extensively as it could. Even though it is less than ideal, it was determined that the benefit that would be gained by a complete redesign would not outweigh the cost in time lost on new development.

8.2 Issues

The primary issue was determining how to preserve orthogonality without changing too much of the existing design. One of the best ideas was a way to separate template and interface interaction. This is detailed in Section 9.4.3.

8.3 Solution Approach

The solution that was chosen is to retain as much as possible while still aiming for orthogonality and development speed. The system is broken into different modules that each have a well-defined problem domain and provide cohesion while minimizing coupling.

9 Design

9.1 Overview

TOME is subdivided into several components to promote orthogonality in design. The three main components (or "modules") are TOME, TOME::Interface, and TOME::TemplateCallbacks.

9.2 Assumptions

None.

9.3 Component Diagram

Figure 9 graphically depicts the component layout of the TOME system.

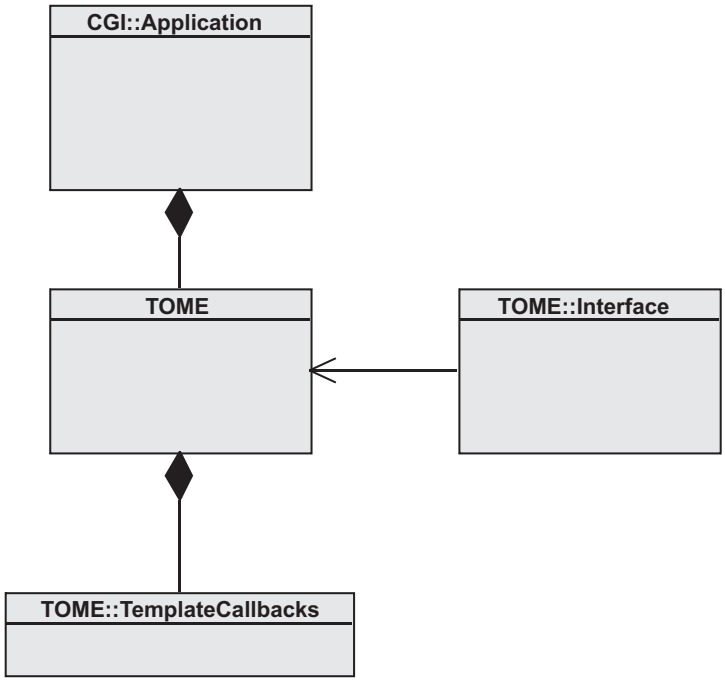

Figure 9: Component Diagram

9.4 Component Descriptions

9.4.1 TOME

The purpose of this module is to provide database connectivity and all utility functions used by other parts of the program. Ideally, the TOME module shouldn't know anything about users and especially nothing about templates.

9.4.2 TOME::Interface

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

9.4.3 TOME::TemplateCallbacks

Implements the TOME::TemplateCallbacks module. All templates are given a "tome" object that is an instantiation of this class. The idea is that when the template is given ID numbers of various things in the database (patrons, books, tomebooks, etc.) by TOME::Interface, the template can use this object to query the database through the TOME module and get string representations of the data. This module has the kinda funny place of knowing a little bit about both the template and the database. Most subs will probably just be wrappers of TOME subs, but if there is any additional data manipulation that needs to take place before the template (without crossing the line of actually doing template work), this is the place to do it. The reason this is separate from TOME is to give an extra layer of abstraction between the actual database calls and the template. Giving direct database access from inside a template is a thought too horrible to contemplate.

10 Interfaces

10.1 Offered Services

None.

10.2 Internal Support Services

Internal support services will be provided by TOME and TOME::TemplateCallbacks.

10.3 External Services Required

The services defined by the external libraries will be required.

10.4 External Libraries Required

In addition to Perl, Apache, PostgreSQL, and the Perl bindings for those technologies, the following Perl modules (and their dependencies) are required:

- Template
- Template::Plugin::Comma
- Template::Plugin::CGI
- CGI::Application
- CGI::Application::Plugin::Session
- CGI::Application::Plugin::DBH
- CGI::Application::Plugin::HTMLPrototype
- CGI::Application::Plugin::ValidateRM (needs Compress::Zlib, even though CPAN doesn't show it as a prerequisite)
- CGI::Application::Plugin::Forward
- DateTime
- DateTime::Format::Pg
- SQL::Interpolate
- MIME::Lite
- Crypt::PasswdMD5
- DBD::Pg

11 Test Strategy

A framework will be created that is capable of running automated tests on all parts of the TOME system to ensure functionality and to prevent regression. Tests will be broken up into the following categories: Database, Core, and Web Interface. Tests will be made for all levels of functionality, starting from the database level, to check the consistency mechanisms and triggers, all the way up to simulating interaction with a user and a web browser.

Once completed, these tests will be run on a nightly basis. They will be capable of creating a clean database and simulating a full range of activities. All developers will be required to run and pass all tests before checking any new code in.

12 Database Structure

The overall structure of the database can be seen in Figure 10. In addition to the structure, there are also triggers that ensure the consistency of the database at all times.

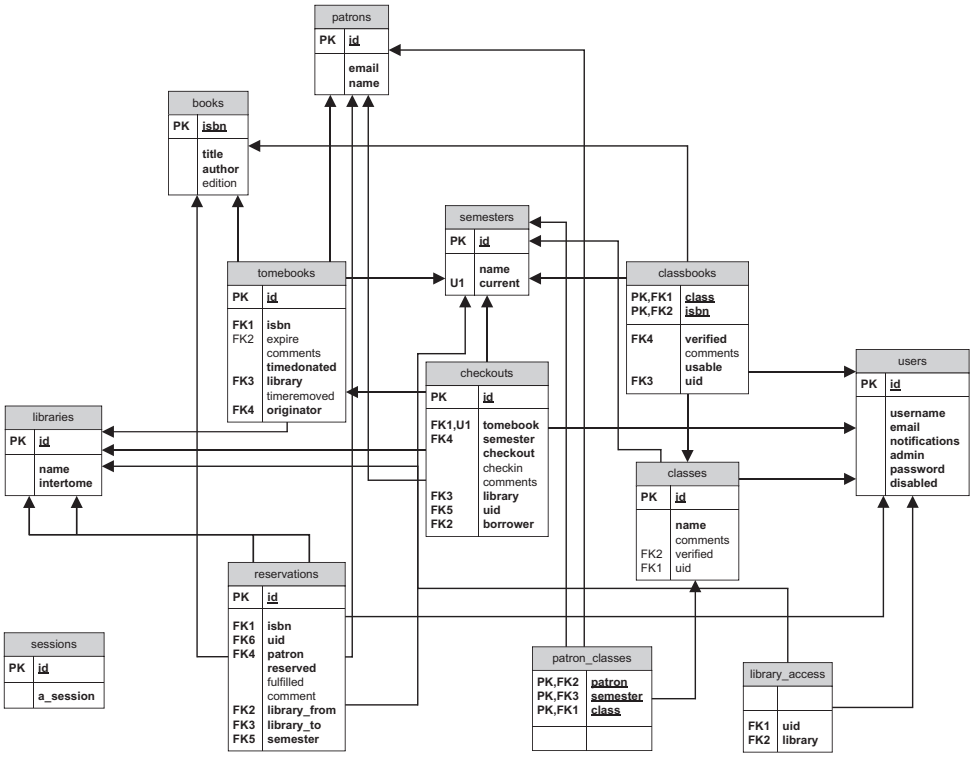

Figure 10: Database Diagram