JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 4, July-August 2003

Dynamic Caching Design Proto-Pattern for J2EE Web Component Development

Serestina Viriri, Bindura University of Science Education, ZIMBABWE.

Abstract

This article covers how to extend the scope of caching to uncacheable content. It shows how to optimize the performance of J2EE Web applications by caching some of the dynamically-generated content. In this paper, I have identified a design proto-pattern, named Dynamic Caching, which addresses the performance impact of the dynamically generated Web content.

1 INTRODUCTION

The system needs to serve dynamically generated Web content with much shorter latency. Dynamic contents are increasing rapidly on the Web such that, caching dynamic contents becomes an increasingly important issue that affects the scalability of the Web. The performance of the frequently changing Web content should be improved in order to handle the increased volume of content updates. One study of Web accesses shows an 85 to 87 percent rate of repeated access to dynamic objects [Rabinovich01]. Thus it is important to design dynamic content in a cache-friendly way and to extend cache benefits to dynamic content as much as possible.

The Dynamic Caching design proto-pattern presented here uses JSP Custom Tags to convert dynamic content which does not frequently change into html files within the application scope, thereby promoting significant optimizations in the performance of Web applications since the data will be cached and made readily available to clients, and template caching that separates explicitly dynamic and static portions of the page. This technique is mainly aimed at client caches.

UML sequence and class diagrams are used to show static relationship between classes and dynamic interactions between objects at runtime.

Cite this article as follows: Serestina Viriri: "Dynamic Caching Design Proto-Pattern for J2EE Web Component Development", in *Journal of Object Technology*, vol. 2, no. 4, July-August 2003, pp. 113-117. http://www.jot.fm/issues/issue_2003_07/article2

2 THE DYNAMIC CACHING DESIGN PROTO-PATTERN

Context

The system needs to serve dynamically generated Web content with much shorter latency. The performance of the frequently changing Web content should be improved in order to handle the increased volume of content updates.

Problem

Unlike legacy cache products that were designed to cache static content only, dynamically generated Web pages should be also designed in a cache-friendly way. When the dynamically generated content is not cached, the application will be characterized by poor performance due to:

- Extra request processing resulting in poor latency.
- Creation of processing overhead on the server.
- High bandwidth consumption.

Forces

- Both static and dynamic Web pages should be cached.
- Some dynamically generated Web content that appear relatively static, should be cached as static content.
- Dynamic and static portions within a page should be explicitly separated.
- Too much extra requesting processing is required resulting in poor latency if caching is not considered.
- Creates processing overhead on the server as each request generates a new Web page without taking advantage of the previously generated pages if there are no modifications.

Solution

Use content delivery tools to propagate content updates to their caches, thereby bringing the content closer to the clients for availability and even improving latency. Some dynamically generated Web content that appear relatively static with the same unmodified content is converted into static content for easy caching. Frequently changing content is designed in a cache-friendly way that explicitly separates dynamic data from static templates.

Structure

Figure 1 shows the class diagram representing the Dynamic Caching design proto-pattern.

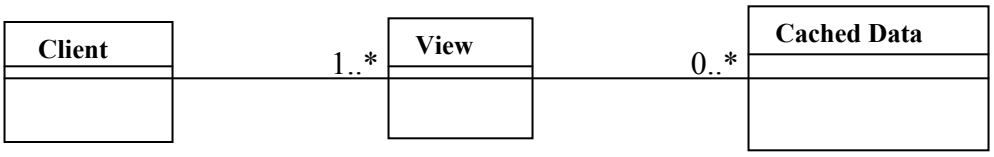

Figure 1 Dynamic Caching design proto-pattern class diagram.

Participants and Responsibilities

Figure 2 shows the sequence diagram representing the Dynamic Caching design proto-pattern.

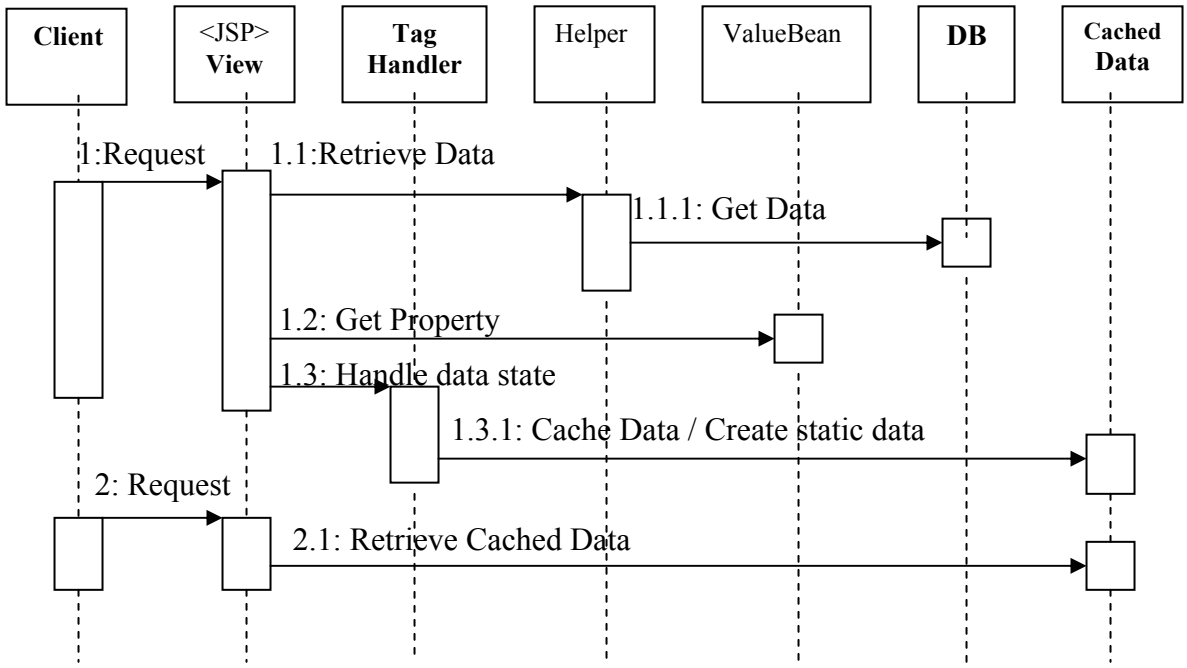

Figure 2 Dynamic Caching design proto-pattern sequence diagram

Strategies

- **Custom Tag Strategy**
  Cache management is implemented using JSP custom tags (JSP 1.1+). Logic implemented within the custom tag distinguishes the Web content that frequently

changes from the one that does not frequently change. Therefore, it determines the content to convert into static content. These tags are very powerful and flexible, but require a higher level of effort.

- **Template Caching Strategy**
  Template caching separates dynamic and static portions of the page explicitly. The static portion is augmented with macro-instructions for inserting dynamic information. The client caches the template, and downloads only the bindings for every access instead of the entire page.

Consequences

- **Improves Performance**
  Generating a display that includes numerous subviews may be fast, as most of the content is readily available. Bandwidth consumption and latency are reduced while server load is eased.
- **Improves Application Reuse**
  Business logic that is provided by these custom tags can be reused over a multiple of applications.
- **Cache Consistency Problem**
  The rate of stale delivery can be high if there are no mechanisms to enforce the freshness of cached data.

Related patterns

- **Evictor**
  The Evictor pattern describes how and when to release resources such as memory and file handles to optimize resource management. It monitors the use of a resource and control its lifecycle using some form of strategy such as Least Recently Used or Least Frequently Used. Periodically, the application releases or evicts resources that are not frequently used, and maintains in memory the resources that are frequently used.

3 CONCLUSIONS AND FUTURE WORK

The implementation of the Dynamic Caching design proto-pattern reduces bandwidth usage and latency, thereby promoting significant optimizations in the performance Web applications. Object hit rates are more than 70%.

Since J2EE design patterns are characterized by the continuous improvements and refinements, identifying and perfecting these patterns is still an ongoing process.

REFERENCES

[Shall01] Allan Shallow, James R. Trott: *Design Patterns Explained: A New Perspective on Object-Oriented Design*, Addison-Wesley, 2001.

[Rabin01] Michael Rabinovich, Oliver Spatscheck: *Web Caching and Replication*, Addison-Wesley; 2001.

[Brown01] Simon Brown et al: *Professional JSP,* 2nd Edition, Wrox Press, 2001.

[Alur01] Deepak Alur et al: *Core J2EE Patterns: Best Practices and Design Patterns*, Prentice Hall PTR, 2001.

[Gamma94] Erich Gamma, et al: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[Apple] Brad Appleton: *Patterns and Software: Essential Concepts and Terminology*. Online: http://www.enteract.com/~bradapp/

About the author

Serestina Viriri is a Lecturer at the Bindura University of Science Education, ZIMBABWE. E-mail: sviriri@yahoo.com.