

# The Catacomb Project: Building a User-Centered Portal the Conversational Way

Mark Ginsburg  
University of Arizona, MIS Department  
mginsbur@bpa.arizona.edu

## ABSTRACT

Enterprise computing is marked by large-scale information systems, such as databases, document management, and groupware that present significant obstacles to consistent cross-application use: dissimilar user interfaces, incompatible security schemes, and the undesirable property of serving only parts of the user community (islands of use) and accessing only some of the enterprise knowledge assets (islands of information).

World Wide Web (WWW) architectures do not solve this problem directly. WWW software components are combinable in many clever ways but until recently there were no specific efforts to solve the enterprise computing problems of islands of use and information.

The situation is changing now with nascent efforts to architect Web Portal systems with small software modules, for example with Java “portlets” or the Python-based Zope framework [15]. These are program-centric approaches to coalesce information sources and unify the query interface without inherent user modeling. This paper discusses in detail an alternative: a user-centered, conversational portal which extends the ALICE chatbot technology platform and links the user conveniently to information resources, such as Web Services, with specialized query routing. The approach offers scalability, extensibility, and coordination between end-users and developers. A university Intranet implementation, the Catacomb system, is presented and discussed to illustrate the advantages of the conversational Web portal approach.

## Categories and Subject Descriptors

H.5.2 [Information Interfaces and Presentation]: User Interfaces – *evaluation/methodology, theory and methods, user-centered design*; H.5.3 [Information Interfaces and Presentation]: Group and Organization Interfaces – *theory and models, web-based interaction*.

## General Terms

Design, Experimentation, Human Factors, Languages.

## Keywords

Conversational Portal, ALICE, Query Routing, Portal Design

## 1. INTRODUCTION

Information islands - silos of information separated by access barriers - have been bemoaned in the MIS literature over the past several decades as being strategically inferior [4, 10, 11] and economically inefficient to bridge with converters or adapters [5]. We see this in large businesses and large universities that feature physical sub-organizations (medical school, law school, business school, etc.) and partially overlapping user communities (faculty, staff, students).

The Web does not solve the problems of access across organizational subunits in federalist firms or universities: the fundamental problems remain of dissimilar search and retrieval interfaces, wide-ranging users’ community of practice which frames the query context, and unknown *a priori* relevance of the information resources to the individual or workgroup task at hand. As Etzioni points out [3] an unstructured Web can just as easily be a “quagmire” as it can a “gold mine”.

One avenue to approach the problem, which is the focus of significant development effort of late, is the use of programmatic Web portal architectures [15] which make use of small Java servlets (Apache Jetspeed framework) or Python routines (Zope) which can access Web Services and other information resources to build an information kiosk. Both approaches offer a set of core APIs to access the portal components. The idea is to lessen information asymmetries and to unify information access with a “one-stop” portal interface. But how does such a portal grow? How can we utilize user experiences? The program-centered solution does not have answers to these important questions. This paper addresses these issues with a novel user-centered web portal implementation; the Catacomb project. By combining the technology platforms of conversational agents and query routing, we provide a new kind of enterprise information kiosk. It can converse with the end-user in real time, scale to handle popularity, and be extended to add in new functionalities. Furthermore, user queries which are not routed are still answered with basic chat, and are amenable for post-processing (conversational mining) in a convenient XML format to suggest to the developers which new features should be added next.

This paper presents the Catacomb architecture, and demonstrates how components (Simple Object Access Protocol (SOAP), Web Services, ALICE chatbot Artificial Intelligence Markup Language (AIML) files, Rhino JavaScript, JSP pages,

and Java Beans) are bundled into an intuitive portal interface. We conclude with future directions to improve system coordination and remarks on how other research efforts can be integrated.

## 2. PORTALS AND ENTERPRISE INFORMATION RESOURCES

The ambitious role of a Portal is to quickly and efficiently match as large a cross-section of the end-user population as possible with as broad a set of information resources (the **Information Space**) as possible. There has been recent activity developing a consistent set of programming APIs to provide basic Portal functionality, for example the Python-based Zope framework [15]. In Enterprise computing, there is a set of overlapping end-user constituencies, a **Constituency Space**. Many Information Technology (IT) systems cater to individuals only, so that a user’s interaction with an IT system is not sharable across users or workgroups. This suggests another design goal of a Portal --- to be able to identify, over time, emerging Constituencies (for example, dynamic workgroups that are constantly forming and reforming and which span organizational boundaries, as described by Nonaka [12]).

### 2.1 Portals and the User Experience

In the Web Environment, the end-user starts a session by accessing a Portal page. The general design goal should be for the Portal to interpret the user’s activities (text input, mouse clicks, etc.) and the user’s current context and provide accurate and timely information. If a portal structure is built statically, maintenance and growth is difficult.

### 2.2 Approaching the Problem in a Conversational Way

Having a conversational front-end adds a dimension of richness to the portal. A side-benefit of the front-end interaction is a convenient logging of all the conversations as users interact with the system; whether or not the user inputs keywords which map to our specialized methods --- this adds a coordination pathway between the cumulative activity of the users and the system maintainers who are seeking to increase the portal’s range of capabilities.

## 3. A Conversational Architecture for Information Island Bridging

To accomplish a conversational portal, we use the Turing-award winning ALICE<sup>1</sup> chat engine. This freely available platform produces output chat logs in XML format with a built-in XSL style sheet attached. The ALICE system uses the Web HTTP

Protocol and listens by default on port 2001. The design pattern is that of Model-View-Controller (MVC) [6, 8]. The View, which is the system’s representation to the user, is a simple Web front-end. In the MVC design pattern, if the View changes, in this case by a changed user input, the Controller is responsible for communicating the changed View to the Model (which contains the logic of the system). With ALICE, the Controller is the Web Server since the requests are Web HTTP Post methods. The Model is a set of “AIML” (Artificial Intelligence Markup) files in XML format which are composed of categories with input patterns and “Templates” which tell the chatbot how to respond if the user’s input matches that pattern. The patterns include wildcards (\*) and the ability to respond randomly, as well as redirect the query to another pattern.

### 3.1 Query Routing: Specialized Method Affordances

To function as a portal, we start the system by taking into account typical queries which might be posed by the University constituencies and add on top of the ALICE engine specialized query routing – for example, to report on class schedules, faculty office hours, local weather, local bus schedules, and so on. The query router branches out of ALICE into small parsing wrapper code we write in a combination of RHINO JavaScript and Java. Results are channeled back into the conversational front-end.

The sum total of all the methods we implement form the **Action Space** for the portal. Note that the Action Space is the end-result, under the hood we have the implementation details of how the methods access and process the data from the **Information Space**. As the users explore the Action Space, entering in keyword queries, the keywords, or tokens, are either recognized (map to our actions) or default to the generic ALICE chat behavior. This suggests the need to map unfulfilled queries to an implicit user request to expand the Action Space --- a coordination mechanism between end-users and developers. The larger our Pattern Matching dictionary (i.e. the more methods we implement) the greater our Action Space. Conversely, if the user is “floating” too long in a pure chat environment, he or she may have a negative reactions [2].

### 3.2 Query Routing and ALICE: Some Details.

There are many implementation choices to layer query routing on top of the ALICE framework. This section details our choices in the Catacomb system implementation.

Since AIML is an XML-syntax language that can be used to provide standard text and HTML output for the chat engine, and since AIML standard files include JavaScript to provide dynamic content (The Alice server includes the Rhino JavaScript engine<sup>2</sup>), we chose to use Rhino JavaScript to link to JSP pages

---

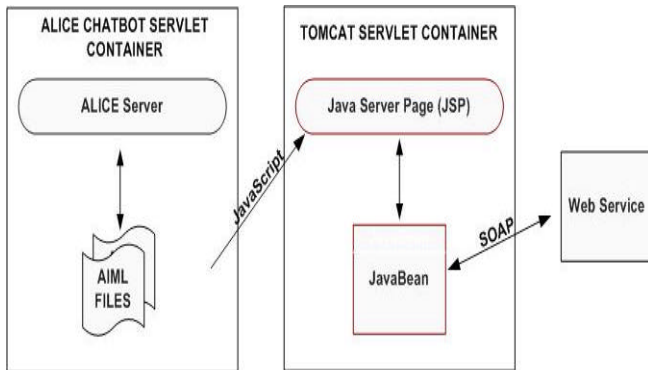
<sup>1</sup> Dr. Richard Wallace’s ALICE chat architecture has been written in several languages (for example, C++ and Java) and is downloadable from <http://www.alicebot.net>. We use Jon Baer’s Java-based ALICE Version “D”.

---

<sup>2</sup> See <http://www.mozilla.org> for more information on the Rhino JavaScript engine.

which in turn make use of Java beans, SOAP calls, and Web Services. A diagram of the relationship of these components is shown in Figure 1.

In the Catacomb system, we use Apache and Tomcat to house servlet container to deploy JSPs and JavaBeans. The JavaScript in an AIML file calls a deployed JSP page with any needed variables added to the URL as a query string (similar to an HTTP GET method). The JSP page is used to format output of the Web Service and to provide an interface between the AIML file and a Java Bean. The JSP page passes variables from the query string to the Java Bean and invokes the Java Bean methods that call the Web Service.



**Figure 1. Interrelationship of Catacomb Components**

The JavaBean used to call the Web Service gets the input variables from the JSP page, creates a SOAP message to request whatever Web Service it uses, opens a connection to that Web Service, sends the SOAP message request, gets a SOAP message response, parses the response, and formats it for display in the JSP page. The JSP page displays the Web Service results. JavaScript in the AIML file captures these results and displays them in-line with the chatbot.

Figure 2 shows the Model-View-Controller of our Query Routing Subsystem, which is subsumed inside our ALICE base framework.

In the Catacomb implementation, we modify the Generic ALICE view slightly and allocate an additional HTML Table Cell to display Query Router output, as shown in Figure 3. This accounts for the dashed line showing a linkage from the Query Router View to the Generic ALICE view in Figure 2.

### 3.3 Properties of the Architecture

The Catacomb system brings together two technological platforms, conversational front-ends and query routing back-ends. In the past, query routing has been demonstrated with search engines [14] and as middleware for data-intensive applications [9]. Our portal approach anticipates query routing being built up over time to accomplish specific pieces of portal functionality.

### 3.4 Comments on Interfaces and Usability

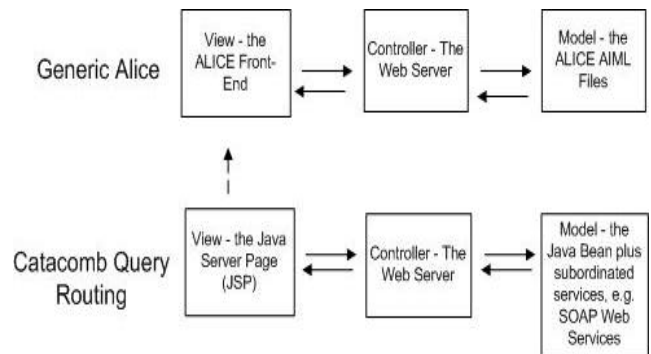
We also leave the interface design rather open ended. Having the end-users funnel into a conversational portal gives them a consistent look and feel to ask questions; if the portal recognizes the query the result can be displayed in a consistent way. Similarly, unrecognized queries are also displayed consistently and logged for possible system expansion. The portal does not impose design principles for retrieval or visualization on other major types of enterprise assets (databases, documents, spreadsheets) and cooperates rather than compete with existing subsystems.

### 4. The CATACOMB Implementation: Currency Conversion Walk-Through

In this section we walk through the Currency Conversion feature, which shows the interrelationship of the Alice system, the RHINO JavaScript, the JSP Page, and the Java Bean and its use of SOAP messages and Web Services. This example also illustrates our basic interface design.

Figure 3 shows the introductory screen. The user has two options. He or she can type something into the text box and press the “Say” button or pick an icon from the 2-dimensional array on the left. The system allows for three levels of icons; one specific to the University of Arizona, one specific to Tucson, and one “General” (shown). The icons are presented in a two-dimensional array and blank “tiles” are reserved for future feature addition. In this example, the user presses the “coins” icon which is located on the right, in the second row to invoke the Currency Converter.

The user needs to supply values to the angle-bracketed parameters; this will match a specialized query routing input pattern in one of our modified ALICE portal categories. The category’s template will invoke a JSP page for further processing.



**Figure 2. Catacomb and ALICE Model-View-Controller Design Pattern**

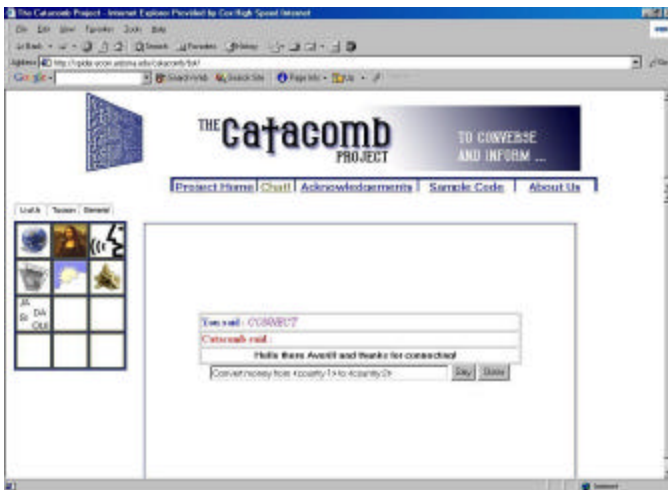
## 4.1 Under the Currency Conversion Hood – The Software Components

The first component is an AIML file that recognizes the Currency Conversion request pattern<sup>3</sup> and then calls the Java Server Page (JSP) program.

The CurrencyExchange JSP program takes the variables passed in by the Alice AIML file and invokes methods from the “currency” Java Bean. In general, the JSPs display HTML and include short snippets of code that call Java Bean methods<sup>4</sup> to display the dynamic content of the HTML page. This design has the desirable properties of separating View and Model.

## 5. Comments and Future Work

Just as XML has added structure (semantics) to the presentational HTML web, so our approach adds pragmatics



(context-based meaning) to the semantic Web. We can support pragmatics with a user-centric modeling approach in our portal design, as called for in [13]. The users make conversational requests and with their unfulfilled queries (those which did not match specialized query routing patterns) we can build conversational mining agents. The idea is to coordinate between the end-users and the developers to facilitate the addition of new features.

The conversational logs across the various constituencies also illuminate “information scent” [1] --- the foraging behavior of the user communities. And, if a set of users is involved in a structured workflow process, this approach can shed light on information foraging behavior in a specific task context, assisting

<sup>3</sup> The Currency Conversion AIML and JSP pages are online at [http://louvain.bpa.arizona.edu/projects/catacomb/docs/curr\\_ai\\_ml.html](http://louvain.bpa.arizona.edu/projects/catacomb/docs/curr_ai_ml.html) and [curr\\_jsp.html](http://louvain.bpa.arizona.edu/projects/catacomb/docs/curr_jsp.html).

<sup>4</sup> The Java Bean code is Appendix A in the online document: [http://louvain.bpa.arizona.edu/projects/catacomb/docs/XMLS\\_py-WDSL.doc](http://louvain.bpa.arizona.edu/projects/catacomb/docs/XMLS_py-WDSL.doc).

in building Process Coherence [7] in what is nominally an unstructured HTTP hyperarchy.

## 6. Concluding Remarks

This paper has presented a novel conversational portal architecture which is simple to implement, and has the desirable property of capturing user interaction over time in well-structured system logs. The key challenge is to establish effective coordination pathways between the end-user constituency groups and the developers. If we can leverage the cumulative user sessions with conversational mining, we can decrease the cycle time between new feature additions and the system will scale up smoothly. We are working toward the goal of making full use of our **Information Space** with a rich set of access techniques (the **Action Space**) in order to serve the dynamic user communities which make up the **Constituency Space**.

## 7. ACKNOWLEDGMENTS

The author gratefully acknowledges the support of the Spring 2002 University of Arizona MIS507B Data Communications Programming Team: Dan Keltner, Lead Programmer, Rusma Mulyadi, Mark Coppola, Jun Liu, Averill Cate, and Byron Marshall.

## 8. REFERENCES

- [1] Chi, E.H., et al. *Using information scent to model user information needs and actions and the Web*. in *SIGCHI Conference on Human Factors in Computing Systems*. 2001. ACM Press.
- [2] DeAngeli, A., G.I. Johnson, and L. Coventry. *The unfriendly user: exploring social reactions to chatterbots*. in *Proceedings of The International Conference on Affective Human Factors Design*. 2001. London, UK: Asean Academic Press.
- [3] Etzioni, O., *The World--Wide Web: Quagmire or gold mine?* *Communications of the ACM*, 1996. **39**(11): p. 65-68.
- [4] Evans, P.B. and T.S. Wurster, *Strategy and the New Economics of Information*. *Harvard Business Review*, 1997: p. 71-82.
- [5] Farrell, J. and G. Saloner, *Converters, Compatibility, and the Control of Interfaces*. *Journal of Industrial Economics*, 1993. **40**: p. 9-35.
- [6] Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994: Addison-Wesley.
- [7] Jain, A.K., M. Aparicio, and M.P. Singh, *Agents for Process Coherence in Virtual Enterprises*. *Communications of the ACM*, 1999. **42**: p. 62-69.
- [8] Krasner, G.E. and S.T. Pope, *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*. *Journal of Object-Oriented Programming*, 1988. **1**(3): p. 26-49.
- [9] Liu, L., et al. *AQR-toolkit: an adaptive query routing middleware for distributed data intensive systems*. in *Proceedings of the 2000 ACM SIGMOD international conference*

*on Management of data*. 2000. Dallas, Texas, United States: ACM Press.

[10] McFarlan, E.W., *Portfolio approach to information systems*. Harvard Business Review, 1981. **59**(5): p. 142-150.

[11] McFarlan, E.W., J.L. McKenney, and P. Pyburn, *The information archipelago --- plotting a course*. Harvard Business Review, 1983. **61**: p. 145-156.

[12] Nonaka, I., *A Dynamic Theory of Organizational Knowledge Creation*. Organization Science, 1994. **5**: p. 14-37.

[13] Singh, M.P., *The Pragmatic Web*. IEEE Internet Computing, 2002. **6**(3): p. 4-5.

[14] Sugiura, A. and O. Etzioni. *Query Routing for Web Search Engines: Architecture and Experiments*. in *9th World Wide Web Conference*. 2000. Amsterdam, Holland.

[15] Wege, C., *Portal Server Technology*. IEEE Internet Computing, 2002. **6**(3): p. 73-77.

**PERMISSION TO MAKE DIGITAL OR HARD COPIES OF ALL OR PART OF THIS WORK FOR PERSONAL OR CLASSROOM USE IS GRANTED WITHOUT FEE PROVIDED THAT COPIES ARE NOT MADE OR DISTRIBUTED FOR PROFIT OR COMMERCIAL ADVANTAGE AND THAT COPIES BEAR THIS NOTICE AND THE FULL CITATION ON THE FIRST PAGE. TO COPY OTHERWISE, OR REPUBLISH, TO POST ON SERVERS OR TO REDISTRIBUTE TO LISTS, REQUIRES PRIOR SPECIFIC PERMISSION AND/OR A FEE.**

**WIDM’02, NOVEMBER 8, 2002, MCLEAN, VIRGINIA, USA.**