TECHNICAL REPORT OF APPLICATION AND PATTERNS USED

BY UDOISANG BLESSING SUNDAY

7588349

ADVANCED COMPUTING AND I.T. MANAGEMENT

INTRODUCTION

An Issue Tracking System (ITS) is a software application used in an organization to record and track the progress of every problem from when it is identified until it is resolved; for example, logging customer complaints/requests in an organization’s customer support call centre. Additionally, an ITS usually contains a “knowledge base”. A knowledge base is a centrally managed repository of articles containing diverse kinds of information such as resolutions to common problems, customer information, etc. A ticket in an ITS is a record of a single issue containing various information about the issue.

DESCRIPTION AND SCOPE OF THE APPLICATION

Description coming soon based on space remaining out of 5 pages.

I focused on managing the “knowledge base” of the issue tracking system. This includes displaying knowledge base articles, adding and editing articles and converting resolved tickets to knowledge base articles. Each article is classified under a category. Categories can have sub categories containing articles also. The application used a local XML data store for persistence.

TECHNICAL DETAILS OF PATTERNS USED

The prototype application made use of the following listed design patterns, which are clearly described with diagrams.

* The Bridge pattern
* The Composite pattern
* The Observer pattern
* The Singleton pattern
* The Controller pattern

THE BRIDGE PATTERN

The bridge pattern was used to abstract the persistence layer of the application.

ARTEFACTS DELIVERED

Technical Report: this report is one of the artefact produced.

Working Prototype Application: A prototype application has also been produced. See screen shots below. The statistics of the working prototype is as follows: SLOC, etc

SCREEN SHOTS OF THE WORKING PROTOTYPE

REFLECTION & AREAS FOR FURTHER IMPROVEMENT

<http://www.javaworld.com/javaworld/javaqa/2001-05/04-qa-0525-observer.html>

The hammer trap? Use it for reflection

CONCLUSION

OBSERVERs Vs OBSERVABLEs:

<http://www.javaworld.com/javaworld/javaqa/2001-05/04-qa-0525-observer.html?page=2>

\*Borrow inspiration of images from here: <http://developerlife.com/tutorials/?p=28>

\*I Noticed that the XML storage structure in itself represents a composite tree with each node containing a node. Can I use this info?

\*See SAX Tutorial for how to describe your classes and inspiration too. : <http://developerlife.com/tutorials/?p=29>

<http://developerlife.com/tutorials/?p=26> [Very Helpful too]

\*A simple Java.util.List could be enought to handle these things, let’s see sha

Hello

SAX: Simple API for XML

DOM: Document Object Model

<http://www.devarticles.com/c/a/XML/Java-and-XML-Basics-3/8/>

Some crazy data to support sax as better to use here than dom!

The three steps to using SAX in your programs are:

* Creating a custom object model (like Person and AddressBook classes)
* Creating a SAX parser
* Creating a DocumentHandler (to turn your XML document into instances of your custom object model).

SINGLE CLASS COMPOSITE

<http://wirfs-brock.com/blog/tag/pattern-repositories/>

“A single concrete class that could support either leaf or composite behaviors. Now that’s a thought…but is it still recognizable as a composite pattern? Sure...” [@REF Wirfs-Brock Associates (2006)]

<http://www.osebboy.com/blog/composite-pattern-with-parent-reference/>

Corroborates the idea of a Single Composite - Leaf[@REF, Omercan Sebboy, 2010]

FURTHER AREAS OF IMPROVEMENT

Create an index for the articles model so that when reading / listing them by category, one doesn’t have to iterate the whole collection, aggregating them, I can just iterate the index and get from the map the specified articles. Iterator Pattern could also be used here to create Iterator-By-CategoryID

Use caching and Lazy Materialization via Virtual Proxy to reduce (1) the number of times we have to return to the model to ask for articles listing by category and (2) reduce the memory foot print of the listed articles, only load full article when details needs to be viewed, otherwise, load a lightweight proxy in the JList component.