alkindi

Alkindi Software Technology

Introduction

Alkindi designed a state of the art collaborative filtering system to work well for both large- and small-scale systems. This document serves as an overview of how the features of our software technology accomplish these goals.

The qualities that contribute to the robustness, scalability and adaptability of our software fall into several categories. These range from the object design to the software development process itself.

This paper addresses each of these qualities in turn. We begin with an overview of the overall application architecture. We then describe the characteristics of the Recommendation Engine implementation itself, which is most likely to be reused. Following this, we outline of our development process, which we feel adds a great deal to the quality of our software. Finally, we include a brief overview of possible future developments in order to give the reader a complete idea of the state of our software technology.

Distributed Architecture

The distributed features of the software are described here.

Server Environments

Front-end and Middleware code in Java

All code is written entirely in clean-room Java with no use of custom native interfaces. This allows it to be run on both small-scale servers and larger scale systems without porting or recompilation.

Database code in PL-SQL

PL-SQL was used for the database code. It is ready to use on any Oracle platform, running under Linux, Windows or Solaris.

Middleware Environment: Enterprise Java Beans

Manager objects exposed as EJBs

The manager objects are wrapped in interfaces that expose them as Enterprise Java Beans. This takes advantage of the features noted above to provide these services as distributed components. Each manager object becomes a type of EJB.

Java Enterprise Environment

The Java 2 Enterprise Edition (J2EE) environment provides a number of important features to the middleware.

- Scalability
  Code running under J2EE environments can be easily scaled up to serve additional users by adding more servers to a cluster.

- Load balancing
  J2EE servers can balance load between identical servers in a cluster. The details differ among commercial J2EE implementations, but most provide load balancing based on monitoring the CPU and memory utilization of the servers in a cluster.
- Failover
  The same services that provide load balancing also facilitate failover in the event a server becomes unresponsive.

Front-End Components: JSP/Servlet Engine

The front-end code runs on a separate set of J2EE servers so that it can scale separately from the middleware recommendation engine. For ease of development, it was written as JSP pages.

Back-End Database

The database code provides two classes of services to the middleware.

Online Unit

The online database handles middleware requests for data. This code is optimized to run quickly and service the largest possible of simultaneous requests.

Offline Unit

The second is the offline unit used for computationally intensive operations. The only code different from the online database is that used to merge data back into the online database.

Replication

The database is ready for replication, which can be used to handle increased load as the need arises.

Administrative Functions

Several administrative features are provided as part of the Recommendation Engine implementation.

Server control scripts

UNIX shell scripts control server operations. This is entirely typical of any J2EE server, and similar scripts can be provided for Windows or any other server environment.

Server logging facilities

On the middleware, software that captures logs and writes them to disk is written in Java and controlled by a UNIX shell script. These logs are published via the Java Messaging Services provided by the J2EE environment. This allows logs from one machine to be captured by another.

Front End logging is performed using the default management tools provided by the J2EE implementation used on that machine.

Production Software

Alkindi's proof-of-principle system demonstrates how one may build a robust and scalable

production system using Alkindi's recommendation technology. Much of the software used was free or low-cost.

Linux Mandrake 7.x

Oracle 8i Enterprise

JBoss EJB Server

Resin JSP Engine

Spyder Message Queue (now JBoss MQ)

WebTrends

Object Design

The basic software design is based on several objects implemented in the Java programming language. The objects are divided into two major categories.

Java Data Objects

This set of objects consists of data structures that abstract the basic business objects required by the system. Objects representing Users, Products, Recommendations and Ratings are all part of this category. These objects all have fairly simple representations in Java, and have very few methods beyond the ones required to create a given object.

A limited number of more complicated objects are also included in this category. These are dedicated list objects, providing two benefits to higher-level code. First, they provide type-safe methods for accessing the members of a collection. Second, they encapsulate a good choice of collection class for each type of object contained in a list. In addition to the collection access methods, some list objects also provide type-safe conversion methods to produce a list of comparable objects of a different type (e.g., a RatingList can be converted to a ProductList).

All these objects are serializable and therefore transportable over any Java network protocol (e.g., via RMI).

Keeping the data objects simple and separate from the functionality described below has further advantages once one considers the packaging of business services for use by arbitrary platforms. This is described in greater detail in the following section.

Java Manager Objects

These objects provide the fundamental services implementing Alkindi business processes. These services break down further into two sub-categories.

Business Processes

The first service subgroup responds in real-time to the requests from external users. Adding a User to the system, or allowing a User to rate Products or retrieve Recommendations are central to this set.

Clustering Algorithms

The second subgroup of services is the set implementing the clustering algorithms

themselves. Because some of these algorithms may take considerable computational resources, they are better utilized in an offline-processing context.

Extensible Design

In order to provide maximum flexibility, several steps were taken in the design and implementation of these objects.

All objects are stateless machines.

Their methods accept as parameters and return values only Java primitive types and those contained in the set of Alkindi Data Objects described above. Therefore, all parameters and return values are known to be serializable.

When exceptions occur, the methods only throw instances of a dedicated exception object.

Methods that modify the database do so in an atomic manner.

This is the ideal case for wrapping the functionality in an arbitrary distributed component model. In the proof-of-principle system, Alkindi used Enterprise Java Beans as described below. However, it is possible and desirable to use a different model to provide access to these functions from an arbitrary client platform. This is detailed in the Natural Extensions section on XML-RPC.

Alkindi Development Process

Procedure

Alkindi adheres to proven methods of software development, incorporating several phases of analysis and rigorous testing standards. A brief outline of our procedures follows:

Requirements Specification

Any project begins with this phase, the collection of basic requirements.

- **Business Requirements Analysis**

  In this step, the problem to be solved is defined. Identifying the users of a system and analyzing what they need to do is part of this step as well.

- **Functional Specification**

  The qualities of the system that will solve the problem are assembled into a set of Functional Requirements. This may require a good deal of mathematical analysis and prototyping.

- **Test Plan**

  Once the Function Specification has been settled, a test plan for the software is created. In general, this boils down to verifying that the new software accomplishes the goals set down in the Functional Specification and does not interfere with other software.

Implementation

- **Software Design**

  While the Functional Requirements can be seen as answering the questions of what

the system is to do, the Software Design answers the question of how the system will implement the Functional Requirements.

- **Development**

Only now does programming the software commence.

- **Unit Testing**

Every component is tested by itself in a test environment using a set of utilities to verify it functions according to the specification.

QA

Before software is accepted as part of the Alkindi system, it must pass a rigorous set of Quality Assurance tests.

- **Integration Testing**

At this time, the new software is integrated into the entire system and its functionality is tested in an end-to-end manner (e.g., using the production-quality user interface and hardware).

- **Regression Testing**

Finally, a set of exhaustive tests is performed to determine if the new software affects the operation of existing software.

Tools

Alkindi uses several tools in order to facilitate the process outline above.

Software

Several third-party software packages enhance our ability to adhere to our design standards.

- Rational Requisite Pro
  Alkindi uses this program to organize Functional Specifications and track changes to individual requirements. It also assists in creating Test Plans.
- Rational Rose
  Alkindi uses Rational Rose to develop Software Designs using the Unified Modeling Language.
- StarBase StarTeam
  For source code management functions, including source control and change management, Alkindi uses StarTeam. We also use it for problem tracking and assigning responsibilities for code changes and fixes.
- Microsoft Project
  Alkindi uses Project for overall project planning and task assignments.

Networks

Alkindi's development team has assembled several networks in order to smooth the development process. These include separate networks for programming, testing, staging and production implementations.

Documentation

As implied in the previous outline of development procedures, Alkindi produces professional documentation for each stage of development. Alkindi has also written documents for Network Design and Server Configuration. Overall policies, such as Alkindi's coding standards, are also documented.

Possible Future Developments

Optimizations

Database

We would like to make several changes to the online servers, in order to increase the number of clients served by a single database server. These changes would:

- Streamline table definitions
- Increase speed of statistical calculations
- Calculate more statistics for Recommendation during OLAP

Server Software Tuning

Reducing "ramp-up" time for the server can increase the performance of the Enterprise Java Bean environment. Also, several parameters of the environment can be tuned, such as those governing database connections. Modifying these settings can increase performance and throughput under load. Accomplishing these goals require we do the following:

- Analyze steady-state distribution of components under load
- Test different middleware environment parameters

Middleware

We are considering several changes to the middleware environment that can improve overall system performance:

- Use sessions to cache calculated data
- Calculate recommendations in advance where possible
- Improve speed by switching to faster EJB engine

Natural Extensions

XML-RPC Middleware Model

We would like to build an XML-RPC model for the Managers' methods. The main advantage would be to increase variety of systems that can use the Recommendation Engine. This would involve the following sub-projects:

- Wrap Manager objects in servlets
- Translate Manager APIs to XML-RPC using XML model of Data Objects

Other Product Types

After additional research to verify the appropriateness of the algorithm for working with other types of products, testing the software with other product types is a logical next step.

Database Independence

Besides porting the code to other databases, Alkindi intends to reduce reliance of the current implementation on Oracle-specific features. For example, the code uses Oracle Table Spaces to perform offline processing and merge results back into online data. Comparable features of other enterprise databases must be investigated.

Programmatic Interfaces

Some integrators would benefit from having direct, in-process access to the Recommendation Engine. A multiplicity of approaches requires further analysis on our part. We currently believe that we would first implement a C++ API to the code, which could then serve as a basis for integrating with other languages.

XML Model of Data Objects

The XML model of the Data Objects should be formalized into XML Document Type Descriptors.

Improved Selection Manager

Enhancements to the code that selects products for users to rate. This improves how the algorithm strikes a balance between learning more about individual users and learning more about the products.

Additional Administrative Tools

In order to take advantage of the data provided by the system, we will develop management tools with web-based interfaces. The ones currently in the planning state are:

- Data Management Tool
  This would provide feedback of algorithm performance and help administrators adjust configuration parameters to provide better recommendations. Additionally, such a tool could be used to view information about members and how they use the system.
- Online Server Management Tool
  Some of the server environments used do not have their own web-based management tools. Alkindi is considering developing its own to provide more convenient access. Most likely, this would involve adding a module to an existing server management console.
- Remote Log Reporter
  Some deployments could conceivably require that logs from several machines be gathered and viewed on a single machine. Since the log facilities currently use a Message Service interface with a publish-subscribe model, we are in a position to create a log consumer that can connect to remote machines and display logs in real-time and save them for later review.