Architecting the Elephant: Software Architecture and User Interface Design for Pachyderm 2.0

Michelle LaMar, California State University, USA
Joshua Archer, California State University, USA
Tom Hapgood, University of Arizona, USA
D'Arcy Norman, University of Calgary, Canada
Tim Wang, University of British Columbia, Canada

Abstract
This paper covers the processes of designing the system architecture and the authoring interface for Pachyderm 2.0, as well as some of the fundamental conceptual strategies that enabled the development of the software. It describes the architecture of the underlying frameworks, applications, and content templates. Areas for potential collaboration and development are identified.

Keywords: open-source; authoring tool; user interface; APOLLO; WebObjects; Pachyderm

Introduction

Examine the Mona Lisa up-close while listening to professional commentary. View an old movie of Ansel Adams climbing up to the "Diving Board." Flip through pages of the Gutenberg Bible and listen to music of the period.

The power of experiencing history, art and music in such multimedia expressions is undeniable, and yet creating such a rich experience from scratch is extremely labor intensive. The San Francisco Museum of Modern Art faced just that challenge in the creation of its *Making Sense of Modern Art* exhibit. Their solution was to create an authoring tool to facilitate the assembly of media-rich online exhibitions. Thus was Pachyderm 1.0 born.

Pachyderm became very successful within SFMOMA and was used to generate more than 10 extensive online multi-media presentations. As is commonly the case with successful prototypes, its users found it indispensable, while at the same time it was running into the limitations of its quick design and development. Users within and outside of SFMOMA loved Pachyderm, but they wanted more than it could deliver. It was time to take the prototype and build a carefully architected, extensible system within which future features could be more easily developed.

Pachyderm 2.0, based on the SFMOMA's original, is well underway now as part of a partnership led by the New Media Consortium and SFMOMA and funded by the Institute for Museum and Library Services. The project brings software development teams and digital library experts from six NMC universities together with counterparts from five major museums throughout North America to build a tool well suited to both the museum and university environments.

Interface Design

The first version of the Pachyderm authoring system was built to maximize the quality of the finished presentation, not to optimize the author's experience while creating the

presentation. Its user base was very small; SFMOMA authors became so familiar with the quirks and inefficiencies of the authoring system interface that they hardly noticed them anymore. Early user tests proved however that this would not be the case for the larger audience. A goal of the Pachyderm 2.0 Project was to retain the power and potential of SFMOMA's original Pachyderm authoring system, with a simplified interface that enables a wide variety of users to create strong presentations.

In order to facilitate the easy creation of presentations in Pachyderm, the team has employed a few standard, proven methods of design. According to designer and educator Philip B. Meggs (1992), the design process is a sequence of events that begins when the designer receives an assignment and continues until the problem is solved and the solution is accepted and implemented. The process includes defining the problem, gathering information, searching for effective ideas, deciding on a solution and finally implementing.

Design of the new authoring screens was based on the existing tool in use by SFMOMA and on the results of many discussions with the main user groups of the software, namely museum curators and university faculty. In addition, the need to implement new features and functionality required redesigning portions of the authoring process.

The first goal of the design effort was to define the authoring workflow. Working with a variety of tools that ranged from iChat and Breeze screen-shares to markers on butcher paper, the development team worked through iterations of the flow of authoring, determining the easiest and quickest succession of screen interactions. The team created a flowchart document for various users with boxes, lines and arrows based on target user groups and scenarios of how certain people would use Pachyderm. By using such a graphical representation of author workflows, the team was able to ascertain the best order for all the interlinking screens, getting a user from the left side of the flowchart to the right using the fewest amount of boxes, or tasks.

In this stage of the design came the first major departure from Pachyderm 1.0. The SFMOMA workflow had authors creating individual presentation screens first, linking them to each other, and then finally grouping them together as a presentation. The development team decided to reverse the process. First an author creates a presentation, with a title and description and then they create screens within the presentation. This ordering better reflects the project-based approach of the target audience.

Once the workflow was established, the team began to sketch out the individual interface screens in the interest of creating a basic graphical system with attention to all the elements that needed to be on the screen, such as navigation and software controls such as "save" and "delete."

Special attention was paid to the interface for creating and editing presentation screens since authors spend considerable time on this task. Pachyderm 1.0 uses an approach based on HTML forms for authoring presentation screens.

Figure 1: The original forms-based interface.

Figure 2: Mockup of the proposed drag-and-drop interface.

In early meetings the team sketched out visions of something far more powerful. Given the increased sophistication of modern browsers using JavaScript, authors could be provided with drag-and-drop functionality for adding their media into a screen template. As the idea developed further, concerns were raised that while this would adding power, it might also be adding complexity that would detract from the straight-forward authoring model that endeared many to Pachyderm. The team conducted user tests and found that indeed, users preferred the forms-based approach. They liked to be able to clearly identify all the "slots" into which data could be entered. Given this feedback, the team made an unusual decision in software design – the cool technical gadget was dropped in favor of the simpler approach.

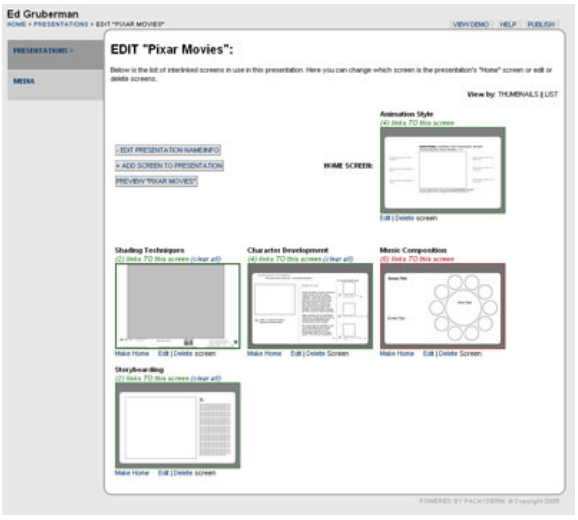

Figure 3: Mockup of the updated forms-based interface resulting from user feedback.

The next stage called for turning the designs into a standards-compliant xhtml format that could be read by Web browsers, where the actual authoring will take place. This included

mocking-up the screens with actual Web form-based buttons and controls and straightforward color and design treatment. The programmers on the team then took control of these screen designs and made them function by marrying the xhtml coding and programming elements. The testing of these working screens is underway at the time of this writing.

During this testing time, the team is paying particular attention to feedback on issues of usability and accessibility, with an eye to moving closer to a parity of visual design between the authoring and presentation screens. The interface design can not be considered complete until it has been proven usable by the intended user group.

Software Architecture

Since Pachyderm 1.0 was created to solve a set of needs internal to the SFMOMA, the developers had a certain amount of leeway in the decisions they could make in terms of platform architecture. They selected tools that would allow them to build the software faster, and integrate with the existing SFMOMA collection management database. The tools they chose to use, however, created dependencies that were very specific to the SFMOMA environment. These dependencies included Windows IIS, ASP, Visual Basic, Windows 2000 Server, SQL Server, Macromedia Generator 1.0, and Filemaker 5.

As additional online versions of exhibits were needed, the Pachyderm 1.0 software was adapted and extended, with pieces being added to support new functionality. The original system was not designed with extensibility as a primary consideration, and the consequences were such that both the database schema and the source code grew large, fragmented and difficult to maintain.

The team knew from the conception of the Pachyderm 2.0 project that two key enhancements would need to be made to the original product: platform independence and extensibility of the template set. Platform independence would allow Pachyderm to be installed and run on servers with different operating systems, different web servers and different database applications. Template set extensibility meant that future authors would be able to add new templates, or presentation screen types, to the original set of 12 that SFMOMA designed.

Refactor or Rebuild

When a software prototype is developed into a full, robust application, there are two accepted paths of development. The first is to throw away all the old code and rebuild from the ground up. This is the more traditional approach; what results is known as a "throwaway prototype." The assumption is that the developers will have learned a lot from coding the prototype and they will be able to build it better the second time around. A newer approach is to refactor the old code. This method involves isolating functional sections of the code (or modules) and carefully rewriting them piece by piece while maintaining the functionality of the whole application.

Refactoring has the advantages that the software as a whole continues to function throughout the process, making testing much easier, and that the engineers are working intimately with the original code and thus not losing any lessons learned the first time around (Spolsky, 2000). Given the small development budget, the ambitious time line and the fact

that the team programmers were not the original developers of Pachyderm 1.0, refactoring seemed to be the most practical approach. It meant that the team would not have to re-solve problems that the original developers had already solved very well, and that something deliverable would always be there in case development fell behind schedule. Since Pachyderm 1.0 was developed in Active Server Pages (ASP), a proprietary Microsoft language, the team knew that in addition to refactoring, the code would have to be ported to a platform-independent language. Thus the original approach was dubbed "Port and Refactor." The first effort proved to be a sanity check on the whole approach. The team took the prototype code, still in ASP, and created a version that could be installed on a Windows server outside of SFMOMA. This turned out to be surprisingly difficult. The team found that the old code not only relied on specific proprietary software, but in fact on specific outdated versions of that software. By the time the "Pachyderm 1.1 Rhino" release, a Windows server installable version of the software, was produced, the team had learned enough to reconsider the whole development plan.

Through the development of the Pachyderm 1.1 version, the team began to analyze the database schema and software design. As they waded into the source code, the team realized that both the software architecture and the database schema were dependent upon the twelve existing templates. In order to add a new template to the system, new authoring and publishing code would have to be written specifically for that template and new database tables would have to be designed and implemented. Clearly the system was not set up to allow for template set extensibility. The team realized at that point that the "Port and Refactor" approach would not work. Instead, the team needed to redesign and rebuild the software architecture and database schema to make them both template independent.

Design Decisions

Having committed to a much more extensive re-write of the application, the team needed to decide on a basic software architecture and a development platform. To support the need for a template-independent architecture, the team developed a component-based approach. A component in this sense is a basic data element of a presentation screen. Thus an image or a block of text would be a component. If the system could be built so that it knew how to deal with all the major components used in the core template set, it would also support new templates that utilized the same components in different configurations without requiring any new coding in the authoring system or the database.

The selection of a development platform, on the other hand, needed to reflect the goal of creating a platform-independent application. The development platform includes the language in which the software is coded as well as any pre-existing applications or frameworks upon which the software will depend. To help in making this decision, the team created a matrix of possible platforms, listing pros and cons for each suggestion, including issues of expertise, ease of use, ease of learning, stability and scalability, and the ability to leverage work already available in the suggested platform.

At the time, the "highest risk" component was Macromedia Generator, used to compile images with associated metadata and publishing these in Flash (.swf) format. The Generator product was discontinued several years ago and is no longer commercially available. Without a functional replacement, it would not be possible to publish the Flash-formatted media that are essential to a Pachyderm presentation. The search for replacements led to the jGenerator

project - an open-source Java library designed to be a drop-in replacement for Macromedia Generator. This jGenerator library was identified as the only reliably functioning Generator replacement.

One of the requirements for Pachyderm 2.0 involved the ability to demonstrate database independence. The system needed to be able to connect to a variety of database servers, including Microsoft SQL Server, MySQL, Filemaker, and potentially many others that had yet to be identified. The primary software development platform that offered this level of database independence and about which the development team as a whole had a significant amount of institutional knowledge was Apple's WebObjects application server. WebObjects is a java-based set of frameworks and applications, including a database management framework called "Enterprise Objects Framework" (EOF). EOF would be able to take care of the "heavy lifting" of connecting to various databases, and would allow the team to focus more on application-level development.

The architecture of the WebObjects frameworks offered a high degree of modularity which could be copied when developing the Pachyderm 2.0 application. The team could build a set of shared functionality into a framework, which could be accessed by any WebObjects application deployed on the server. This could let other groups easily build tools to integrate with Pachyderm 2.0. The fact that WebObjects can be deployed on a variety of platforms, including MacOSX, Windows 2000 Server, Solaris and Linux would satisfy the requirement that the system not be tied to any single server vendor.

APOLLO

Concurrent with the start of Pachyderm 2.0 development, The University of Calgary was working on the successor to the CAREO learning object repository, code-named "APOLLO." APOLLO, built on top of WebObjects, was designed to be an extremely flexible set of frameworks and applications that would allow the development of a wide variety of different but related functionality, specifically aimed at enabling people to publish, locate, and reuse learning objects.

The team realized that there was a strong overlap in requirements between Pachyderm 2.0 and APOLLO, and in the summer of 2004, decided to base Pachyderm 2.0 on the APOLLO frameworks and concepts. This would allow the team to further focus the development effort on the features directly required to support the presentation authoring application, while working with the APOLLO developers to further enhance its functionality.

One very powerful feature of APOLLO is the ability to build small components and plug-ins that can be integrated dynamically into a running application, allowing the addition of customized functionality without modifying an entire application or framework.

APOLLO also provides a set of services for transforming media from one format to another, which could be leveraged as part of the Pachyderm presentation publication process. As part of that process, the system would need to be able to resize images, convert them into a Flash-friendly format, and compile them into Flash-wrapped media for use in the final published presentation. This process was a strong limiting factor in the scalability of Pachyderm 1.0, so the ability to generate transformed media on the fly, and on demand, would allow the system to support a potentially unlimited library of media assets.

Open Source
The Pachyderm 2.0 project was strongly oriented toward open source development. It had been a central theme in the project from the proposal stage onward, in an effort to ensure collaboration from interested groups. The idea was for development to be as open as possible and to result in a strong set of source code that could be used and extended by others. The source code and developer's guide for Pachyderm 2.0 will be released to the public in the fall of 2005 under an Open Source Initiative (OSI) approved license.

Pachyderm Presentation Screens
Macromedia Flash is a popular application because it enables professional media designers to easily create interactive and dynamic online experiences. Flash Player has been adopted in web communities because of its ability to produce small files facilitating the streaming of rich media content. Other electronic devices such as Web TV, Pocket PC and some cell phones are supporting Flash files today. The secret of success is the small sized player (less than 500 K for Flash Player 7). The new Pachyderm presentation templates are completely designed in Flash MX 2004 and users are expected to have Flash Player 7.0 or above.

Flash-based web presentations have many advantages over traditional html-based web contents. The most valued benefit of using Flash to author web content is that it is browser- and platform-independent. The same Flash movie will look exactly the same cross-platform (Windows, Mac OS, and Linux) and cross-browser (Netscape, Internet Explorer, FireFox and Safari). This is very important for the online learning communities since educators and learners often have a very diverse selection of hardware and software. However, Flash does have its weaknesses. Developers need to be aware of the version issue while developing Flash files. Similar to most software, Flash is not forward compatible (one can not run Flash 7 files on Flash 6 player). Another concern is that major search engines are still having trouble in indexing the content inside Flash based content packages. This makes the meta-tagging procedure and XML content aggregation so important while creating learning objects in Flash. This is a major reason for the Flash upgrade in Pachyderm 2.0.

Pachyderm 1.0 includes twelve Flash templates created in Flash version 4.0. The templates are really Flash shells that contain the layout, graphic design and interaction coding (with Action Script). The content (images and text) that makes each presentation screen unique is provided to the template in a variable based text file (.ini file). The text files are generated in ASCII standard encoding and meta-data was not adopted within the design. This means the authored learning objects are language-dependent and reference-limited. With additional considerations of forward design requirements such as templates modularization and learning object standards, the Pachyderm team has decided to integrate the XML referencing model into Pachyderm 2.0.

In the new implementation, Flash MX technology is being used. Content aggregations on the presentation end are handled by XML files. Presentations are constructed by templates, and each template references a unique XML file. The Flash templates parse the XML files in order to fetch content into the presentation. While serving the purpose of describing the content, these XML files are also being used for assets searching, template referencing and presentation meta-tagging purposes.

With the up-to-date Flash support, the Pachyderm 2.0 development team can fully integrate XML-based learning standards (i.e., SCORM) with the learning objects authored using Pachyderm 2.0. The presentation will process the IMSmanifest XML file to lay out the navigation through the authored templates. IMSmanifest XML is the key file for SCORM 2.0 compliant learning objects. From this file, a navigational presentation map can be generated. This key file also performs as a "messenger" to connect the learning objects with the learning management system (LMS), which will allow institutions to fully integrate Pachyderm as a learning object authoring platform with their existing LMS.

The next step in the Flash development of Pachyderm 2.0 is to implement the Model View Controller framework and componentized screen elements. This will allow other Flash designers/developers to generate new Pachyderm templates using pre-defined Flash components and not to worry about action scripting. A further step will be to create a tool that allows users to develop custom templates whether or not they know Flash. A truly WYSIWYG Flash designing interface would empower subject matter experts to directly author the templates that suit their needs.

Collaboration

The Pachyderm development team is composed of over a dozen members located throughout the United States and Canada. Large projects even within the same institution can be challenging, but distance can present significant obstacles to intercommunication and collaboration, and introduces risks to the project. Early in the project, the development team had to establish inexpensive efficient means of collaboration that would reduce the risks.

Some significant risks to the project accentuated by distance and distribution were:

a) Wasted or redundant efforts from team members in differing location due to inability to coordinate tasks, and/or inability to coordinate concurrent development efforts.
b) Inefficient use of the resources available to the project due to a lack of understanding of the teams' particular strengths, inability to use each member at his or her maximum potential, a lack of knowledge of what resources may be available to the project (personnel or software), and/or a lack of cohesion or commonality between the members of the development team.
c) Failing to meet expectations of the project due to a mismatch between what is produced by the development team and what is defined in the system requirements, and/or what is defined in the system requirements and what is expected by the system users.

The team felt confident that the latter risk (c) had been adequately addressed by the requirements gathering process used for the project. Any changes to be made to the requirements would be entered into the tracker application, and would be part of the living requirements document, easily reviewed by the development team online.

In the selection of a development architecture, the team was confident that several of the risks detailed in (b) were covered. In choosing the development platform, the decision was reached to use WebObjects and the APOLLO framework for a number of salient reasons, many detailed in previous parts of the paper, but many important factors stand out as they relate to the issues of distance collaboration. With the choice of WebObjects and APOLLO, the team gained a high level of expertise from a majority of full-time development engineering

resources. WebObjects was heavily used in the University of Calgary, and APOLLO was a framework created by one of the staff engineers there. The other team members had no familiarity with WebObjects, but did have significant Java and application server experience to give an adequate starting point for learning about WebObjects. Also, with the use of WebObjects and APOLLO, the team had at its disposal a great deal of frameworks available for use in the application that would otherwise have had to be written from scratch for other development architectures. The choice allowed the team to speed the level of development, and get a lot of infrastructure for free, allowing a greater focus on the application logic itself.

After deciding on the development platform, the team realized that the first risk category concerning waste due to disorganization and poor communication from (a) remained unaddressed, as well as issues of team cohesion and inability to use each member at their maximum potential from (b). The method used to address the remaining risk factor consisted of four factors; collaboration tools, communication process, development methodologies, and scheduled face-to-face meetings.

Collaboration Tools
Collaboration tools were absolutely essential to the success of the Pachyderm project, and what follows is a list of the different applications employed on the project.

Subversion (Version Control) – http://subversion.tigris.org/
The mere fact that the Pachyderm project involved more than one developer working on the same code base necessitated some sort of version control. Considering that the developers were distributed across the continent, an easily accessed and resilient version control system was required. Although there was a significant degree of institutional knowledge in CVS in the group, Subversion was chosen as the software versioning system for several salient reasons. First and foremost, it is open source and free to use, and CVS itself has a link to it on its home page. It is considered the heir apparent to CVS by many, and many developers have moved to supporting Subversion over CVS. Subversion appeared to be very well suited to a distributed environment, as it used secure http for its communications, tied well into relevant applications such as XCode (the Macintosh IDE being used for the project), and had a web interface for easy viewing of subversion content. While CVS remains a very strong version control system, Subversion appeared to be an overall better fit for the project.

VNC (Virtual Network Computing) – http://www.realvnc.com
Distance pair/team programming techniques required some means for developers to share a common workspace. VNC became an early adoption to the project, because it is open source under the GNU GPL, free to use, and very good. There are actually many clients and servers available that use the VNC protocol, and these are available on most platforms. The team made use of OSXvnc as the server software, and Chicken of the VNC as the client on the Mac. Tight-VNC was used for windows.

Instant Messaging – AIM, iChat, MSN, etc.
The developers employed some sort of instant messaging on a daily basis from the very start of the project, as it is an inexpensive (free) form of direct conversational communication that allows for multi-member chat, and keeps a transcript log of any conversations. File sharing is also simple through an instant messenger client. The team also used audio/video chat (via iChat using the iSight camera) during team programming sessions, which freed up the hands

for coding and allowed for easy communication. In many respects, it was almost as good as being in the same room.

SubEthaEdit (Collaborative file editing) – http://www.codingmonkeys.de/subethaedit
Tools such as SubEthaEdit allowed for joint editing of the same document without having to share an entire desktop, and also tracked contributions/changes from different members in separate colors. Along with VNC, this became one of the primary tools for pair programming.

Wiki (online collaborative documentation) – http://en.wikipedia.org/wiki/
The team maintained a wiki for keeping group institutional knowledge current and available/updateable by all members. Developers would write collaborative documents on the wiki when the timing between participants was asynchronous, and when the platforms of the participants were varied enough that a platform-independent solution was needed.

Macromedia Breeze (collaboration app) – http://www.macromedia.com/software/breeze/
Other collaboration tools were investigated, such as Macromedia Breeze, which did provide for a rich environment for demonstrations, and with different intercommunications widgets available to the user. Breeze was not an ideal match for the team's development style, and was not used often.

Typepad (blog software) – http://www.typepad.com
The Pachyderm project team put up a blog that has been used to communicate news to the project partners, and to act as a centralized repository for project documents. The development team makes use of this to announce larger successes and milestones. It is a way to keep the customer base informed as to progress, and to help keep requirements and development in line with one another.

Communication process
As crucial as the tools being employed by the development team are the methods and habits of communication between the members of the team. The success of the project depends on how faithfully, regularly, honestly and easily these communications occur. The developers used the following habits and methods for keeping in synch over vast distances.

Online Live Communication
Instant messenger became a lifeline for team members to keep in touch with one another, to work together and share pertinent information. Audio and video chat helped immensely in freeing up the hands to work while still communicating. The developers had daily, often continuous contact with one another through these means.

Regular Phone Conferences
Starting bi-weekly, then progressing to weekly and then twice a week, the development team has been holding phone conferences to help keep on track. An online free conference service was used with the only cost being the toll charge to the dial-in number (http://www.freeconference.com).

Online Asynchronous Communication
Email was also heavily used for intra-communication, but not as extensively as instant messenger. Email was reserved for communications with the whole team, or to distribute

documents to a large group at once. The team also used the Pachyderm blog, which helped communicate successes to the whole team and to the outside world.

Development methodologies

On this project, the Pachyderm team members have adopted some methodologies from Extreme Programming, such as "Team" or "Pair" programming, which have been extremely valuable in producing high quality code at a faster pace than normally would have been possible. Using VNC and iChat (or while in person), several team members would join in to a group programming session, with one member controlling the keyboard, and the other members watching, giving syntactical and logic corrections and thinking strategically about the direction of the code. The affect of this methodology was multifold – first, it allowed for the members of the development team with a stronger sense of the application frameworks being used to disseminate that knowledge to other members, in a practical fashion that moved the code forward, instead of taking time out of the development schedule. Second, it built in a process of team code review, since everyone was writing the code at the same time. Furthermore, it blurred the line of ownership of any particular piece of code, allowing for more flexibility on code changes and updates, and helping keep a sense of team cohesion. As a result of team programming, everyone felt an equal responsibility in the code, and felt an equally important member of the team.

Scheduled face-to-face meetings

Of course, any long-distance project necessarily must contain a component of face-to-face time. The group came together on a larger scale at project meetings, and during conferences at which many members were in attendance. In addition, team members would physically spend periods of time (typically a week at a time, every other month) working on-location with other members of the team. This also allowed time after work for social bonding, and for a more productive team-development environment. Sometimes nothing quite beats a bunch of people in the same room with a white board. The team found that productivity would soar during one of these extended visits, and would continue to remain high for weeks afterwards. They were an integral part of the development process.

Future Plans

When planning began for Pachyderm 2.0 in October of 2003, the team came up with a lofty vision of what the authoring tool could be. The team has spent the last year and a half carefully pruning expectations to define a goal that is both desirable and achievable. The big ideas, however, have not gone away. They give the project directions in which to grow. The project is actively seeking funding to continue development of the larger vision. Once Pachyderm 2.0 is released to the public, the team expects that user feedback will guide the development of additional features that the community truly values.

References

Meggs, Philip B. (1992). Type and Image: The Language of Graphic Design, New York: John Wiley & Sons, Inc., 153.

Spolsky, J. (2000) Things You Should Never Do, Part I. consulted January 28th, 2005. http://www.joelonsoftware.com/articles/fog0000000069.html