A Common Knowledge Framework and Lessons Learned Module

Chris Knight
Caelum Research Corp.
NASA Ames Research Center, Mail Stop 269-2
Moffett Field, CA 94035-1000
cknight@ptolemy.arc.nasa.gov

David W. Aha
Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Code 5515
Washington, DC 20375
aha@aic.nrl.navy.mil

"The significant problems we face cannot be solved at the same level of thinking we were at when we created them." ***–Albert Einstein***

Abstract

Postdoc and other web-based collaboration tools are heavily adopted across NASA and in industry. Postdoc's main purpose is to support document management capabilities for distributed project collaborations. These documents, which include e-mail threads, are a rich source for lessons learned. However, they have not yet been mined, in great part because, like other web-based collaboration tools, Postdoc lacks a common infrastructure for supporting synergy between applications (e.g., a document manager and a lessons delivery module). We propose a new, object-oriented, multi-application environment for data management that will take advantage of the best-of-breed technologies and will serve as an adaptable and open architecture for continued enhancement and growth. We also describe designs for an active lessons learned elicitation and delivery module in this enhanced environment.

1. Postdoc

Postdoc, a WWW document management and distributed collaboration environment developed at the NASA Ames Research Center (Postdoc, 2000), is being used by thousands of NASA employees and collaborators from education and industry. Postdoc has proven to be an effective tool for organizing, storing, and retrieving documents of all types (e.g., word-processing, spreadsheet, presentation, image, video, audio, executable).

Postdoc has many strengths: a rich set of document management features, familiar user interface, full-featured access control and user management, e-mail list creation and archival, and a pure WWW interface. However, it was designed with document management as the sole target application and remains limited by a highly custom, dated, complex, and tightly coupled architecture (see Figure 1). The primary drawback of this architecture is that it complicates the tasks of integrating Postdoc with other tools and disparate application environments so that data can be shared between applications with a minimum of effort by the developers and integrators.

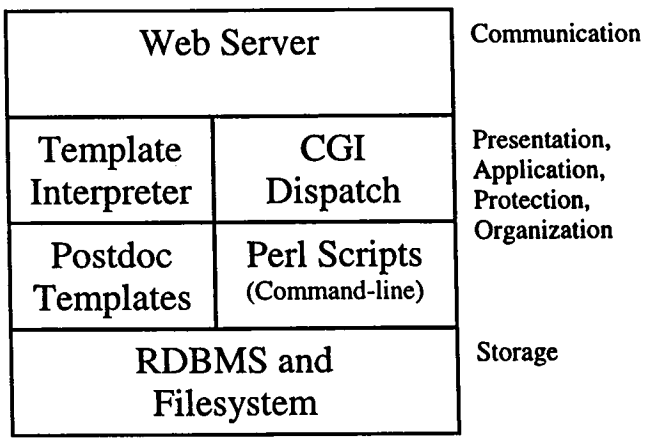

Figure 1: The current Postdoc architecture.

Meanwhile, many groups within NASA and elsewhere have requested richer knowledge management and document workflow features, as well as integration of lessons learned elicitation and management strategies. The potential synergistic benefits between a wide variety of collaborative technologies have also motivated the need for change. This paper briefly outlines our proposal for re-designing Postdoc's architecture to better support modular extensions, and an example module for lessons elicitation and delivery.

2. Revised Postdoc Architecture

Integrating these components into the existing Postdoc architecture would be difficult. Therefore, we decided to re-architect the Postdoc environment using modern technologies and standard off-the-shelf tools wherever possible—while retaining and improving much of the existing document data, usability, and features of the existing Postdoc environment. In this context, we selected a loosely coupled, modular architecture (Figure 2) to provide the most flexibility for (multiple) user interfaces, and allow us to choose and/or switch to the best-of-breed off-the-shelf software and technologies as they evolve, much like a network "stack" architecture.

The following subsections describe the high level features that will be supported. All of these features must be accessible and easy to manage and manipulate for end-users in a full-featured development language to fully

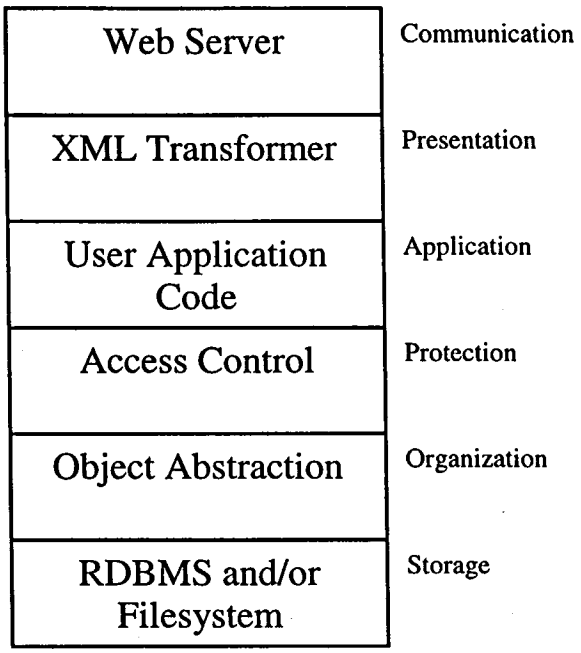

Figure 2: Proposed, modular Postdoc architecture.

realize the environment's potentials without impeding user development with complex interfaces and development language constructs.

2.1 Object-oriented architecture

First, like other knowledge management environments, it must adopt a flexible storage architecture capable of handling continuous and incremental user enhancement of knowledge data, such as metadata about stored documents and archived mailing list communication. Second, to support workflow applications, as well as user extensions of functionality (Section 2.2), user contributed application logic must also be stored in this storage architecture. These features entail using an object-oriented approach (Booch, 1994), which is a logical fit for this environment because it supports user-managed data collection (via object attributes) and application extension (via methods). The object-oriented feature of inheritance adds another layer of functionality, allowing for common features to be transparently shared across multiple objects in this environment.

2.2 End-user application extension

End users of a knowledge management environment are frequently not professional programmers, and may not have much programming experience. The redesigned Postdoc environment will be extensible by these users, providing a simple yet fully functional application language and a variety of development support tools in order to "ease" these users into extending their environment to meet their own needs with a minimum of assistance from professional application developers.

Currently the target language for end-user application development is Python (2000), an easy-to-use yet powerful scripting language with a strong user community (especially in the scientific, educational, and web application realms). However, this does not limit application development to one language. Our goal is to support a wide variety of "client application" platform languages including Java, C/C++ and others through various network API's (e.g., CORBA and XML-RPC).

2.3 Access control

Finally, in order to support a wide variety of users in a collaborative environment of this type, advanced access control and authentication must be enforced to protect data from unauthorized access and modification. Most databases, independently of whether they are object or relational databases, provide some level of access control and authentication. However, most partition access control on the table, object, or similar large scale. This level of access control is insufficient in an environment where the atom of manipulation by collaborators may be individual metadata attributes.

For example, a document may have a variety of attached attributes, such as *owner*, *name*, *keywords*, *description*, and *related documents*. In a review process, an additional attribute of "review rating" may be added/manipulated by the reviewers of the document, who may have privileges to manipulate the document's other attributes.

Few open standards exist for access control paradigms and management architectures. However, an emerging standard has been established by the Web Distributed Authoring and Versioning (WebDAV, 2000) Working Group that provides the level of control and flexibility required by the redesigned Postdoc environment. Also, WebDAV is an excellent target communication mechanism: the standards being developed are for client-server communications for web servers.

3. An Active Lesson Learned Module

Postdoc's users have repeatedly reported how they have greatly benefited by extracting lessons from archived documents (e.g., e-mail threads), thus providing them with knowledge gained during previous projects. Because lessons commonly represent best practices to pursue or alerts for avoiding known pitfalls, sharing them among project members can be of great value to Postdoc groups. For example, if analyses of inter-group communications yield reasons for mission failure, then any lessons resulting from these analyses, when properly delivered to subsequent project participants, can help avoid repeating costly mistakes.

However, users rely entirely on a reactive, simplistic textual search to find lessons, and often fail to find both relevant lessons and opportunities to apply them. This complicates the process of recognizing and applying archived lessons. Indeed, many potential lessons go unrecorded or, even if informally recorded, unrecognized.

Therefore, as a first test of the redesigned Postdoc architecture's utility, we plan to develop and integrate modules for active lesson elicitation and retrieval (Weber et al., 2000). This will semi-automate the process of recording lessons, and will automatically bring lessons to the attention of users as needed.

3.1 Knowledge sources

In this context, we define *lessons* as characterizations of project and/or program experiences that have passed criteria for reuse through some validation procedures, where the focus will be on extraction from e-mail threads and project reports. This will require imposing structure on these document sources to simplify the extraction task. For example, project reports will be structured using an object-oriented approach, where the objects will include:

- Tasks
- Deliverables
- Project member responsibilities
- Schedule
- Milestone constraints
- Resources (e.g., equipment, facilities)
- Resource constraints (e.g., due to sharing)

Additional knowledge sources will be required to link these objects. For example, a universal task list will be required to provide background on how tasks can be decomposed, their duration (i.e., how they relate to the schedule), their resource requirements, and their functionality (i.e., how they relate to the deliverables). Other objects may be required to reason about resource constraints and feasibility of project success (e.g., project member expertise). Intermediate project reports will describe recent accomplishments; reasoning strategies will be needed to identify project status with respect to intended milestones and schedules. Characterizations of this status will be used to index lessons. For example, if a project begins to slip in schedule due to the unavailability of a specific resource, and this closely matches the index of a recorded lesson, it can be brought to the user's attention to see how this problem was resolved in previous projects, and whether the result was satisfactory.

Likewise, e-mails will also require structuring so that their content can be related to project objects and their status. Although some of the information required to perform inferencing will be available via standard e-mail structures (headers) and extractable using natural language understanding techniques, it will not be feasible to capture all this information from free text. Therefore, e-mail users will be prompted/guided to identify the focal objects of their notes. Users will be able to use an extendible abstraction hierarchy that relates the project objects, and threads can be used to suggest some defaults.

3.2 Lesson extraction

We will develop a learning apprentice, using a case-based reasoning strategy (Watson, 1997), to extract cases. It will automatically analyze project documents, focusing first on identifying project failures and potential best practices. Lesson representation will be selected to promote reuse. For example, each lesson will contain at least the following fields:

- Conditions (i.e., defining the lesson's context)
- Past experience
- Recommendation
- Benefits

The conditions will refer to project objects and their status (e.g., relations between objects). To promote lesson reuse, the language for defining conditions must support generalization. For tasks, the universal task list knowledge source can be used to help define some generalizations, so long as the list implicitly defines generalization hierarchies. For other objects, corresponding abstraction hierarchies will be required to support the matching process. For example, a knowledge source must exist that defines relative temporal relations between the schedule and the milestones. Fuzzy set definitions should prove useful for this generalization process, allowing relations to be expressed in ways that are more amenable to generalization hierarchies than using raw temporal relations.

Past experience must encode the result of project efforts when the lesson's conditions existed. They must describe how the project's goals were affected, and identify the impact/severity of these effects. Again, an object-oriented representation will be used to communicate these experiences.

A lesson's recommendations will encode suggestions on how to proceed in projects where this lesson applies. Because they will be specific to the recorded project's objects, generalization hierarchies will be required to help relate them to the current project's objects.

Finally, a lesson's benefits will be described with respect to what positive impact a lesson's application will have in comparison to ignoring it. Again, this will refer to project objects (e.g., deliverables, schedule, milestones), but support primitive relations (e.g., *faster completion*) that

can be used to succinctly identify the reason for implementing the lesson's recommendation.

3.3 Example lesson extraction

E-mail messages in the current Postdoc system are ripe for lesson extraction, containing historical archives of communications often comprised of specific problems and their ad-hoc (repeatable) solutions.

An excellent example is a communication on the Postdoc Maintainers mailing list between a software developer and an individual installing the developer's software on another computing system:

> ***Subject:*** *Postdoc on RedHat 5.2*
>
> ***Date:*** *Tue, 4-May-1999, 14:42*
>
> *I am trying to install Postdoc on RedHat 5.2 and am having trouble installing eperl.*
>
> *I ran eperl from the command line to see what the output was when I tried to log in as a registered user; here is the result from the login.ehtml file.*
>
> [Technical details removed.]
>
> *I also ran eperl from the command line to see what happened when I tried to log in as a guest: Here is the output from the main.ehtml file.*
>
> [more technical details]
>
> *It seems that in both cases the script is stopping on POSTDOC_AUTO_PASSWORD. In my config.mk it is defined as: POSTDOC_AUTO_PASSWORD = 1.*
>
> *RedHat is now using egcs. Is there a chance that this or the new glibc could cause a problem?*
>
> ***Subject:*** *Re: Postdoc on RedHat 5.2*
>
> ***Date:*** *Tue, 4-May-1999, 17:38*
>
> *In include/Makefile, change:*
>
> [Technical details]
>
> *to:*
>
> [Technical details]
>
> *Apparently, this difference is fatal on Linux but not Solaris.*
>
> *This solution was arrived at by having detailed information, such as you have provided, which is the kind of information I need to diagnose and address problems remotely without my own login on a given machine.*

Obviously, from this extract, a lesson about operating-system specific changes necessary to install Postdoc's eperl component can be extracted to present to future installers. We anticipate that useful lessons can be acquired from email threads, and then made available for reuse, for a wide variety of distributed projects.

4. Conclusion

Our goal is to develop enhancements of Postdoc, a document management tool for supporting distributed project collaborations and their knowledge sources. Postdoc's current architecture complicates attempts to enhance it. We will re-architect it to support incremental user enhancements of project data, modular extensions, and advanced access control and authentication procedures. We will also incorporate an active lessons capture and retrieval module in Postdoc, focusing on capturing and reusing lessons from e-mail and project report archives.

Acknowledgements

Thanks to the NASA-Ames Research Center, and particularly to Helen Stewart, for supporting this research. Thanks also to the Office of Naval Research for a grant to the Naval Research Laboratory to support our efforts on researching and developing lessons learned systems.

References

Booch, G. (1994). *Object-oriented design with applications.* Reading, MA: Addison-Wesley.

Postdoc (2000). The Postdoc home page. [http://ace.arc.nasa.gov/postdoc]

Python (2000). The Python programming language. [http://www.python.org]

Watson, I. (1997). *Applying case-based reasoning: Techniques for enterprise systems.*

WebDAV (2000). Web Distributed Authoring and Versioning (WebDAV) Resources. [http://www.webdav.org]

Weber, R., Aha, D.W., Branting, L.K., Lucas, J.R., & Becerra-Fernandez, I. (2000). Active case-based reasoning for lessons delivery systems. To appear in *Proceedings of the Thirteenth Annual Conference of the International Florida Artificial Intelligence Research Society*. Orlando, FL: AAAI Press.