

Jim Zoetewey
CS 693
Dr. Hans Dulimarta
9/7/2004

Distributed Content Management System for a Web Site

Masters' Project Description:

The Basic Idea:

Most organizations operate web sites these days. Many times the people in these organizations are not particularly knowledgeable about graphic design, html or even comfortable with computers. Nonetheless, the organization has a web page. In many situations, the content of this web page must be continually changed and it may be that many people will have a say in the content of a particular document.

This application is aimed at small to medium sized organizations that want many people in the organization to have the ability to modify the contents of the web pages, but don't want them to use Front Page, GoLive, Dreamweaver (or some similar tool) and risk accidentally destroying the design of the page. Ideally, this would allow any member of the organization to make changes without having to in anyway contact the IT people in the organization.

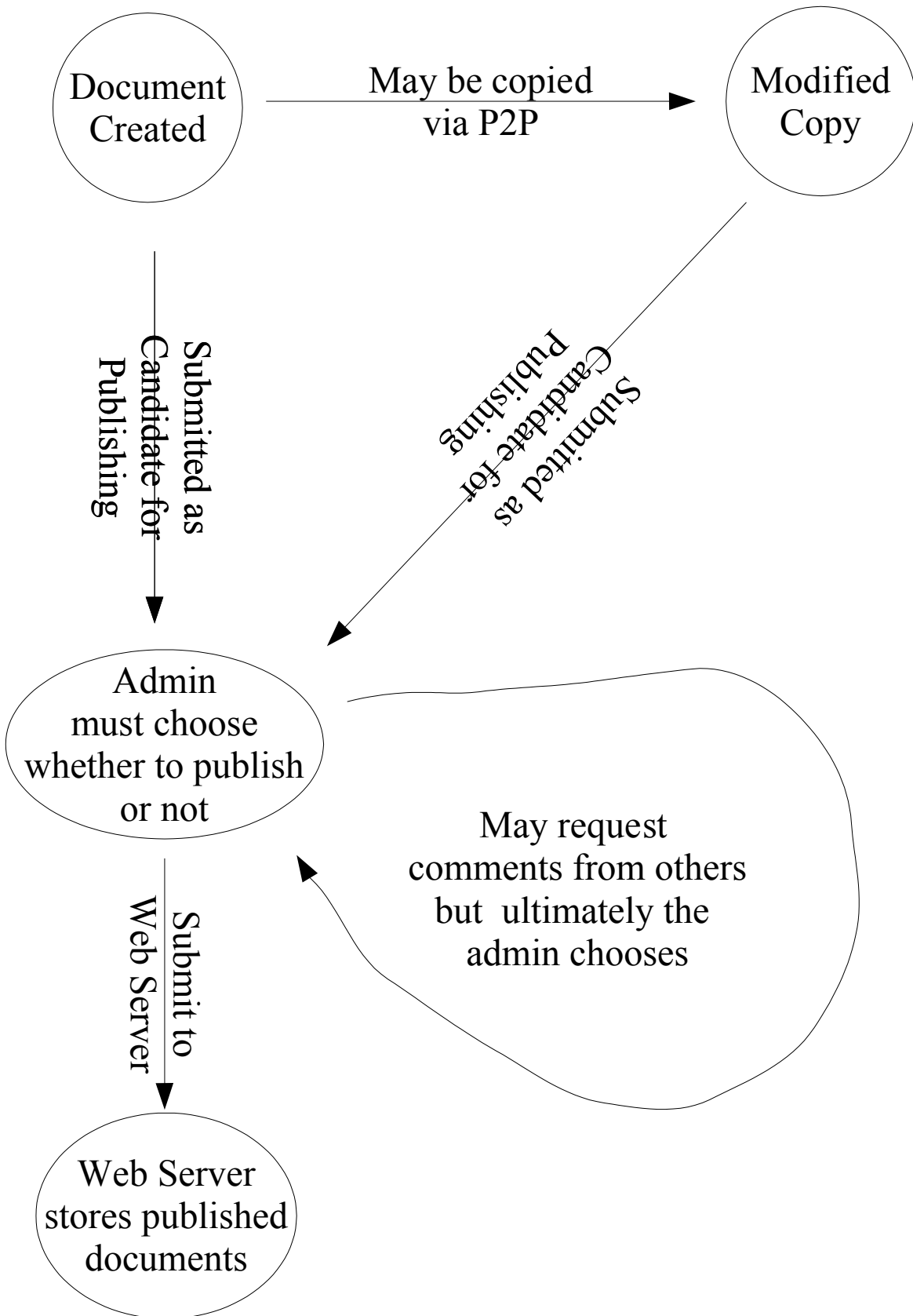
The Core Problem:

Organizations need to update their websites regularly, but also need to keep control over what appears on the website. People need a tool that will allow people to update, but will also put some people in control of what information goes out. This way embarrassing mistakes may be avoided.

This means that there need to be individuals (admins) with the power to update the website. Everyone else must submit their updates, changes, and new pages to them. Only admins have the ability to change a document from being a candidate for being on the website to an actual page on the website. They can choose to upload the document to the web server from the cms server.

Because this program is targeted toward small organizations, there is no need to build a lengthy or complicated document approval process into the program itself. It may be useful, however, to build in some communication between individuals. Thus, while one member of a department may be all that's needed to change a document's state from candidate to published, that person in the department should have the ability to send messages to others, telling them to inspect a particular document and then to receive their comments.

Document Approval Process:



Document Approval Process:

From a user's perspective, a document has three states. Initially, it is just the user's document and is only accessible by downloading it from the user's computer. It may be worth including some sort of privacy setting so that people can choose to keep certain documents unseen until they're ready to have others inspect them. Nonetheless, for the most part, any document that a person is working on can be downloaded at any time.

The second state is that of candidate. At this point, it has been uploaded to the cms server as a candidate for publishing. At this point, the admin responsible for setting things to a publishable state will examine the document. Other people that that person (may or may not) invite to view the document can as well. Also, the person who submitted the document can remove it from consideration. In addition, other people can download the document, make changes, and resubmit it under their own name.

The third state is published. At this point, the document is on the web server. A record of this is kept separately so that anyone can make modifications to it and submit it for consideration to be published.

Thus, at no point is anyone editing the exact same document. They are always editing their own copy of the document, but they can at any time make a copy of a document (in whatever state) and modify it.

Document Versions:

It may be necessary to be similar to CVS in terms of keeping track of different versions of the website's content. Written content is different from writing code in that code can fail to work as a result of factors that change in the surrounding code, making strict version tracking, branching, and merging more important for programming than it likely is for website content.

Though the program may store versions by number, much as CVS does (1.1, 1.2, 1.3...), it seems likely that it would have to display it differently to the (likely technically challenged user). Perhaps there would only be a future branch, a current branch, and a past branch. Future would be for things that are not going to be displayed at present. Current would be for modifying what's on the website right now. Past would be the last version of the page that admin's okayed prior to the present one.

Admin's would be able to create new branches, delete branches and merge branches. They would also be able to allow or prevent any changes to be made to particular branches.

Technologies:

Java: Though Java can require a small amount of porting to be completely compatible with some operating systems, it does give programmers a large head start in making their programs portable. This is the language I plan to use for the client/server program.

JXTA: An XML-based protocol for peer to peer connections, allowing people to pass messages, files, and other forms of information. Originally created to be used with Java, JXTA has been ported to several other languages. In this context, JXTA would be used to abstract the process of sharing files and messages.

XML: A markup language that allows people to create their own formats for passing information. In the context of this program, XML would be used to describe the type of information in the various sections of the document. This would allow a server

based program to parse the useful information after the document is submitted to the web page. Alternately (with the addition of appropriate stylesheet information), the document might be used as a web page.

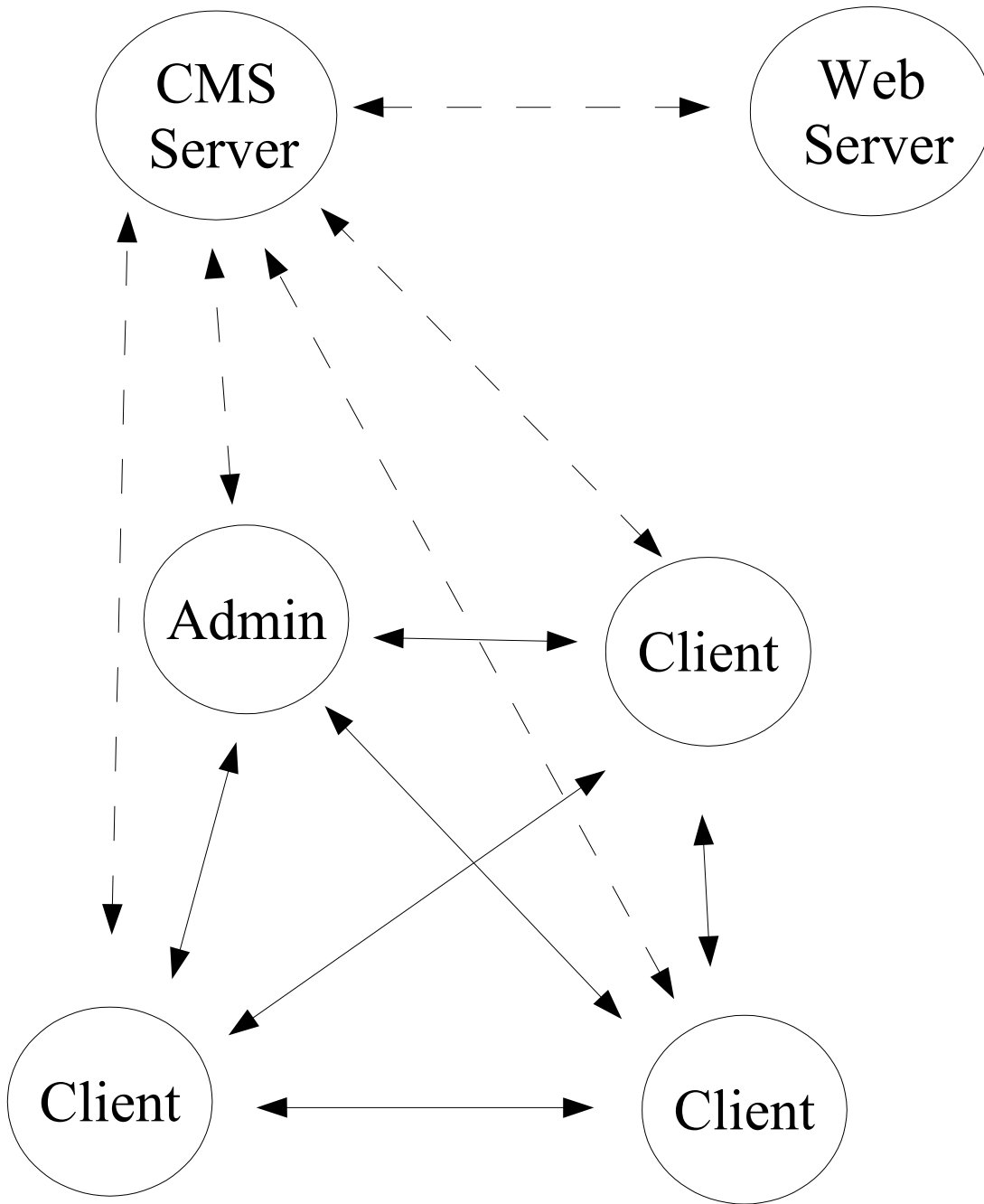
The Network Architecture:

This program has a mixed architecture. At different points, it requires peer to peer and client/server interaction. It requires peer to peer interaction between people who are attempting to create documents prior to those documents becoming candidates for publishing or published documents.

Once documents become either candidates or published, they are kept on the cms server. This may or may not be on the same computer as the web server. When a person decides to publish a document, it is moved to the web server if necessary.

The cms server also holds records related to user privileges and metadata about the documents one can create, distributing this information as needed.

Network Architecture Diagram:



◄————► Peer to Peer connection
◄ - - - ► Client/Server Connection

Core Program Functionality:

In theory, all people should need for this program to fill its function is some sort of ability to pass files between users, upload to the server, and a system for deciding which documents should be passed and which should be revised.

In practice, it will be more convenient for the users to do as much as possible within the program without having to resort to pulling out another. For example, a surprising number of users have no idea what ftp is much less how to do so. Best to make it a part of the cms system, allowing them to remain as ignorant as they want to be. Thus it might be useful to allow comments about documents to be passed on along with documents.

Nonetheless, some requirements are more important than others. I've listed the bare minimum in this section.

Ability to pass XML files via a peer to peer network: The peer to peer aspect of things would be handled with the JXTA protocol, but the program must also be able to encode and decode XML. There are multiple XML parsers. This program will use the one that turns out to be the most convenient.

File Management: The program must keep track of which page (or section) of a web site the document corresponds to (or whether it is new). Each document must also be categorized by whether it is the published document, a candidate for publishing, or currently unfinished. There must also be some sort of system to easily indicate which files have been accepted for publishing and which rejected.

Graphic User Interface: Bearing in mind that the target audience will be non-technical people, the program must have some sort of GUI. Expecting someone without technical experience to use a command line interface is unreasonable.

Uploading: The users must be able to move files out of the content management system up to the web hosting provider. At the very least, this means they should have access to FTP. Alternately, the client program might run on the server if the hosting provider will allow it.

User Management: There must be some sort of user management. Not everyone in an organization should be allowed to edit a particular file. In many cases, someone else in an organization should be able to look over a file before it goes out. Thus, every user must be identified to the network. In addition, every user must have a list of permissions, indicating whether they can write, edit, delete, or upload a particular file. In addition, admins must be given the ability to assign privileges. Finally, this means that some sort of user level security system must exist in order to stop users from accessing other people's accounts. In addition, this information must be securely passed (or at least accessed) throughout the network.

Optional Functionality:

Dynamic XML Formats: Some people might need to use more, less or simply different information than this client allows. Allowing people to create their own XML formats would allow people to modify the program to their needs. Unfortunately, it would also require a dynamic user interface. This would be hard to deliver within one semester.

HTML Layout: This program's purpose is not to layout html pages. Ideally what would happen is that all the layout would occur server-side based on the XML tags. Nonetheless, people will want to include links. People may want to create lists of one kind or another. Some people may know enough html that they could reasonable layout a

page. Similarly, they might want to add emphasis to particular words with bold or italics.

As such, it would be useful to allow html tags for those who know them. Ideally, it would also be good to allow some sort of wysiwyg layout capability. Unfortunately, it seems likely that that would take far too long to implement.

Tracking Changes: In an ideal world, it would be good to allow people to make or discuss possible changes within the document itself. The changes might be allowed / disallowed and then erased before the document gets published. This might also take far too long to implement, but would be useful when many people need to edit the same document.

Tracking Documents: It might be useful to graphically or textually represent the heritage of a document so that one could follow various versions of a document easily. This seems likely to add extensive infrastructure and might be excessive for the audience's needs.

Middleware (DCMS):

Various parts of the program will have different architectures. All will make use of a common middleware built using Java and JXTA. This middleware (called DCMS for now) handles the following tasks:

- Identifying who is online and what files are accessible to all.

- Moving files from the CMS server to the webserver.

- Moving files from person to person or to the CMS server..

- Handling who has permissions to create/modify content.

- Tracking documents and their versions.

- Encoding/decoding documents into/from XML.

Advantages/Disadvantages and Alternative Options:

Why use p2p java clients when you could use php/mysql directly on the web server?

First of all, web servers are not available to all people at all times. For example, a person might want to take their laptop on a plane and edit documents that happen to be on their computer. They have no internet access at that point, but nonetheless, they might need to work. If they have to contact a web server to do it, they have little choice but wait for internet access.

Second, this potentially allows more control over posting than most services in which people directly edit the document on the server. Typically with those services, the person edits the document and it changes with their edits. In this situation, the document's state might ultimately be controlled by one person. This person could then decide whether to upload it to the server. Many organizations like this level of control.

Thirdly, locating/modifying the content management system directly on a server simply wouldn't be a distributed system. I'd like to do something with distributed systems for my project and using php/mysql would not be distributed.

Why use JXTA when SOAP, sockets or even FTP might suffice?

SOAP, though a very popular protocol for distributed systems, seems more complex than JXTA. JXTA, by contrast was created to do peer to peer. So, though SOAP is more popular and more in demand, I would have to make SOAP do something that it can do, but wasn't specifically created for. JXTA's reason for existence is to do peer to

peer distributed services.

Using FTP could also potentially work, but, like SOAP, FTP really isn't created to be used for doing peer to peer web services. It could be used for such, but it really would require many workarounds to cover the inadequacies of the tool.

If JXTA turns out to be more complicated to use than simply implementing the whole thing in sockets, I probably will switch to sockets. I'm hoping I don't have to, however.

Why Use XML?

XML's good point is that it can be used as part of any data system that includes a parser for xml. This means that you are not limited by programming language, operating system, or database.

This works with this system because the idea for this system is that the content management occurs within the collection of clients. The system on the website will be a different thing. It may or may not be use Java. It might use perl, php, ASP, VB or even Lisp. Whatever the case, the web server will be able to process XML file and put it into a database or even use directly.