Reflections

In this exercise, the architecture of an existing application was reconstructed from the source code, the deployment descriptors, and the existing documentation. An incomplete software architecture documentation was created using UML 2.0 and informal notation. Below are observations regarding the Java Pet Store implementation, the UML 2.0 notation and the wiki as a tool to capture architecture information.

Contents

- Comments on Java Pet Store Artifacts
  - About the Existing Documentation
  - Issues in the Implementation
- Reflections on Using Wiki for Architecture Documentation
  - Benefits of Wiki for Architecture Documentation
  - Weaknesses of Wiki for Architecture Documentation
- Comments on Architecture Documentation, Notations, etc.
  - Benefits of an Architecture Documentation Template
  - UML 2.0 Notation
- Other Comments

Comments on Java Pet Store Artifacts

About the Existing Documentation

There are two public documents published by the authors of the application: the Sample Application Design and Implementation and Chapter 11, "Architecture of the Sample Application," of the Designing Enterprise Applications with J2EE book. The books provide the documentation of the architecture from the manufacturer's point of view. It is shows in fair detail the beans and servlets used in the application. They show in detail the EJBs and servlets used in the application. The documents also include sequence diagrams that explain some of the main functionalities of the system.

Nevertheless, from the perspective of an implementor the document is incomplete and inconsistent. Some of the diagrams have ambiguous notation, and there is no clear direction on what the static (module) and data views of the system are.

Issues in the Implementation

Several issues were found in the implementation during the architecture reconstruction effort:

- One interface contains a nested class and the nested class implements the enclosed interface. Class DefaultXDE is a public member of `com::sun::j2ee::blueprints::xmldocuments::XMLDocumentEditor` and at the same time extends that same interface. There is no documentation in the code or in the architecture that specifies why this is done. This class is used and extended by other classes in `<app>` opc and `supplier`. This looks like a code smell.
- There are several classes with wrong comments, such as,
  - `waf::controller::web::DefaultWebController`
  - `waf::controller::web::WebController`
  - `waf::controller::web::action::CreateUserHTMLAction`
  - `waf::controller::web::PetstoreComponentManager`, and
  - `controller::events::OrderEvent`, `CustomerEvent`, `CreateUserEvent`, `SignOnEvent`.
- There are many classes missing their class comments, especially in the components group.
- The tables in petstoredb are missing foreign keys that would be necessary to ensure referential integrity of the data. For example:
  - category in ProfileEJBTable
  - The usernames in CustomerEJBTable are not restricted to the users with password in the UserEJBTable.
  - The favorite category in the ProfileEJBTable is not restricted to the available categories in the CATEGORY table.
  - The banner preference (ProfileEJBTable) and the status (AccountEJBTable) are not restricted to any values... they can take any value... and this is not appropriate.
- There are 12 classes with the same behavior and method names, but no interface or class hierarchy to ensure consistency and help implementation of commonality. The tools::populate package contains 12 Populator classes. All of them have a very similar behavior and have the same class method names, but strangely enough there is not a common interface that all classes implement. The same code is repeated over and over. Inheritance would be a natural solution, but apparently the authors of the code did not consider this possibility before jumping to code. This code is difficult to maintain and error-prone.

Reflections on Using Wiki for Architecture Documentation

The architecture diagrams of the PetStore application were created using Microsoft Visio. To help distribution of effort and facilitate configuration management, the diagrams corresponding to each view were kept in a separate Visio file. The reflections on the use of wiki are restricted to the product that was used to develop the documentation: [www.mediawiki.org/ MediaWiki] version 1.4.

Benefits of Wiki for Architecture Documentation

- Transclusion. This feature of the wiki allows writing a text once and embedding it in different pages. For the reader, it seems like the text was created in that page. If the text is modified all pages that have it will be updated. Transclusion is the solution for a common problem in technical documentation: repetition of information. Very often pieces of information are needed in multiple places in the document. The common solution is either (a) add a link or pointer to a single location (e.g., Acâ€“?~ÁSee Section XAcâ€“?~Á) or (b) copy the information, creating the duplication. The problem of the first alternative is that it makes the document less readable, especially when printed. The problem of the second alternative is that
duplication makes the document more difficult to change. With transclusion, the embedded information is flawlessly integrated to the pages. The reader cannot notice there is an embedded text or image in the page. Yet, there is only one place to edit the information.

- Public comments. Any reader can comment on a page.
- Change notification. Users can subscribe to a page so whenever a change happens they will receive an email.
- Version control. The wiki keeps tracks of changes, can show differences between versions, and bring back an old version.
- Easy modification. Modifications in the wiki are easy, there is no additional tool to be installed; the text can be modified within the browser. Rich text features (links, bold, underline, etc.) are easy to work with thanks to the editing toolbar and the “Show Preview” feature. Having sample text is very helpful.
- Changes the paradigm from a centralized configuration control board to a distributed model. [Bachmann_2005]
- Pages can show who refers to them. So it is simpler to understand relations among views.
- Architecture views can be connected using links. For the reader, moving from one view to another is just one click away.

**Weaknesses of Wiki for Architecture Documentation**

- Online access and a screen appropriate for reading is needed.
- Many readers prefer the printed format (with formatted page breaks, margins, page numbers and other formatting options) over the www screen format.
- Server reliability and security issues matter.
- There is not a standard Wiki format/syntax. Porting a wiki from one Wiki server product to another may require rewriting most of the pages contents.
- Sophisticated formatting (i.e., text of different colors, variety of fonts, image layout) and printing setup are complicated or not possible.
- Different privilege levels cannot be granted on a document or category basis. Ideally, there should be a way to restrict the access to a given page to specific groups of users at different capacities (read only, provide comments only, read/write, etc.) WikiMedia does not support a rich access management policy.
- Document version control for binary documents. Binary documents such as images and worksheets can be uploaded to wiki, but the version control system is precarious, a person can erase the work of others easily.
- Comments page is not visible from the main page.
- Comments cannot be tagged for a particular version of the page. How to manage comments that relate to an old version of the page? MediaWiki does not provide a solution. A feature could be to timestamp the comments to document versions and allow the writers to tag the comments (i.e., open, closed, overcome by events, etc.)
- There is no way to print the whole wiki in a single step.
- There is no way to tag a version of the entire wiki. This could simplify going back to a previous baseline. If such option existed, there should also be a way to easily bring old versions and compare them.
- To personalize the wiki sysadmin privileges and direct access to PHP files (and understanding of the PHP programmin language) is needed. This is a big drawback and acts against the very idea of having a wiki.
- Table of contents are not easy to handle in WikiMedia. Numbering is fixed, there is no way to change it or to keep it consistent across pages.
- Having a unique table of contents automatically created for the project would be ideal.
- There is no automatic temporal save. If the page is closed the information is lost for good. It would be desirable to have this functionality, in particular, for blank pages.
- Import/Export. Exporting a Wiki is not simple. It has restrictions as it will not export history of changes or binary files. This is a major drawback specially if a project lasts several months and infrastructure is damaged or changed within that period of time.
- Base wikis. WikiMedia does not allow prebuilt sites, a functionality similar to that of templates in word processors is highly desirable as it would allow easy creation of SAD templates and their instantiation.

**Comments on Architecture Documentation, Notations, Etc.**

**Benefits of an Architecture Documentation Template**

By having a structured architecture document that follows a template, documenting is a matter of filling the sections prescribed by the template. It is easy to gauge how much of the document is left to be done. The “TBD” abbreviations was used consistently through the document to let readers (and authors) know what parts where missing.

**UML 2.0 Notation**

TBD

**Other Comments**

In a manual architecture reconstruction, some mistakes in diagram depictions can be detected when creating descriptions for the element catalog. This happened several times in the reconstruction of this system architecture. The element catalog definition served as a validation technique to the diagrams that had been issued.

Back to the Pet Store SAD main page