

Elastic JavaDoc

Kasper B. Graversen, kbilsted@it-c.dk
The IT University of Copenhagen
May 2001, Copenhagen, Denmark

ABSTRACT

We present a poster of the Elastic JavaDoc (EJD), a documentation presentation system. The system is based on the JavaDoc idea about keeping documentation in the source code (the principle of high proximity), and structuring this documentation in named chunks [2]. The system is not restricted to Java, any Object-Oriented language can be used, however the preliminary implementation works only with Java code.

The system yields progress in four areas

- The use of an underlying database for new and dynamic views on the documentation.
- The use of many different tags for identifying various chunks of documentation.
- The use of JSP (JavaServer Pages) as presentation layer — the layout is highly configurable and enables the filtering of information in an easy yet advanced way.
- Bridge the documentation to process oriented aspects of software engineering.

KEYWORDS: Presentation system, structured documentation, custom mark-up tags, JavaServer Pages, proximity documentation.

INTRODUCTION

The background for EJD sets foot not in the area of documentation but in the hypertext world. It was the article “Structures Web Site Design” [6] that gave rise to the idea. The article described an approach to building hypermedia applications called “OOHDM” (the case in the article was a site on art named “Portinari Project”). The concept of the OOHDM is to structure underlying information in a OO manner, completely separating information from layout and navigational features. With structured data they were able to reuse information in different situations, i.e. information on a painting (name, year, painting style) could be used in presenting the painting, the same information could just as well be used in building a “painters time line”. The very same information

could create more than just named nodes on the time line, it could be used to present a time line of paintings in a specific painting style. General information on art through time could easily be combined, and the user could easily combine “how art changed in time” with how the painter “changed in time” in order to find parallels (or maybe the opposite which could be just as interesting). Since the pages were dynamically generated no special attention was needed on i.e. the time line when adding a new painting to the application, since the time line was generated by requesting all paintings by the author sorted in various styles depending on the type of time line.

The concept gave rise to the idea that documentation could utilize these features as well. Our presumption was that since programming languages tend to be highly structured and cleansed of ambiguity, an implementation had the potential of becoming “beautiful” (in technical terms).

In the field of improving navigation of Java documentation limited work has been conducted, or at least been contributed to the public. The “DJavadoc” by Berglund is a step in the direction the EJD takes.

STRUCTURE

Imposing structure on the documentation creates navigation facilities for the reader. The structure can easily be understood by a program (i.e. the EJD) which can then present the documentation according to the readers wishes. As Berglund points out in [1] the Javadoc contains excessive information in all work situations, thereby blurring the users view.

The concept of the EJD is utilizing a combination of the structure of the programming language and the high degree of structuring of the documentation. Through this combination, useful information can be generated automatically. A good example is Samentingers idea about code extending other code should just as well extend the documentation. “From the viewpoint of a class’ reuser there is no difference between an overriding and an extending method”[5]. Only by containing all the information in a database can new innovative types of requests be implemented with minimum work.

The EJD project proposes several tags for identifying various chunks of documentation. Here we present a limited but diverse extract of these tags to display some of the visions behind EJD. Each chunk is identified by a tag of the form

“@name”.

@bug Specifying bugs in packages, classes or methods enables i.e. listings of all bugs in the system or parts of it. Thereby i.e. the project manager can get an overview of the condition of the system in development.

@todo The todo’s will be efficient for programmers to note down unresolved issues. By placing this information in the code, the presentation system could be instructed to show only the todo’s of certain parts of the system, i.e. the parts containing the authors name (specified by an @author tag).

@job Enables traceability between the source code and (change) requests. All code concerning a certain request could be displayed.

@designpatterns Specifying the design pattern used along with the entities (i.e. classes or methods) that are affected by it, enables the programmer to easier understand the structure of one or more classes.

@threadsafe/@notthreadsafe Multithreaded issues is crucial information when reusing code. The java language support synchronisation of object access, however these synchronisation’s are sometimes intentionally left out to increase runtime speed. The javadoc utility has no direct support for multi-thread documentation

PRESENTATION

The presentation of the documentation is done with JSP — a language for creating dynamic web pages[3]. JSP has the ability to understand custom made tags. The EJD project presents a framework of custom tags targeted documentation presentation. There are five categories of tags in this framework: iterator, if, information and links. The fifth category is for creating custom userinterfaces. The ‘iterator’ iterates over a set of entities, i.e. classes as seen in the example below. The ‘if’ executes its body if the entity it corresponds exists. The “<class>” in the example is an if-tag. The ‘information’ tag outputs information, the “<class-text-displayname/>” is such a tag. The ‘link’ tag produces a link with information on which frame to display the result, which JSP file should present it and parameters to the database information fetcher.

As the tags only output raw information, mark-up tags needs be inserted when setting up the system. However this empowers many formats of presentations to co-exist in the EJD, i.e. HTML, L^AT_EX or how about documentation on the mobile phone using WAP.

The tags also contain business logic which enable i.e. an iterator tag to sort the information to iterate according to a given parameter. A list of all classes in the EJD system would yield the following code:

```
<class-iterator sort="alpha">
  <class>
    <class-text-displayname/>
  </class>
</class-iterator>
```

A <link> tag could encircle the <class> tag to enable navigation features tied to the class. As tags can be combined in an unimaginable number of combinations, new exciting request are just waiting to be discovered.

SUMMARY

The vision of EJD is to build a documentation presentation system presenting information tailored to the user. We have presented the basic ideas of the EJD and showed some of the possibilities for the system.

A preliminary report in english on the EJD system and its implementation has been written and is freely available when contacting the author at kbilsted@it-c.dk

REFERENCES

1. Berglund, Erik, *Use-Oriented Documentation in Software development*, Linköping university, Sweden 1999
2. Friendly, Lisa, *the design of distributed hyperlinked programming documentation*, Sun Microsystems, 1995
3. Joy, Bill and Guy Steele and James Gosling and Gilad Bracha, *The Java Language Specification 2ed*, Addison-Wesley Pub Co., 2000, isbn: 0201310082
4. Pelegri-Llopart, Eduardo and Larry Cable, *Java Server Pages Specification v1.1*, Sun Microsystems 1999
5. Samentinger, Johannes, *Object-oriented documentation*, Johannes Kepler University of Linz, Austria
6. Schwabe, Daniel and Gustavo Rossi and Simone D. J. Barbarosa, *Systematic Hypermedia Application Design With OOHDM*, Hypertext 96, 7th. ACM Conference on Hypertext.