Operational Assessment Support Information System (OASIS)

Glenn D. House Sr.
2Is Inc.
75 West Street
Walpole, MA 02081
(508) 850-7520 x202
glenn_house@2is-inc.com

Charles A. Cox
Franz Inc.
2201 Broadway, Suite 715
Oakland, CA 94612
(510) 452-2000 x125
cox@franz.com

ABSTRACT
In this paper, we describe an Operational Assessment Support Information System (Oasis) predominantly implemented with Lisp and Lisp based building blocks.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Lisp, AllegroCache, AllegroServe, WebActions, and Backbase and Features – *distributed computing, Software as a Service (SaaS).*

General Terms
Design, Languages.

Keywords
Lisp, expert system, logistics, supply chain, forecasting, WebActions, Allegro Cache, SaaS, distributed computing.

1. INTRODUCTION
2Is Inc. has developed an Operational Assessment Support Information System (OASIS™) application. OASIS™ verifies the output of Enterprise Resource Planning (ERP) logistic systems used by, for example, parts suppliers and manufacturers. Oasis then recommends a set of corrective actions when conditions (rules) are violated. The System has been implemented, with the exception of AJAX controls, entirely in Lisp, and is served using the Software as a Service (SaaS) model. Verification is performed using raw ERP data as well as other database sources. OASIS independently computes relevant ERP functional outputs and then compares the results against OASIS' own results. A production rule based expert system determines the most likely causes and effects of the specific difference between expected outputs of the ERP system. OASIS then recommends corrective actions for consideration. A scoring mechanism is generated so users can understand the health of their population of repair parts, where health of the population is defined and the number of critical discrepancies per part.

OASIS™ is a secure, web-based, collaborative application, hosted at a Government Owned Contractor Operated (GOCO) data center rated to handle top-secret information. Where ERP logistic systems are focused on individual parts, OASIS provides support for families of parts, also known as populations. OASIS ranks populations according to discovered anomalies and reports a score. These populations can be easily explored in a one-click, web-based interface.

Designed to be super-scalable, OASIS is capable of serving millions of part numbers while providing access for hundreds of simultaneous users. By taking advantage of 2Is proprietary distributed computing implementation, analysis and production rules are evaluated in parallel, consuming resources required to meet a particular time stationery window. All data is aggregated in a replicable object-oriented persistent data store.

Being a web-site, OASIS has minimal intrusion into the user's desktop resources. Additionally, no interaction with a user's local registry or file system is necessary. All that is required is an Internet Explorer compatible browser. OASIS uses customized AJAX controls, which result in a very efficient presentation and a one-click-away interface which is usually reserved for desktop applications.

OASIS is currently in test at the Defense Logistics Agency and a major defense contractor. Its Lisp language implementation uses AllegroCache as its distributed object database and a proprietary Lisp web

server based on AllegroServe. AJAX web browser components are supplied by BackBase[1].

2. Scalability

Scalability was a large issue to be tackled in OASIS. There were three areas for scalability: subscriber access; persistent store, and data analysis. OASIS uses an integrated load balancing approach implemented in our custom version of AllegroServe. This approach allows for the expansion of hardware in the front-end to handle simultaneous user load. An attribute of this control allows for the replication of the master data server as required should connection to the persistent store ever become a bottleneck.

OASIS must perform forecasting, evaluate production rules for its expert system and perform numerous calculations on each part in the database. Many of these operations are costly in time and space. A parallel architecture was designed such that a single control executive, called DAC (for Data Acquisition and Crunching), allocates tasks through workers that autonomously and asynchronously communicates with the database. Through these workers, blocks of parts are processed simultaneously. The DAC controls how many workers, which machines, block size and many other attributes. It is controlled through a web based application call the OASIS Management and Control Center (OMCC). Each worker's lifetime depends solely on the needs of the DAC and numerous self controls are built into each worker – such as self-destruct.

This data parallelization approach allows for the preparation of data as a function of hardware and bandwidth while minimizing the effects of a growing population of parts.

AllegroCache is used as the database for OASIS. This provides a natural persistency to the Common Lisp Object System (CLOS) objects. Furthermore, the dynamic nature of CLOS and AllegroCache makes it easy to redefine the database schema after the database has already been built and is one key to our dramatically reduced development and maintenance cycle times.

[1] Backbase, (founded in 2003) is a pioneer and leader in the emerging commercial RIA industry and we provide our customers with world class RIA solutions, http://www.backbase.com

3. Web-based User Interface

3.1 Normal Interface User Description

The user interface to OASIS is an interactive tabbed window for specific task areas. The initial tab presents a table of parts for a selected population. Each entry in this table lists the part number along with information about the part, such as its current price, quantity on-hand, forecasted demand (calculated by OASIS), as well as about a dozen other pieces of information. This table is dynamically sortable by any column and searchable by part number and other attributes.

The user can select a particular row, corresponding to a particular part and can find additional information about the part using windows. The windows slide out from the side of the main window to display the part's demand history (textually or by chart), demand forecasts, and the systems in which the part is used.

The user can drill down deeper into a part, by switching to a tab that displays more information about the part, such as stocking information and codes that are meaningful to part manufacturers. This page also presents buttons that pop up windows of tables for procurement history and details of stocking information as well as forecasting results.

On the main user interface window, another tab presents categorizations of the results of the OASIS expert system. Each "rule" is labeled with a code and the user interface can display all the violations either by code, or by other criteria such as priority or the system in which it belongs. This organization is presented to the user as an interactive tree dialog. On the left side of the window is the tree and on the right side is a detailed description of whichever item in the tree is currently selected.

A three tier context sensitive help system is available throughout the user experience. Dynamically calculated tool tips explain the myriad of acronyms. Context sensitive case studies allow users to get the help and guidance they need similar to the exceptions they are trying to resolve. A full feature help facility, complete with video tutorials, pdf documentation and a hypertext user manual is available.

3.2 User Interface Implementation

The original implementation of OASIS was as a Windows[2] application built using Allegro Common Graphics. In order to reach a wider audience, minimize distribution costs (while still being able to have central control over the parts data), a secure http web site was created using Allegro WebActions. The widgets for the windows interface within a browser are from an AJAX-based widget library provided by a third-party company, Backbase[3].

This is the first known attempt to use the Backbase interface with Allegro WebActions. Indeed, this may be the first time Backbase has even been used with a Lisp server. Since Allegro WebActions and AllegroServe are designed to provide HTML data via HTTP, and since Backbase is designed to receive XML data via HTTP, we needed to invent some machinery for translation that had to be efficient and fast. This approach consisted of wrapper XML forms around HTML files that are expanded in the normal course of Allegro WebActions' processing.

AllegroServe includes an HTML rendering macro that very easily maps Lisp s-expressions to HTML expressions. Backbase, however, uses an extensive set of XML tags and attributes to control its widgets. Generating such XML expressions goes beyond what is immediately available with the AllegroServe's HTML macro. In some cases, we created template files, known in Allegro WebActions as "clp" files that contained XML wrappers. In other cases, we had to use Lisp directly to write the XML expressions.

3.3 Testing the Allegro WebActions separation of tasks claim

Allegro WebActions allows user interface developers to build web sites using a separation of content and code. Specifically, we had a non-programmer web page designer become familiar with what Backbase provided then designed the visual interface experience. This designer developed a prototype shell with dummy data that would be replaced by calls to the WebActions clp (Lisp) functions that would provide the actual dynamic data.

The task of converting the designer's shell to a dynamic web site was straightforward. The challenge was to continue having the web developer enhance the interface after it had been "fitted" to the application. At one point, new static pages with dummy data were generated for the web designer to use. This approach ended up failing so the Lisp developer, who had by then become very familiar with Backbase, eventually handled the visual changes as well as the backend development.

Thus, we had much initial success with separating the web page design tasks from the programming tasks, but after fitting the interface to the application, we had only limited success maintaining the developer/designer separation.

4. Results

OASIS was developed using 2Is Software Development process which has been successfully audited against the SEI CMMI Level 3 standards. Our team is geographically dispersed. SEI CMMI is a standardized model for system and software development. It was developed at Carnegie-Mellon University.

Our initial OASIS implementation, released in June of 2007, was able to process 50,000 parts in 20 hours. It also served a population averaging 3,000 parts to the user's desktop in one minute.

Our current release (Nov 2008) deploys our parallelization approach; we process 60,000 parts in less than 2 hours. Tests of our system indicate that data processing time is logarithmic with workers and machines. Processing time for millions of parts is now proportional to the hardware infrastructure available to the DAC.

Another area for optimization was load time of parts to the user's desktop. Improvements in architecture and database schema allowed for users to gain access to 20,000+ part populations streamed to their desktop in less than one minute under typical conditions. These population sizes far exceed typical sized populations of 1000. Once streamed, most operations, including sorting and searching, are instantaneous. The speed at which these optimizations were developed, tested and deployed were only possible due to the flexibility of AllegroCache's redefinition capability and the Lisp environment.

---

[2] Windows is a trademark of Microsoft Corporation

[3] Backbase is a registered trademark of Backbase Inc.

5. Conclusions

Lisp has proven to be an effective implementation language for OASIS providing well known and documented productivity and execution speed[4]. In addition, AllegroCache has shown to be effective in storing and retrieving a real-time database backed web application. AllegroCache's flexibility in redefinition has reduced the development and maintenance cycle, allowing the development team to be responsive to the customers' feedback. Further work needs to be done to integrate a web authoring environment into our design methodology. WebActions allow for the separation of content and presentation in addition to driving AJAX controls. We attribute most of the implementation success of OASIS to using an all Lisp components.

[4] Gat, Eann, 2000 Lisp as an Alternative to Java, Winter 2000 Intelligence, A Publication of the ACM, JPL, California Institute of Technology,