

Chandler for Higher Education: Requirements and Recommendations

May 2, 2003

I. EXECUTIVE SUMMARY.....	3
II. CHANDLER – THE PRODUCT	5
CHANDLER – THE VISION.....	5
CHANDLER PRODUCT ROADMAP.....	6
CALENDAR	6
EMAIL	7
SHARING AND COLLABORATION	7
CHANDLER AS PLATFORM.....	7
SECURITY	7
CENTRALIZED DEPLOYMENT ISSUES AND USAGE SCENARIOS	8
OPEN SOURCE AND STANDARDS BASED.....	9
III. AN OVERVIEW OF THE CHANDLER ARCHITECTURE	10
INTRODUCTION.....	10
VIEWERS	11
DATA MODEL API	12
DATA SOURCES	12
IV. DETAILED REQUIREMENTS AND RECOMMENDATIONS.....	14
CALENDAR PROPOSAL FOR WESTWOOD.....	14
WESTWOOD CALENDAR DEPLOYMENT SCENARIOS	15
HIGHLIGHTED CALENDAR FEATURES FOR WESTWOOD.....	18
EMAIL PROPOSAL FOR WESTWOOD.....	21
CONTACTS AND LDAP INTEROPERABILITY	28
SECURITY	31
V. CENTRALIZED DEPLOYMENT ISSUES AND USAGE SCENARIOS	37
INTRODUCTION.....	37
CHANDLER CENTRALIZED DEPLOYMENT ISSUES	37
HIGHER EDUCATION USAGE SCENARIOS.....	39
CONCLUSION.....	46
VI. CSG MEMBER VOLUNTEERS.....	47

I. Executive Summary

The Open Source Applications Foundation (“OSAF”) is developing a next generation, open source personal information manager, code-named Chandler. From two all-day meetings in December 2002 and January 2003, the Common Solutions Group (“CSG”) concluded that Chandler could be a compelling application for higher education.

OSAF’s original development scenario for Chandler targeted the needs of individuals and small organizations rather than the needs of higher education with its accompanying complex mix of existing infrastructure. Encouraged by the response from CSG, the Andrew W. Mellon Foundation awarded a grant to OSAF to determine the incremental requirements and to develop a set of recommendations for a higher education version of Chandler.

To create this plan, OSAF worked with a number of CSG members including both CIOs and technical domain specialists¹. These representatives participated in working groups based on areas of expertise, providing key input into both requirements and recommendations. The fruits of this collaboration are presented in this document.

Canoga – our 1.0 release – will be the first stable and robust release of Chandler. *Canoga* will target mainly info-centric individuals and decentralized groups, and provide core email, small workgroup calendaring, contacts and tasks functionality within a strong information management foundation. The higher education version of Chandler, code-named *Westwood*, will be built on top of *Canoga*. **We believe the key incremental requirements and recommendations for *Westwood* are:**

Support nomadic usage and central repositories

A single user might need to access her information from many different machines throughout the day. Some of the computers may be personal machines; others may be public kiosks or lab machines, where personal data is deliberately not preserved.

Westwood clients will be able to access remote centralized Chandler repositories using our Repository Access Protocol (RAP). Information on the central repository can be synchronized with personal machines for offline or disconnected usage or simply faster access.

To address scalability issues at a university level, OSAF will deliver a pilot version of a centralized server with *Westwood*. This server will provide scalable, high-performance servers with administrative control, reliability and robustness not offered in the peer-to-peer configuration. As the product matures, future versions will provide additional server robustness and specialized administration tools to ensure successful campus-wide installations.

Standards based Calendar Access Protocol (CAP) client

In addition to decentralized workgroup calendar solutions, universities also need centrally managed calendars. *Westwood* will be designed as a CAP client to ensure interoperability with future versions of Calendar servers such as Oracle Calendar (Corporate Time), Sun One, and open source calendar servers that will support the emerging CAP standard. By embracing an open industry standard, the *Westwood* client will provide flexibility and reduce ‘lock-in’ to single vendor solutions. Chandler will also include designation/ delegation and resource scheduling features to provide a full client calendaring solution for universities.

¹ The list of CSG members who volunteered for this project and the process we followed are listed in Section VI.

Full interoperability with existing standards based infrastructure

In addition to POP functionality, Chandler will be a well-behaved IMAP client. In particular, Chandler will not try to download all messages from all folders at the start of every session. Instead, Chandler will use heuristics and user overrides to determine when, and which messages to download.

Chandler will allow users to continue using other IMAP clients if they choose. Chandler meta-data such as annotations and categorizations will not be reflected back to the IMAP server. Instead, we use a technique we term *data compositing* to provide richer information management capabilities for Chandler while ensuring IMAP server compatibility with other email clients. We also intend to use data compositing to integrate LDAP and CAP data into Chandler.

Chandler will be able to synchronize contact and group information with LDAP directories. We will also support LDAP for authorization and access control. Chandler will support schema transformations to accommodate the many different LDAP configurations in universities.

Higher education institutions and OSAF share a respect for strong adherence to technical standards. We are pragmatically committed to supporting important emerging standards such as CAP as a calendar protocol, and Shibboleth as a federated policy framework

Robust security framework

We will create a security framework for data sharing based on existing standards and best practices. The foundation of the architecture will be TLS for transport layer security, and SASL for authentication. This architecture guarantees that no password will ever be transmitted in the clear.

To ensure Chandler is a trustworthy application, we will perform security audits for major design decisions and code reviews. We will support pluggable security extensions and allow numerous policies to be configurable so that each institution can determine their own safety and convenience trade-offs. Security considerations will always be designed into user interface implementations, communicated clearly where necessary, and transparent to the user where achievable. This will ensure that security is not just theoretical, but practiced in every day usage.

Conclusion

Openness is critical to the success of the Chandler project. We rely on it to develop the best possible solutions, to allow institutions to modify Chandler to better meet their needs and to provide users with ultimate control over their destiny. Chandler is available under open source terms and free of charge. Our development is also open: we invite interested parties to get involved and help Chandler mature.

We believe this plan complements the excitement of the original Chandler vision with a set of down-to-earth recommendations for meeting the particular needs of the CSG members.

We hope this plan will result in a commitment by CSG members to deploy Westwood. If so, this document will form the basis for a subsequent funding request to the Andrew W. Mellon Foundation for the implementation of a higher education version of Chandler.

II. Chandler – The Product

Chandler – The Vision

The Chandler vision combines broad, long-term possibilities for personal information management with an initial focus on the functionality most needed by today's info-centric users: shared calendar and contact information, email, and task management. Chandler will be both an end user application itself, as well as a platform for developing other information management applications. It is currently under development and will run on Windows, Mac, and Linux-based PC's.

With Chandler, users will be able to organize diverse kinds of information for their own convenience – not the computer's convenience. Chandler will have a rich ability not only to associate and interconnect items, but also to gather and collect related items in a single place, creating a context sensitive 'view' of many types of data. Users will be able to mix-and-match calendar data, email, mailing lists, instant messages, appointments, contacts, tasks, free-form notes, weblogs, web pages, documents, spreadsheets, slide shows, bookmarks, photos, and so on (and on). This approach also allows the same data to appear in multiple views such as 'project', 'things to do today', 'my tasks', etc. at the same time. This is a very different approach from that of today's common PIMs, which require for example, that an email message belong to only one folder, and only in the email section.

Chandler pulls information from various data sources to provide the most suitable context or 'view' for the relevant task at hand. Data sources can come from disparate systems such as IMAP, LDAP or web servers or even multiple instances of the same type of data sources e.g. an instance of an IMAP server containing personal email and an instance of an IMAP server storing USENET information.

Chandler will help keep track of numerous concurrent, ongoing activities, which requires the ability to collect just the right sets of related items. For example, it will be easy to retrieve a list of calendar events for a given day, together with the tasks you are expecting other people to do in preparation for these events, and all the emails you have sent and received grouped by these events.

Choice is central to the Chandler vision. Chandler is open source / free software so that developers and users can always choose to make Chandler better, to participate in its general development, to improve its integration into particular settings, and to control their ultimate experience. Chandler will be extensible, modular and customizable so that those using and deploying Chandler can choose how to make the Chandler releases function most effectively for them.

Chandler Product Roadmap

Chandler Releases and naming conventions



We anticipate the Chandler product to undergo four distinct phases of development:

1. In the **Canoga Early Developer Releases** (Chandler 0.2-0.5 release), we will focus on platform and infrastructure, designing and testing out our compelling innovative capabilities, putting in place a modular architecture and providing basic elements of end-user applications to validate our platform.
2. In the **Canoga End User Releases** (Chandler 0.5-1.0), we will target the **info-centric early adapters** by building strong info-management capabilities, power email features, basic calendar and contacts functionality with sharing and collaboration as an integral piece of the product.
3. In the **Westwood Releases**, we will target **higher education users** by providing rich, standards-based calendar capabilities, interoperability with existing infrastructure, central server support for nomadic access, and other university-specific requirements.

The Westwood release will be the first release we believe will be satisfactory for deployment within universities. It will have excellent workgroup support but only rudimentary central server support and maturity. Central server features will not be ready for campus-wide usage but will be ready for pilot test and smaller workgroup deployments.

After Westwood, we will work on issues and functionality such as server robustness, stronger and more specialized administration tools, and look into providing a CAP-compliant calendar server and support for other important emerging standards.

4. For the **Pasadena and Future Releases** (Chandler 2.0+), we will focus on building Chandler to support third parties who will market the products for the mainstream, pragmatic audience. This will involve providing features that will help create and sustain a value chain for Chandler such as interoperability, migration, extensible framework, distribution support, technical support, and so on, as well as polishing and pruning the existing features of Chandler.

Calendar

Chandler's Calendar will provide individual and group calendaring features. In addition to the core functionality one expects, Chandler's calendar will include innovative features which reflect our vision of combining and inter-relating data of all types for the users' convenience.

Chandler will enable people to easily share rich calendar information on an ad-hoc basis. In particular, today's users can only share and use free/busy time with other users of expensive, centrally managed, proprietary server-based products. Chandler's peer-to-peer calendaring system will enable any subgroup of Chandler users – a small business, a study group, a non-profit's board, or even people planning a surprise birthday party – to efficiently schedule meetings, browse the calendars of others, and see overlays of multiple calendars simultaneously.

For Westwood, we intend Chandler to be a standards-based CAP client. Besides decentralized workgroup solutions, universities need centrally managed calendars. By being a CAP client, we will ensure that Chandler interoperates with future versions of Calendar servers such as Oracle Calendar (Corporate Time), Sun One, etc., whose vendors have officially stated their support for the emerging CAP standard as well as open source CAP servers. Westwood will also include designation and resource scheduling features to provide a full calendaring client solution for universities.

Beyond Westwood, we would like Chandler to be a full CAP server; first, as a means to communicate with other CAP servers and later, as a stand-alone Calendar server. A detailed description of planned Calendar features is included on <http://wiki.osafoundation.org/bin/view/Main/CSGreferenceDOCS>

Email

Chandler's Email parcel will support both POP and IMAP in well-behaved ways. Chandler will not interfere with other clients' use of the IMAP or POP servers, and will be careful about which messages from which folders to download at what time.

While Chandler will happily co-exist with other email clients, we believe that Chandler's novel email features will be compelling enough that many people will switch to Chandler for email. Chandler will provide a number of features specifically tailored to allow the user to manage large volumes of email more efficiently and effectively. Besides reducing the number of keystrokes and mouse actions needed to process messages, Chandler will have easy-to-use tools to assist users in organizing and prioritizing their messages.

In particular, we've recognized that people use email as a form of very lightweight "to-do" list, leaving messages in a very visible place until they have been read, acted on, and/or replied to. Chandler will have very explicit features to facilitate the "task management" aspects of email, including powerful filters, views, threading, integration with other data types, and a task-centered UI.

Sharing and Collaboration

Sharing and collaboration infrastructure will be built into Chandler. For example, Chandler already includes instant messaging capabilities. Today, Chandler users can share calendar and contact data, and will be able to share much more in the future.

Chandler facilitates information synchronization and updating between repositories. If a user has subscribed to another user's contacts information, and then that information changes, then those changes will automatically propagate to the first user and update her repository.

Computers currently provide user collaboration through two capabilities: sending and receiving messages which form a stream (email, IM's, even blogs), and creating, reviewing, and publishing documents (web pages, word processing). By tightly coordinating these capabilities, Chandler will make it extremely easy to share all types of information with others, to facilitate discussions, organize and coordinate projects, create and review documents, and manage the flow of information and tasks.

Chandler as Platform

Because Chandler is designed as an extensible platform for information management applications, new parcels (the basic unit of organization of Chandler code) can be easily created and added. Our initial testing suggests that adding new parcels is surprisingly easy. Anyone, institutions or individuals, can create and add new parcels to meet their particular needs. Parcels themselves are written in wxPython (Python + wxWindows) and can do practically anything, and have access to all of Chandler's subsystems. The open source and platform nature of Chandler make it easier to support higher education initiatives such as the Open Knowledge Initiative (OKI) or MIT's OpenCourseWare (OCW).

Security

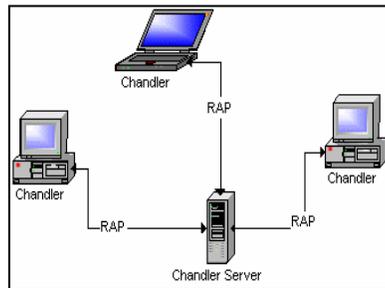
The foundation of our security architecture will be well-proven standards based solutions. We will utilize TLS for session security, and X.509 certificates and SASL (and minimally GSSAPI) for authentication with

other Chandler clients and standards based servers. This ensures that passwords won't ever need to be transmitted in the clear.

To ensure that Chandler is a trustworthy application, we will perform security audits for major design decisions and code reviews. We will support pluggable security extensions and allow numerous policies to be configurable so that each institution can determine its own balance between safety and convenience. Security considerations will always be designed into user interface implementations, communicated clearly where necessary, and transparent to the user where achievable. This will ensure that security is not just theoretical, but practiced in every day usage.

Centralized Deployment Issues and Usage Scenarios

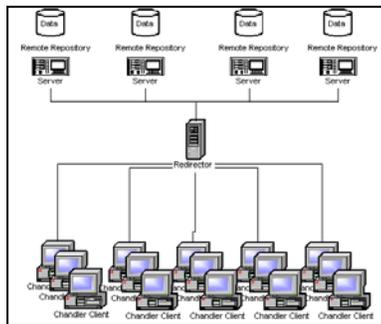
A primary requirement in higher education settings is support for *nomadic* computing. Nomadic usage, where a single user will need to access her information from many different machines throughout the day, is the primary usage mode on college campuses. Some of the computers used may be personal machines – a desktop in the office or a laptop in a meeting – where personal data may be kept on the local hard disk. Some of the access also may be from a kiosk system – a computer in the library, computer lab, or a public kiosk in the student union – where it is important that personal data, including login information, is deliberately not preserved. The Westwood version of Chandler will support centralized servers by splitting the front-end ‘client’ and back-end ‘server’ and using our Repository Access Protocol (RAP) to communicate between the two.



Nomadic Usage Scenario

In Westwood, the primary storage for personal information may either be local, or on centralized servers, with the option to store personal data in multiple repository locations. Chandler will synchronize the multiple repositories, allowing a user to see up-to-the-minute information regardless of which machine is being used to access the data.

To respond to the specific requirements of higher education, OSAF will develop a pilot version of a centralized server, not available in the standard server-optional, peer-to-peer configuration. Westwood will deliver specialized, scalable, high-performance servers that can augment or replace the data store for a standard Chandler front-end client. Over a series of releases, these servers will meet the expressed needs of higher education institutions by providing the required stability, speed, parallel processing capabilities, centralized management, interoperability with existing centralized IT infrastructure, quota management, and deployment on appropriate server platforms.



Large-scale Deployment Scenario Example

In a large-scale deployment with centralized servers, Westwood can be configured so that data from multiple users is not intertwined. With administration tools to manage the Chandler server cluster, separate users can use separate servers using a redirector with a single IP address. Instead of each Chandler server having its own storage there could also be a configuration with multiple servers reading and writing to a single network attached storage device.

The above diagram represents just an example of central deployment. Different institutions can customize their specific deployment of Chandler. Also, in Westwood, Chandler will still retain its peer-to-peer nature and will not absolutely require a central server.

Open Source and Standards Based

Openness is critical to the Chandler project: critical to our ability to develop the best possible solution; critical in allowing institutions to modify Chandler to better meet their needs; critical in providing users with ultimate control over their destiny. We are committed to supporting open standards and protocols. Chandler is available under open source terms and free of charge. Chandler is also an open source development project, in which we invite interested parties to get involved and help Chandler mature.

Similarly, this open approach allows us to collaborate more fully with other open development projects, such as the University of Washington's calendaring projects. Enhanced collaboration improves the chances of developing useful solutions sooner.

Chandler will be a well behaved POP and IMAP client. Chandler will not interfere with other clients' use of the IMAP or POP servers, and will be careful about which messages from which folders to download at what time. Chandler will inherently recognize standards based formats such as iCalendar and vCard for interoperability with other clients.

Chandler meta-data such as annotations and categorizations will not be reflected back to the IMAP server. Instead, we use a technique we term data compositing to provide richer information management capabilities for Chandler while ensuring IMAP server compatibility with other email clients. Likewise, we intend to use data compositing to integrate LDAP and CAP data into Chandler.

Chandler will be able to synchronize contacts and groups information with shared LDAP directories. We will also support LDAP as a means for authorization and access control. Chandler will support schema transformations to support the many different LDAP configurations in universities.

Higher Education and OSAF share a respect for strong adherence to technical standards. We are pragmatically committed to supporting important emerging standards such as CAP as a calendar protocol and Shibboleth as a federated policy framework.

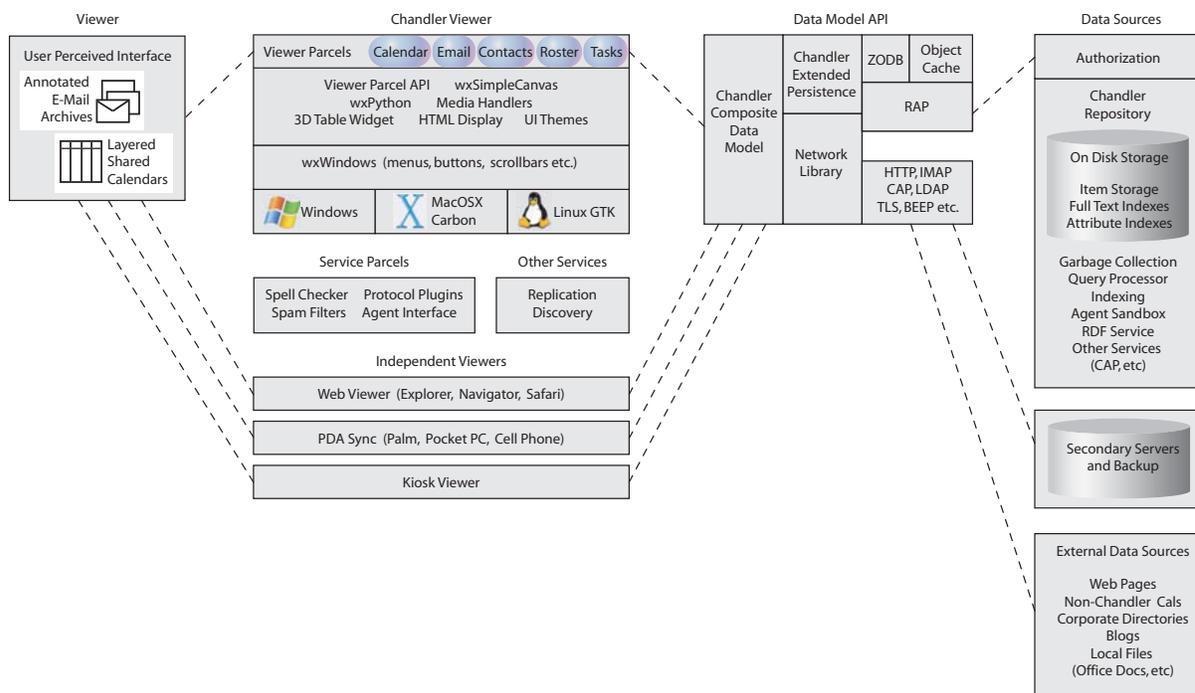
III. An Overview of the Chandler Architecture

Introduction

As Chandler is still in the early stages of development, the architecture is still evolving as we learn more about the problem space. Our vision of Chandler as a platform for a variety of information-centric applications requires a serious upfront investment in planning and design. This section is a snapshot of OSAF thinking as of the 0.1 release of Chandler.

Note that some of the features on this diagram will not be available in either the Canoga or Westwood versions of Chandler. They will come either in later releases or via third-party plug-ins.

Chandler Architecture



Overview

The diagram above is an attempt to show some information about data flow and code modules. The diagram starts in the upper left corner with a task-centric (rather than a delivery-method-centric) view of some data items that are built from a variety of data sources and integrated by Chandler.

The remaining three columns describe the viewers, the interface to data, and the storage of data. There is a deliberate vagueness about client/server boundaries because each viewer implementation might make different decisions about where the code which implements Chandler's notion of shared data will be resident.

The piece labeled *Chandler Composite Data Model* provides a unified view of objects whose constituent pieces could come from a variety of attached and detached sources. For highest performance, this piece needs to be close to viewers of the data. This “middleware” may be instantiated in different places depending on the type of Chandler installation. In the case of a standalone Chandler client, it would exist in the client. In the case of a web-based Chandler client, it might exist in the repository server or in the web based client code.

Viewers

The second column in the diagram shows the Viewers, pieces that aggregate, present data and interact with the user. There will eventually be a variety of ways to interact with Chandler data; the Chandler application is just one of them. The code used to build the Chandler application is split into modules that can easily be included in other applications to provide Chandler behavior on a variety of platforms.

ChandlerViewer

The program that most people think of as Chandler is the *Chandler Viewer* application, in the top box of the second column. This application is written in a combination of Python (for the high level interaction code) and C++ (for the platform specific and performance sensitive pieces). This application is built on a set of high-level info-centric modules. Some of the modules are shared with all Chandler viewers, and some are specific to the Chandler Viewer.

These modules are packaged in a *Parcel*, which is a piece of code that extends the capabilities of Chandler. The *Service Parcels* are the modules that are shared across all applications that view Chandler data, while *Viewer Parcels* are modular user-interface components with domain-specific interactions for different data types. In the 1.0 of Chandler, we will provide parcels for manipulating calendar, e-mail, contacts and task lists.

Viewer Parcel API

Chandler provides a rich API for implementing info-centric applications. The foundation of this is the Viewer Parcel API, which provides simple access to common services shared by all parcels. This architecture makes it relatively easy for programmers to create new parcels that can display new item types, or provide a specialized view on existing item types. Since parcels are written in Python, they can be dynamically loaded, allowing the user to extend Chandler without having to re-compile.

wxPython

Chandler's UI framework is based on wxPython, which is in turn based on wxWindows, a cross platform UI toolkit that uses native widgetry on all three platforms with a common API.

Media Handler, Table Widget, HTML Display, etc.

Chandler will extend the off-the-shelf interaction components available with wxWindows to include various other pieces that are useful in building data-centric applications. There will be host of Media Handlers for displaying images, sounds, HTML, etc, which are used in various parts of the user interface.

We are planning to develop a custom wxWindows user-interface Table Widget. It will be used to display information items in a tree view, where each node in a tree is a table. An HTML layout engine will be used for both displaying web pages and editing HTML e-mail.

Other Services

Here is a peek into the application independent portion of Chandler services. As with the viewer parcels, this list of services can be extended dynamically by importing new parcels.

Discovery

Chandler will use a variety of techniques to learn about the network, allowing Chandler users to interact and share data without requiring central administration. These features are still in early stages of investigation, but we expect to depend on whatever the emerging standards are in these areas.

Replication

Although the servers may perform replication, the control over replication lies with the client. Replication is an action performed at the user's request, not a transparent action performed on the user's behalf.

Other Viewers

At the bottom of the second column are examples of other viewers, such as a Web Viewer, a Kiosk Viewer and PDA Sync applications.

Data Model API

The Data Model API is in the third column. Left to right, across the top of the diagram, we see a rough outline of the implementation stack for the Chandler Data Model.

Chandler Composite Data Model

Chandler provides a unified view to information from multiple sources. In addition to Chandler's own item store, Chandler can view information from IMAP servers, CAP servers, and LDAP servers.

In each case, the foreign "item" (message or component) might have Chandler specific "attributes" (annotations or markup) that need to be stored in a Chandler repository. The Chandler client needs to composite the Chandler item with the foreign item for the viewer. The Chandler item will have meta-attributes that describe its foreign counterpart, including where the foreign item is stored, the protocol used to access the foreign item, the authentication identifier used to access the foreign item, etc. With the exception of "kiosk" mode, there will be a local Chandler repository, so the Chandler item may also cache all or part of the information about the foreign item, for use in offline access or fast searches.

Note that both the Chandler repository and the foreign data source are authoritative. Both data sources need to be examined when fresh information is needed. This task is simpler than it might look at first because each item will be a "composite" of at most two sources, one foreign and one Chandler. Based on the protocol, Chandler will know which data source has precedence for each attribute.

Extended Persistence

Object persistence has long been favored as a highly productive way for programmers to deal with data. Chandler extends the concept of persistence to include network transparency, versioning, and both attribute based and full text indexes.

ZODB

Python has an excellent persistent object store, ZODB. Current plans are for Chandler to use ZODB as the base for implementing extended persistence. As data moves out of memory, Chandler's object store will use RAP (Repository Access Protocol) to transfer data between clients and remote Chandler repositories.

Data Sources

The rightmost column shows myriad data sources that can feed into Chandler. The top portion of the Data Sources column shows a Chandler data repository operating as a server responding to protocol requests from several sources. A Chandler client may act as a server to other clients. A user may have their repository on their own laptop, or they may be accessing their repository remotely through a Chandler information kiosk.

Chandler Repository

A Chandler repository starts with an authorization layer, which provides authentication and access control. Garbage collection or compaction of the data in the repository is provided transparently to the Viewer.

The repository has a query interface that is used to return references to objects based on access to the various indexes stored in the repository. Our intention is that this query interface will be based on a simple but currently unspecified query language.

The creation of indices has to happen in the portion of the Chandler cloud where the data is fully introspectable, and this is currently a fuzzy boundary between the client and server portions, but the data for the indices will be on the server, and so the intention is to have the code which creates the indexes also live on the server.

Repository Service Interpreters

Chandler repositories may also be interesting data sources to other network clients, and so there may be other services which provide an interpretation of the repository. At the bottom of the repository, as examples, we've listed a service that provides an RDF view of the item store and one that allows CAP clients to connect to the Chandler repository

External Data Sources

In the bottom right of the diagram are some examples of data sources. As Chandler is trying to erase the boundaries that exist between different types of data, Chandler will need to inter-operate with a variety of other existing sources of information. Initially this will be limited mostly to importing and categorizing data, but as we gain experience our hope is to extend the full Chandler capabilities of sharing and data-compositing to as wide a variety of users as possible.

IV. Detailed Requirements and Recommendations

Calendar Proposal for Westwood

Introduction

Chandler's first release, Canoga, will include individual and group calendaring features as one element of a more general personal information manager (PIM). We find these aspects of the Chandler calendar particularly compelling:

Inspired by the products Agenda and Ecco, calendar data and calendar views will be part of a flexible general information management system. In Chandler terminology, calendar data (events and tasks) will be represented as "items" with "attributes" and stored in the Chandler repository along with other PIM "items", including contacts, emails and notes. Calendar items might have both calendar-specific attributes, e.g. "start time" and "location", as well as general attributes that any item is likely to have, e.g. "project" or "last modified date". In addition to the traditional day, week, and month views, Canoga will have general purpose outline or table views where users can view calendar items along with other PIM items. For example, a "project" view may display email messages, notes, events and tasks in the same view. Additionally, the calendar views can be used to view other types of items. A user might create a custom view where email messages are displayed in a month view, based on the date of the message. All of the features planned for Chandler items in general will be available for calendar items, including data sharing, custom views, filtering, fast searching, smart parsing, and so on.

- Canoga will offer server-optional group calendaring for small workgroups. We do not necessarily mean that workgroups will not use any servers, but that a 100 person workgroup would not have to hire an administrator to run a special server to use group calendaring. Users will be able to share calendar information and invite others to meetings by publishing calendar data to a common "relay" server, which is just another instance of Chandler that remains available. A Chandler user would be able to invite any other Chandler user to a meeting, whether or not the users are in the same workgroup, as long as the Chandler users can access each other's repositories.
- Chandler's second release, Westwood, will include all of the Canoga features and will add features to support the needs of universities. In particular:
 - Westwood will be a standards-based CAP client. While universities do sometimes have small departmental workgroups who might like to run a decentralized calendar solution, universities also have the need for centrally managed calendars. If a university runs a CAP server to manage such a calendaring system, Chandler will be able to run as a client to that system. This solution will allow the university to provide different kinds of access to the calendar data, including Chandler, web access, and/or any other CAP client.
 - Westwood will include calendaring features of special importance to universities (features that might not be as important to small workgroups.) In particular, Chandler will include designation features and resource scheduling features in Westwood.
 - Westwood will use IETF calendaring standards for interoperability. Chandler will be able to publish or subscribe to iCalendar data, and to use iTIP/iMIP for meeting invitations and responses.
 - Westwood will support a kiosk mode to facilitate the needs of nomadic campus users.

This following section will discuss the Westwood calendar features in more detail.

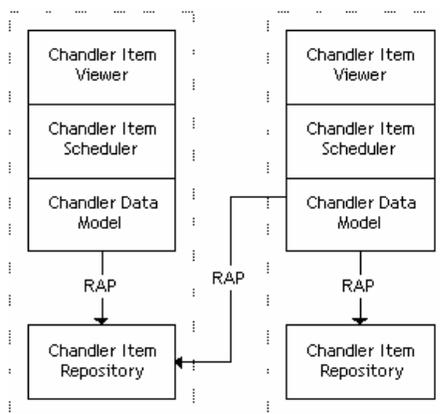
Westwood Calendar Deployment Scenarios

For Westwood, Chandler will support two different modes of operation, which we'll call the workgroup scenario and the CAP client scenario. The workgroup scenario involves the Canoga “serverless” calendaring and low-administration calendaring features. The CAP client scenario involves Chandler using open standards to act as a calendar client (or calendar user agent) to a centrally administered CAP calendar store.

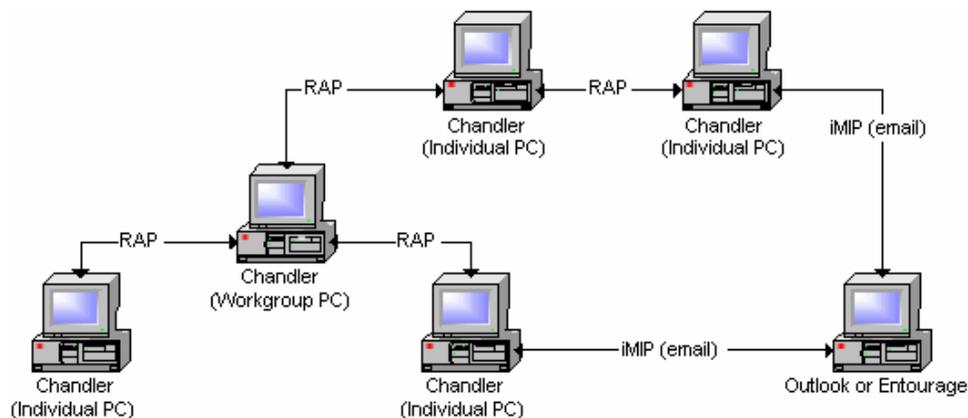
OSAF also has plans in future releases for two other scenarios: full CAP interoperability, so that workgroup users can interoperate with any CAP users, and Chandler as a full CAP server. These four scenarios are described in more detail below.

Workgroup Scenario

In the workgroup scenario, a Chandler user's data is stored on the user's local repository. The Chandler client talks to the local repository using Chandler's RAP protocol. The RAP protocol is a general “item” protocol, the same protocol used for all of Chandler's PIM data.



A Chandler client uses the same protocol to browse or search a remote Chandler repository. In the case of the calendar, the protocol can be used to retrieve free-busy time in another user's repository. As Chandler users might not keep their computers on all of the time, a Chandler user may choose to replicate all or part of their repository on another computer that is on all of the time (a “relay” server—the “workgroup PC” in the diagram below). A workgroup may choose to run one machine as a relay server, for all users in the workgroup to publish shared data. For calendar data in particular, free-busy time would be shared to this relay server. The repository replication features might also be used to synchronize a Chandler user's repository on two machines, for example a home and a work machine.



Meeting scheduling

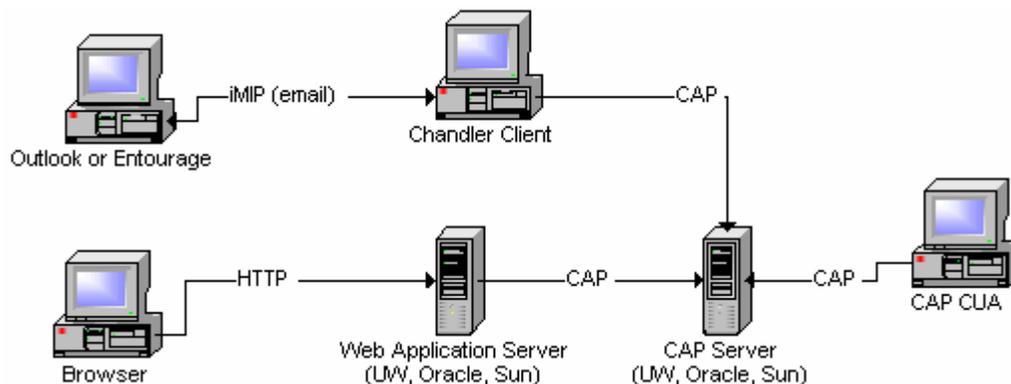
Free-busy data is either an attribute on an event item, or is an item in its own right. Either way, free-busy time can be queried and read in the same manner as all other Chandler data. Special care will be taken to make sure that free-busy time can be queried without fetching other extraneous data, as these requests will be particularly common for scheduling purposes. In general, RAP can fetch “partial” items from the repository: RAP can fetch a subset of an item’s attributes, to reduce network traffic.

Chandler meeting requests and responses will be stored as “items”, similar to iTIP objects. We expect to have a lossless mapping between our calendar items and iCalendar data, including iTIP objects. The meeting request and response items will be stored, queried, etc. using RAP, just like all other Chandler items. The Chandler client is responsible for interpreting these requests and handling them accordingly. This architecture will facilitate interoperability with both iMIP and CAP: the Chandler client will behave similarly whether a meeting request arrived via email or through the repository.

OSAF understands that universities have a history of adopting workgroup solutions in isolation and over time integrating with many decentralized groups across the university. The Chandler workgroup solution for Westwood needs to support scaling in this fashion. One use case particular to calendaring: if many departments are running Chandler in the workgroup scenario, they would expect to be able to invite people from other departments (workgroups) to meetings using free-busy information without the system breaking down.

Chandler as a CAP Client Scenario

While the workgroup solution uses calendaring standards (iCalendar, iMIP, iTIP), it uses RAP as the protocol between the client and the calendar store. For Westwood, we will extend Chandler’s support for calendaring standards to include CAP.



In this scenario, a Chandler user’s calendar data is stored on a CAP server. The CAP server would be provided by a third party. UW, Oracle and Sun have been mentioned as possible providers of CAP servers.

Chandler will “replicate” all or part of the calendar data into the Chandler user’s local repository. This replication would provide some benefits: (1) long term storage if the CAP server has a quota on the user’s data, (2) deep integration between calendar data and other types of data in the Chandler client, (3) access to data when offline. The calendar’s local repository will have meta-data to know what items and attributes are “cached” from a CAP calendar store and what items and attributes are “local” to the Chandler repository.

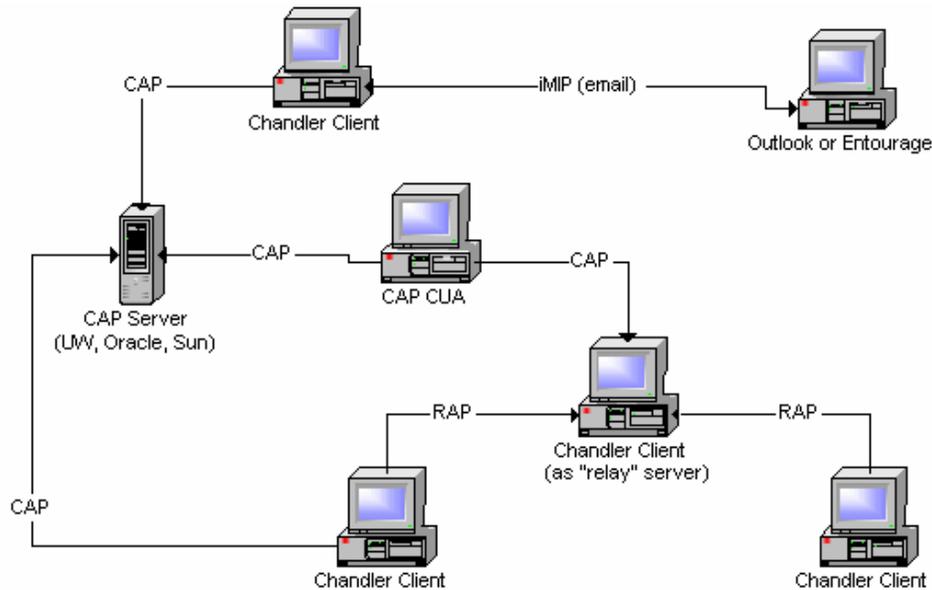
The Chandler client might be aware of different repositories (local vs. a remote CAP repository), but the items and attributes will appear to be the same to the Chandler client’s item viewers. This architecture allows some interoperability between calendar data in a Chandler repository and calendar data stored on a calendar server.

A use case to illustrate the CAP client scenario:

A teaching assistant in the Computer Science department uses Chandler to manage several calendars. The teaching assistant is also the organizer of an intramural ultimate frisbee team. The computer science department maintains a CAP server, and course schedules and all departmental meetings are managed on the CAP server. The TA has a calendar store on the CAP server, and uses Chandler as a CAP client to schedule meetings with other TAs and professors, who also have calendar stores on the CAP server. The TA runs a computer in her dorm room that acts as the “frisbee” calendar relay server with an instance of Chandler running on that machine.

1. The TA can view all of the calendar data in one view in from any of her computers.
2. The TA can invite any of the other CS department colleagues to meetings, and vice versa.
3. The TA can invite any of her frisbee teammates to meetings, and vice versa.
4. The frisbee team members can see the TA's free-busy time, and her CS department colleagues can see the TA's free-busy time, but only if the TA sets up Chandler to replicate her free-busy time to both “servers”.
5. The TA’s CS department colleagues cannot view each others' free-busy time, because Chandler will not support full CAP interoperability in the Westwood timeframe. Similarly, her frisbee teammates cannot view each others' free-busy time.

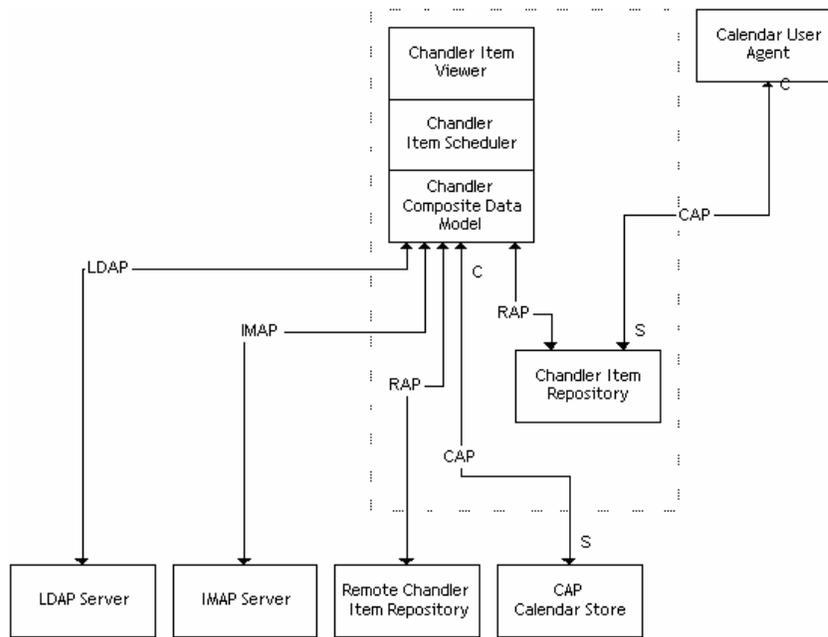
Full CAP Interoperability Scenario



To support full CAP interoperability, the Chandler repository would have to support the CAP server interface. In this scenario, any CAP CUA could schedule meetings using free-busy information with any Chandler user (as long as the permissions were ok), and vice-versa.

While Chandler will not support this functionality in the Westwood timeframe, we are considering this while designing the Chandler architecture.

In general, Chandler's relationship to CAP has some similarity to its relationship to IMAP and LDAP. We'd like the Chandler architecture to support multiple data sources.



Chandler as a CAP Server Scenario

In the long term, OSAF envisions creating a full server version of the Chandler repository. A full server version would need to include administration features, and would generally need to support a full university deployment. When Chandler offers a full server version of the repository, we'd like to be able to provide CAP server access to the calendar data.

Highlighted Calendar Features for Westwood

A more complete list of the calendaring features for Canoga and Westwood can be found on <http://wiki.osafoundation.org/bin/view/Main/CSGreferenceDOCS>

The following section addresses a few of the features that are particularly interesting to universities.

iTIP/iMIP scheduling

Westwood will support iTIP/iMIP scheduling, where meeting invitations and responses are sent via email. iTIP/iMIP scheduling will provide for some degree of interoperability with other email clients.

However, when email is used as the transport for meeting scheduling, the user cannot have access to free-busy time information about the other users..

Designation features

Westwood will support calendar designation features. (Sometimes these features are called “delegation” features).

The user scenario, calendar designation:

A busy CIO has hired an administrative assistant to manage her meetings for her. The assistant has permissions to read and write to the CIO's “work” calendar. The assistant gets a copy of all schedule related notifications (invitations and responses). When the assistant sends a schedule related request (an invitation or a response), it is clearly marked as “on behalf of” the CIO. The assistant never has to “pretend” to be the CIO by logging in as the CIO or knowing the CIO's password.

Although we don't yet have a complete design for the user interface of these features, we are assuming functionality similar to Corporate Time's designation features, or Outlook's delegation features.

Designation features have some interesting implementation issues, related to security. For Westwood, Chandler will implement these features using groups and permissions.

A use case to illustrate using groups and permissions:

An administrative assistant is scheduling a meeting for the CIO. The administrative assistant needs to see the free-busy time for all of the potential meeting attendees. This means that the administrative assistant needs read access to the free-busy time of all of the meeting attendees. The administrative assistant has these permissions because the administrative assistant and the CIO are in the same “group” that has been granted these permissions.

One could imagine the scenario above being handled differently, where the administrative assistant is granted the correct permissions while acting “on behalf of” the CIO, via some complicated rights delegation scheme. We understand that SASL can help facilitate this scheme: after a user authenticates, they can assert an optional authorization ID to “act on behalf of” someone else. While Chandler may do this in a later release, we're not planning on doing full rights delegation for Westwood.

Handheld synchronization (Palm and Pocket PC)

Westwood will include synchronization with Palms and Pocket PC's for calendar events and tasks, we understand this is a must-have feature for universities.

In the long term, we'd really like to write a version of Chandler tailored specifically to handheld devices. One could imagine this client accessing a remote repository wirelessly, and/or having a local repository specifically designed for the device. Although we're not planning on doing this for Canoga or Westwood, we're considering this scenario when making architectural decisions.

Resource scheduling

Westwood will support basic resource scheduling as part of its group calendaring features. Meeting rooms and projectors are examples of resources. Westwood will not provide a full reservation solution, but it will provide features on par with Outlook and Corporate Time.

Web access

Although our eventual plan for Chandler includes full access to calendar functionality from the web, we will not implement a web application for Canoga or Westwood. We understand that web access is very compelling to nomadic users in universities. There are two possible solutions:

As Chandler will be a CAP client, universities can provide web access through their CAP server. The CAP server providers that have been discussed (UW, Oracle, Sun) will all provide web applications for browser access to calendars.

Chandler is open source, will have well documented APIs, and will make use of IETF standards where possible. For example, calendar data will be available from the repository as iCalendar data. Individual universities are welcome to write web applications to access this data. OSAF will do what it can to support these kinds of efforts.

Groups

We plan on having strong support for Groups in Chandler. Like CAP, Chandler will not distinguish between individual users and groups for calendaring features. Groups can be invited to meetings, can be used to define access rights, etc.

A use case for group support features:

An administrative assistant would like to schedule a recurring monthly meeting for all of the administrators on campus. When a new administrator is hired or retires, the administrative assistant adds or removes the person from the “administrators” group. The next meeting that occurs should involve the current set of administrators, not the set of administrators that existed when the meeting was created.

As mentioned above, good group support also helps facilitate the designation access use cases. In general, groups are very important to data sharing and security features, as discussed in Data Sharing in Canoga at: <http://wiki.osafoundation.org/bin/view/Main/CSGreferenceDOCS>.

Event Calendaring

While Westwood will not be a first class event calendaring application, Westwood will have some features that help support event calendaring. Westwood users will be able to “publish” an iCalendar calendar to a server via WebDAV, as does Apple's calendar application. Westwood will also be able to “subscribe” to an iCalendar on a WebDAV server. More generally, calendar data created with a Chandler client and stored in a Chandler repository will be available as iCalendar data.

An illustrative use case, iCalendar support:

A student can download a course schedule from a website, published by the department hosting the course. The course schedule is stored as iCalendar data. The calendar is imported into the student's Chandler client. The course schedule may have been created by the department's administrative assistant using Chandler, or it may have been created using some other application, as long as it is iCalendar data.

Free-busy time

Westwood users will be able to view each other's free-busy time, if they choose to publish that information. To use Westwood's workgroup solution to publish free-busy time (instead of using a CAP calendar store), the workgroup would run a separate instance of Chandler as a “relay server” for highly available shared information. Like a departmental file server, this machine would be administered locally, and remain turned on and on the network. Individual users can “replicate” their free-busy time and other shared information from their local repository to the repository on the “relay server”. A user can then view shared information from the repository on the “relay server”, as long as the user has the right permissions.

Most of the calendar-related traffic in a workgroup setting concerns viewing this free-busy time as users schedule meetings. A Chandler client uses the RAP protocol to fetch this information. Its important that the Chandler client be able to fetch just free-busy information (not all information associated with an event) and be able to fetch information for a particular time period (not for all events in the calendar). Using RAP, the client makes a query to fetch items from the item store. RAP allows the client to specify the “attributes” of the items to be fetched. The query language will support the free-busy time request use case, allowing the client to fetch items from a particular calendar, in a particular date range.

Oracle Calendar (Corporate Time) interoperability

Our primary strategy for addressing interoperability is through IETF calendaring standards. For Westwood, Chandler will be a CAP client. Oracle has publicly stated that Oracle Calendar will support a CAP server in a future release. We are in ongoing discussions with Oracle about standards based solutions for calendar interoperability.

Inter-Institutional Calendaring

Our primary strategy for inter-institutional calendaring is through IETF calendaring standards.

For Westwood, Chandler will support iTIP/iMIP, which enables users to schedule meetings via email. This strategy allows Chandler users to do some group scheduling with a wide variety of calendar users, including folks at other institutions who use Chandler. iTIP/iMIP does not allow users to conveniently publish calendar information, such as free-busy time, as email is used as the transport.

As Chandler will be a CAP client for Westwood, ideally users would be able to use CAP to share calendar information across institutions, including free/busy time. This scenario requires an authentication infrastructure that spans institutions. Federated systems such as Shibboleth are an example of such an infrastructure. We will support such standards as they mature and are adopted.

Chandler users will be able to fall back on the Canoga peer-to-peer mechanisms for sharing information, within institutions and across separate institutions, if other authentication infrastructures are not available. In the case of the calendar, any calendar information that is stored in the Chandler repository (cached from the CAP server, for example) will be accessible through this mechanism. For more information on how peers authenticate with each other in Canoga see Data Sharing in Canoga: <http://wiki.osafoundation.org/bin/view/Main/CSGreferenceDOCS>.

Email Proposal for Westwood

Email is one of Chandler's core, fundamental parcels, but we realize that CSG members do not perceive current email clients to be inadequate. While we will take care that people will be able to use Chandler in conjunction with a non-Chandler email client, Chandler's email parcel will have some features that we believe will encourage people to switch to Chandler. This section will discuss Canoga's compelling features, and then discuss how we will tailor Westwood to deal with higher education's specific needs including filter security and IMAP interoperability.

Canoga's Compelling Features

With Canoga, users will be able to manage large volumes of email more efficiently and effectively than with current email clients. Canoga will help people

- view and manipulate data of any data type (e.g. Calendar and Contact) grouped by project
- focus on email messages that are most relevant and important to the task at hand
- read through messages faster

Canoga will do this by

- integrating email tightly with other data types
- providing rich filtering and display capabilities
- eliminating spam

Tight integration with other data types

Canoga will integrate email tightly with other data types, in both display and in function.

- Users will be able to easily add arbitrary attributes to information items then search all items based on any attribute (inherent or user-defined). For example, a user could tag particular messages with the attribute "Cobra Project", and then view all of the information with the attribute "Cobra Project" -- calendar items, messages, contacts, etc -- in a "virtual folder".
- It will be easy to augment email messages so they have the properties of additional data types. For example, it will be easy to convert a message with a request into an explicit Task, complete with Due Date and Actual Completion Date fields.
- Canoga's powerful text analyzer will recognize elements in email messages and connect them appropriately to other data elements. While a significant amount of interaction design and user testing need to be done to determine how to best make use of this feature, some potential uses include:
 - If a date and time appear in the body of an email message, selecting that field could create a Calendar entry for that date and time. (That Calendar entry could then be back-linked to the email message.)
 - If a phone number appears in the body of a message, selecting that field could bring up a dialog box for adding that phone number to the sender's Contact record.
 - If the phrases "Please", "I would like you to" or other such "request language" appears in the message, the message could be augmented with Task attributes. If the message contains a phrase "by next Tuesday", the due date could be set to next Tuesday by default.

Rich filtering and display capabilities in Canoga

Current email programs do not make it easy for users to see messages of immediate importance separated from messages that are not germane. People are able to easily sort their inbox by various fields like the date or the sender's name, but those fields usually bear little resemblance to the priority order.

Some power users use filters to move messages into separate folders based on message attributes. For those skilled at creating filters, this can group messages nicely. Unfortunately, filters are difficult for most people to create. Furthermore, most people have a difficult time keeping track of their “to-do” messages -- ones they need to read, reply to, or act upon -- when they are spread across multiple folders.

Use case example, View by relevance

Canoga will make it much easier to view messages grouped by relevance *in the inbox*.

One morning, Professor Mabel Garcia looks at her inbox. In the current View of her inbox, she sees unread messages listed in this order:

1. messages from her spouse
2. messages from her colleagues in the History Department
3. messages from her colleagues in the Communications Department
4. messages from students in her History of Writing Systems class
5. messages from students in her History of Communications Technologies class
6. messages from other people at the University of East-Central Illinois at Hoopston
7. messages on her “orality and literacy” mailing list
8. messages from her friends
9. messages from her family
10. messages from other people she has corresponded with
11. messages on her parachuting enthusiasts' mailing list
12. messages from strangers

Each group has an expand/collapse button to allow her to hide messages that don't interest her at the moment.

Canoga will have several features that will make it extremely easy to view messages in this manner:

- Filters
- Views
- A “done” button
- Powerful threading

Filters

First, Canoga will support powerful filters that users can easily import and export on a filter-by-filter basis. Thus, if one person comes up with a handy, worthwhile filter they can make that filter available to many other people. These filters might include individual spam filters or filters to help prioritize messages.

The question has been raised as to why any filtering should be done on the client side instead of on the server side. While server-side filtering is valuable, there are some very good reasons to do client-side filtering instead of or in addition to server-side filtering.

Most server-side filtering is very limited, particularly in dispositioning. For example, the base SIEVE implementation only has the ability to reject a message, file it into a folder, redirect it, delete it, or pass

it through. To enable some of Canoga's high-volume email management features, the filters must be able to change a message's attributes in a more flexible way than even the SIEVE IMAP extensions allow.

For non-programmers, server-side filter conditions have traditionally been either very limited (e.g. SIEVE) or very daunting (e.g. procmail). Canoga will be able to check a number of useful filter conditions that are difficult to check on the server side including, “is the sender in my address book?” and “have I corresponded recently with the sender?” (The latter is very useful for vacation messages.)

- While we hope that plug-ins will catch a high degree of spam, eliminating spam is an extremely important part of processing email more effectively. While there are many spam features that a server is in a better position to recognize, there are some spam-related analyses that the client is in a better position to perform. **Whitelists.** For whitelists to work well, every address that a user sends a message to should automatically be put on a whitelist. Every address in the user's address book should also automatically be on the whitelist. Because SMTP, IMAP, and LDAP are disjoint, this is difficult to do properly on the server side.
- **Display processing.** Spammers are starting to BASE64-encrypt messages and to modify the text so that it looks very different in “eye-space” than in “ASCII-space”. (For example, spammers sometimes insert HTML comments into the middle of “spammy” words, like `Via<!--92438-->gra.`) In order to recognize spam, the spam filters are starting to need HTML parsing -- something that the clients already have.
- **Compute-intensive algorithms.** There are some anti-spam strategies (e.g. CamRam) which involve providing a proof of computational work, on the theory that such computations would be too expensive for spammers to do millions of times per day. Even if the algorithm is more computationally expensive for the sender than for the receiver, a university's central email server might not enjoy doing this many times per day.
- **Learning classifier systems.** There are a few very promising anti-spam techniques that involve learning systems (e.g. The spambayes project). By their very nature, they depend upon feedback from the user. That feedback is tricky to channel back to a server-side filter.

While there is no “silver bullet” to eliminate spam, there are lots of “lead pellets” that in combination can eliminate spam very effectively. Canoga will make a number of anti-spam mechanisms available by default. Some have been mentioned above; OSAF will eagerly incorporate new techniques as they are proved effective.

Filters can also help to organize and prioritize non-spam messages. Again, to do so well requires extensive understanding of a user's individual patterns – what mailing lists the user subscribes to, who they correspond with frequently – that require information that is normally available on the client side but not the server side. These techniques will be elaborated on in the next sections.

Views

Canoga will have very powerful, customizable Views that are very easy to import and export. Again, easy sharing greatly leverages the efforts of a single person.

The combination of easily sharable filters and easily sharable Views is potent. Filters can change attributes of a message that Views can then display differently. For example, if you import a:

- filter which changes a message's Category to the Category you gave the sender's Contact record
- filter which assigns a particular Category to messages from inside your domain
- filter which assigns a particular Category to a particular mailing list
- filter that assigns a particular Category to people you've corresponded with in the past
- View that groups messages by Category

then you will have exactly the type of View that Dr. Garcia has in the example above.

Note also that because Canoga will implement all folders as “virtual folders” – results of stored queries – a message can be visible in more than one folder at a time. Thus, regardless of what task you are

trying to accomplish, you can always view the relevant set – and only the relevant set – of messages in one view.

Visual coding

The powerful Views can also help users recognize which messages in a group are most important. Not only can Views have particular sort orders for fields like Date or Importance. Items can be visually differentiated by fields or by how the receiver was addressed. Visual differentiation could be done through color-coding or by typographic styles. For example, Dr. Garcia could color-code (or import a View that color-codes) as follows:

- red - messages that are TO her and only her
- blue - messages that are TO her and other people
- green - messages that are CC her only
- black - messages that are CC her and other people
- grey - messages that are BCC her.

No other consumer-grade email client has such powerful organizational features. While Microsoft Outlook's Views can do similar grouping and color-coding, there is absolutely no way to share Views. While Microsoft Outlook's users can import and export filters, the filters are not as powerful and can be imported or exported on an all-or-nothing basis.

“Done” button

Canoga will have a “done” button that users can press to mark the selected message when they no longer have to read, respond to, or act upon it, but don't want to delete it. Views can then be adjusted to show only messages that are not “done” yet, helping the user to focus on “to-do” messages. When a message is marked “done” and disappears from that View, the next message will automatically be selected.

Some email clients allow the user to set a flag, which in theory a user could use in this manner. Alas, in practice, flags do not work well for this purpose.

- Messages usually arrive with the flag un-set, so either the user has to declare that:
- An unset flag means “to-do” and a set flag means “done”, or
 - set the flag manually for each message that has been read but is not done
 - write a filter to flag each message as it arrives
- Messages occasionally arrive with the flag set by the sender. If the user has defined that a set flag means “done”, then these messages will automatically be marked “done”.
- It is sometimes a fair amount of work to flag a message and move to the next one. On Microsoft Outlook, for example, it takes at least four mouse actions to flag a message and select the next message.
- No consumer-grade email client is able to hide Unflagged Read messages without also hiding all Unread or all Flagged messages.

Powerful Threading

Canoga will have powerful threading and summarizing. Views will be able to show messages in threads and to hide threads that have no unread messages.

In addition, we plan to explore Views that displays contextually related paragraphs from a message thread in an outline view, with extraneous quoting and signatures stripped out. (See <http://zest.sourceforge.net/> for an example.) This makes the messages seem much more like the transcript of a conversation. This reduces the cognitive load enormously and allows users to read through messages much faster.

Westwood Filter Issues

Some of Chandler's ability to help users deal with large volumes of email quickly is enabled by powerful, easily sharable filters. For Canoga, we plan to use Python for the enormous power it gives. We believe that, for Canoga, malicious filters will not be a big problem for the same reason that malicious shareware downloads are not a big problem. When shareware is downloaded from Web sites, the filters are traceable, require user intervention to install, and the user has incentive not to be indiscriminate. It's much easier for a malicious hacker to spread a virus by email, so that's where they invest their effort.

We realize that in a university environment, students might not be as sophisticated, won't have as much incentive to keep campus machines from going down, and might not keep track of where an importable filter came from. Instead, for Westwood we will share filters in the same way that we share other sorts of Chandler Items -- by allowing people to read other people's repositories. For example, a university could set up a 'blessed' repository where tested and approved filters and other Chandler plug-ins could be obtained.

In effect, people would share data structures instead of code. Filters would thus only be able to take a small number of actions instead of the entire universe of actions that Python makes available. The worst that a filter could do is move messages to the trash; the second worst thing would be to alter the categorization (equivalent to "changing the folder"). While these are both undesirable, they are much less calamitous than spreading a virus.

There will also be several features of Westwood filters that will reduce a malicious filter's impact.

- Users will only be able to "pull" filters from someone else's repository, instead of filters being "pushed" to a user's repository.
- When a user imports a filter from a friend, the friend's name and IM address will be saved with the filter, so that malicious filters have a chance of being traced in a chain back to their source.
- Users will have the option of examining the filter at any time, including before they save the filter. Users have a chance to notice if a filter has strange behavior.

IMAP Interoperability

Canoga has a data model that assumes that most data resides in a repository, while IMAP has a data model that assumes that the canonical source of email messages is the IMAP server. Merging these two models will be complex but possible.

We understand that many people will want to use Chandler for some functions (particularly calendar) but continue to use a different IMAP client. Thus,

- Chandler will interoperate with an IMAP store in both off-line and on-line modes.
- Chandler will not alter the IMAP data in unusual ways that would break any other client's usage of the IMAP server.
- Chandler will also respect the IMAP server is a canonical email source.

Note that the IMAP server might only be one "canonical source" out of several because a local copy of email used during an off line session is also a canonical source during the off-line session. Naturally, changes to both sources must be merged when the client goes "on line" again.

Double storage: advantages and issues

One possible is to use "double storage". Double storage would occur if Chandler always kept a copy of all the IMAP data in its own repository. Keeping a second copy is desirable for a few reasons:

- A second local copy allows for caching and off line access.
- Once loaded, a local copy means faster response time.
- Keeping a copy in the repository makes implementation easier for the programmer, since the repository provides a unified method for searching, accessing, annotating, and versioning.

Unfortunately, keeping a second copy is also a problem in a networked repository environment. The second copy means extra storage and higher bandwidth requirements for moving the entire IMAP store to a local repository. There are also situations where an IMAP server may provide access to public folders that are far too large to be stored in the user's repository. For example, CMU exports USENET via IMAP public folders, so any CMU user might appear to have several thousand folders with thousands of messages each.

Solution: Compositing

A proposed solution is to use *data compositing* -- keeping the email on the IMAP server and only storing Chandler metadata for the messages in the Chandler repository. Chandler would combine message data pulled from the IMAP server with metadata pulled from the Chandler repository to provide a complete data view for a message. For example, an annotation for a message would be stored in the Chandler repository. When viewing the message, the annotation data from the repository would be combined with the message data from the IMAP server to present an annotated message.

For email, the compositing is relatively simple. There are only two sources, the IMAP data and the repository annotations. One source (the annotations) takes precedence. Due to these assumptions, the code that merges the data can be relatively straightforward and does not require complex collision handling routines.

IMAP Annotations

If IMAP annotations become an IETF standard and enough servers support it well, OSAF would be delighted to do away with data compositing and instead use the IMAP annotations to keep all the metadata on the IMAP server. However, it would be imprudent to base our design upon unsupported infrastructure that has no momentum with server providers.

Email caching

Local caching of email is important for almost all IMAP clients. Caching is useful for speed of repeated access as well as for off-line use. When a repository is local to the machine it would be used for caching purposes. In Chandler's case the copy would not be a true "cache" because changes made to that data during off line use would be reflected back to the IMAP server. Examples of this would be deletions or changes to message flags. Changes made to the IMAP server would also be synced back to the repository.

Caching of infinitely large IMAP stores such as IMAP access to USENET is clearly not practical. Chandler will need to add a method for expiring cached data for messages from these kinds of IMAP stores. In the normal email case cached messages will probably never expire from the repository, so we will need to identify which folders Chandler should hoard messages from. (For many servers, Chandler will be able to use the NAMESPACE extension to determine the user's personal folders.)

A possible simple solution is to default to hoard INBOX messages and to expire messages from all other IMAP folders. We will also allow the user to explicitly specify which personal folders Chandler should hoard and which Chandler should expire regularly.

Other IMAP issues

Interruptible downloads

Since IMAP doesn't allow for multiple connections by the same user to the same mailbox and doesn't have intrinsic parallelism we will need to use special methods to allow the user to interrupt operations that can potentially take a long time. Examples of this are message header retrievals and large message retrievals. Other clients have dealt with this by always asking for things in manageable chunks, for instance asking for twenty headers at a time or 1 K of message data at a time. Chandler will probably deal with interruptibility via this chunking method. For greater efficiency Chandler will try to dynamically adjust the chunk size based on the speed of the connection.

Off-line vs. On-line modes

Most IMAP clients have a notion of off-line and on-line modes. Some require you to specifically choose an off-line mode so that all the data can be downloaded from the server at that time. Chandler would like to have a more seamless experience. To do so, Chandler, unless instructed not to, will download all email messages to the local repository when getting mail, but will try to do so in an intelligent manner. Chandler will not follow the POP model that requires all mail to be retrieved up front before you can view it. Instead, Chandler will act like a traditional IMAP client, first downloading only the necessary headers, and then standing by to retrieve any message that the user wishes to view. During idle network time, Chandler will download messages from specified folders to the repository in the background.

For example, user Fred fires up Chandler and checks his mail.

- Chandler will get all the headers necessary to display the message list for Fred's INBOX. Chandler will try to get the list in the sort order that Fred's view has specified and will stream the headers into view so that Fred waits as little as possible to see results.
- As soon as Fred clicks on a message to read it, Chandler will begin downloading that message and displaying it.
- While Fred is reading that message, Chandler will begin downloading the messages immediately following the message Fred is reading.
- When Fred is ready to read the next message, it should already be in his repository, so access will be immediate.
- If Fred clicks on a message that has not yet been downloaded, Chandler will interrupt its download of other messages and immediately download the requested message.

This method of access should provide Fred with a very active response to email reading and since all messages in specified folders will eventually be downloaded to the repository, Fred can take his computer off-line with the expectation of having all of his messages available. Chandler also will need to provide Fred feedback when the download process is complete.

IMAP does not have good support for resynchronization. For instance, there is no efficient way for a client to find out which flags have changed since the last client connection or which messages have been expunged. Thus while "disconnecting" is a fast operation, "reconnecting" might be a slow one. This is especially tricky if you want the client to move from connected to disconnected mode during momentary network blips and then transition back to connected mode when it detects connectivity has been restored. There is currently no known good solution for this. Like all IMAP client providers, we would be delighted to learn one.

Searching

The repository and IMAP each provide its own search mechanism. In general, the repository search mechanism will be more fully featured than the IMAP one. When some messages are only in IMAP and some are cached locally, we have a difficult issue to deal with. Chandler will probably keep a flag that specifies if all the messages have been cached in the repository or not. If all message are in the repository then the repository search mechanism will be used, otherwise the IMAP mechanism will be used. In the future, we hope to find a better mechanism that could leverage both...

Chandler Features reflected in IMAP

Chandler annotations and other changes to mail message bodies will not be reflected back to the IMAP server because IMAP messages are supposed to be unchangeable. In the future, we could implement a feature to write annotated messages back to the IMAP server with new message ID's, but we do not plan to do so in Westwood.

Other IMAP features

To maximize usability of IMAP, Westwood will provide the following options, among others:

1. Chandler will support multiple accounts to multiple IMAP servers
2. The ability to check periodically for new mail and optionally download it

3. The ability to turn off disk caching (for kiosk use)
4. The ability to specify which IMAP folders' messages should be hoarded
5. Use of TLS
6. Optional methods of deletion, including moving to trash, marking and expunging
7. Sent folder CC'ing

Contacts and LDAP interoperability

Introduction

LDAP directories are widely used for a range of applications in larger organizations, including academic institutions. Chandler will therefore need to interoperate with LDAP to some degree to be successful in these environments.

The question of how Chandler should best interoperate with LDAP directories is complicated by the fact that such directories are used for a range of purposes, and are used both by end-users and by applications. Thus we need to first identify interesting use cases of LDAP directories and decide which of these we want to target.

The following list are the significant use cases we have identified:

1. On-line "phone book" (i.e. directory of contact info for end-users)
2. Group directory (list members of organizational groups, mailing lists, etc.)
3. Email routing information
4. Login/service account information
5. Authentication data (public keys)
6. Authorization data (unix groups, ACLs)
7. Directories of system information for distributed system management

Although anyone who uses LDAP is likely to use it for at least some of these cases, things are complicated by the fact that they are not necessarily doing so in a compatible fashion... for example, one organization might store the members of a group using the *uniqueMember* attribute of the *GroupOfUniqueNames* object class, another might use the *member* attribute of the *GroupOfNames* class, yet another might forgo both of these and reconstruct groups from the list of UIDs of the *PosixGroup* object class. Or, to make things worse, some organization might not think any of these approaches adequate for their needs and define a whole new object class for essentially the same purpose.

Ignoring these kinds of issues for a moment, **Cases 1 and 2** are relatively simple to support as a "client", i.e. an application that looks up the information in the directory on behalf of the user... in this case the application (Chandler) could directly use a certain set of well-known attributes and present the rest "raw". This is the approach taken by Netscape.

Case 3 is of relevance to Chandler email, but it is expected that the existing mail transport infrastructure will handle this case, so Chandler will not need to support it directly.

Case 4 is relevant to Chandler account/identify management.

Case 5 has at least one well-standardized instance (X.509 certificate) which is likewise addressed in the next section on security

Case 6 is a difficult one which Chandler will probably need to support in some way. We describe an approach below with some restrictions.

Case 7 is not relevant to Chandler... we can restrict ourselves to thinking of LDAP directories as containing information about people.

For the rest of this section, we will focus on Cases 1,2 and 6. In **Chandler and Data Sources**, we describe how Chandler incorporates data from LDAP. In **Schema Transformations**, we describe a means for allowing Chandler to interoperate with the many disparate LDAP schemas. We then return to the highlighted uses cases in **Detailed Uses Cases**. Finally, we cover miscellaneous university requirements related to LDAP such as FERPA.

Chandler and data sources

Chandler's intrinsic architecture supports multiple data sources. Although the principal data source is the Chandler repository, we intend to also support other sources, e.g. IMAP as a data source for email mailboxes. Since Chandler data is “richer” than typical email stores, Westwood will support objects which have attributes coming from multiple data sources. For example, a mail message might be stored in IMAP, but associated metadata would be stored in the Chandler repository. We do this by compositing; the Chandler repository can contain objects or attributes which are actually external references that cause a lookup to the external data source and interpolation of the result.

Our approach for LDAP fits into this model. Chandler “contact list” information is similar to the data stored in LDAP directories, so if a Chandler instance is running in a context where an LDAP directory is available, the information from that directory will be merged with information from the Chandler repository in real time.

Note that this means that both data sources are authoritative. Chandler may cache LDAP information in the repository, but in contexts where freshness is important it will still have to do a lookup to both data sources every time data is needed. Since there is usually a local Chandler repository on a machine, if the machine is off-line, it might be able to use data cached from a prior LDAP lookup. This is a simple case of synchronization.

Caching policies

Chandler will need to cache data from external data sources in order to be able to function in disconnected mode. Most LDAP data is highly cachable as it doesn't change very often and the freshness of data isn't very important in most use cases. However in some use cases freshness may be absolutely critical, and it may be that while object freshness is not critical, specific attributes must be fresh (i.e. an attribute that grants some form of authorization).

To accomodate this in Chandler, we need a way of specifying the caching policy. This could be done in one of two ways:

1. In LDAP by extending the LDAP schema with an attribute that Chandler can interpret as Caching policy
2. In Chandler by specifying patterns that define caching policy for sets of matching LDAP objects and attributes

The former will be preferable in an organizational context as it puts the configuration with the already established processes for LDAP directories, but has the disadvantage that it requires some change to the existing infrastructure.

Schema transformation

Chandler data is inherently extensible. LDAP directories are also extensible and have flexible schemas, and as we pointed out before these are often extended in an incompatible fashion. We will have a flexible mapping of an LDAP schema to a Chandler data. For example, since we will want to use LDAP information about groups as mailing lists or aliases, we will map the various ways of representing groups of people in LDAP to a Chandler group.

We will define a filter API that lets us write plug-ins to accomplish the transforms. Chandler can then ship with a set of filters for common LDAP use cases, and organizations can modify these or write their own if needed.

The filter will to accomplish several things:

1. Map LDAP DNs to a set of identifying attributes in the Chandler repository so we know which LDAP object matches which Chandler item.
2. Map LDAP attributes to Chandler attributes.
3. Potentially merge multiple LDAP objects into a single Chandler item or vice-versa.

This is complex, but made somewhat simpler by the fact that Chandler only needs to do transforms in one direction.

The filters themselves can either be written directly in Python or we could define a high-level declarative language for transforms and have a generic filter that can interpret this language. This would then allow us to store the filter transforms themselves in LDAP, which would potentially simplify administration.

Use cases

Address book / Contacts

This simplest use case is easy to support. It includes things like dynamic lookup of email addresses, the address book, and expansion of groups for email or other contact methods. For this case “freshness” is probably not critical and Chandler can cache or synchronize all LDAP access to the Chandler repository (through the filter of course). The principal difficulty here is one of data source precedence, i.e. when to go to the LDAP server for data vs. using data from the local repository in cases where the two might overlap or conflict. This is mainly a UI issue. The UI will ensure that all the relevant choices/conflicts are presented to user and/or that the default resolution mechanisms represent “least surprise”.

Group directory

There are two principal use cases for groups. One is a user convenience, e.g. sending an email to a group rather than listing all the recipients individually or inviting a group to a meeting in the calendar. The other is the use of group membership as authorization data, i.e. someone may have access to data or may have certain rights because they are a member of a specific group. The former case is identical to Address book/Contacts, but the latter case is more difficult, and is discussed in the next section.

Groups are somewhat problematic in LDAP because there are so many ways they can be represented, e.g. explicit membership lists, implicit through attributes on the members, dynamic through a search, etc. All these can be mapped to explicit groups using the schema transform mechanism. Chandler will store all groups explicitly in Chandler repositories and all cached LDAP groups will be explicit, even if they were the result of an expansion of a dynamic or implicit group in LDAP.

Some use cases of group directories require access control, e.g. specific people have the right to add/remove members, only members can list the membership, etc. One problematic case for the above model is the need to verify if a person is a member of a group, but we don't have the rights to list the full membership of the group. This is problematic because it introduces a conflict with the desire to always fully expand groups as described above. We will have to handle cases where an external reference remains unexpanded by appropriate representation in the UI. When Chandler is on-line it can do the lookup, when it isn't it has to show the group as being unlistable/unverifiable.

User/Account Information

LDAP is widely used to store information about users of various services, i.e. login names, number user IDs, etc., and a significant amount of administration in an organizational setting is the maintenance of this information. Chandler will use this existing information (mapped through the filters discussed above) as the basis of information about users who can authenticate to Chandler. See the discussion of authentication in the security section for more information.

Authorization and access control

A common case for this is authorization by group membership, i.e. a user is authorized to access something if they are a member of a certain group. It is likely that supporting some of these cases would

be extremely valuable to organizational users. We realize that because Chandler can share data, limiting access by mechanisms already in use in an organization is probably a requirement.

There are a couple of difficulties with supporting this in Chandler using this scheme,

1. Data freshness is very important
2. The transforms must be unambiguous

These should not be insurmountable. We will use an attribute to indicate that cached data may not be used for authorization, so the data must either be looked up in the LDAP directory again, or authorization must fail with an appropriate exception. See also the section above on Caching policies.

There are other authorization cases as well, such as authorization by existence of a particular attribute. All these cases in fact can be mapped to an instance of authorization by groups internal to Chandler, so they could all be handled by appropriate customization of the filters.

Miscellaneous considerations

LDAP binding and data access

Not all the data a Chandler user might want from an LDAP directory is likely to be public, and in the higher education context, regulations like FERPA place requirements of limited access on such directories. We can assume that access control is already implemented at the LDAP level, so to function in such environments, Chandler will have to authenticate as a user who has some level of privileges on that LDAP server.

Conclusion

Chandler will be able to synchronize contacts and groups information with shared LDAP directories as required in a university setting. We will also support LDAP as a means for authorization and access control. As with IMAP, Chandler will access LDAP data through data compositing. Chandler will support schema transformations to support the many different LDAP configurations in universities. Finally, we will be mindful of unique university requirements such as FERPA, to not accidentally access and publish public private student information.

Security

Trustworthy application

Chandler must be a trustworthy application. OSAF will use existing security infrastructure whenever possible.

The users of networked applications today are weary of security breaches. Chandler users must be confident that their data and machines are protected, and their privacy assured. Systems administrators must be confident that their users are not exposing themselves to unknown risks when using Chandler.

By keeping security in the forefront of our design we will deliver a trustworthy application. Every feature will pass a security audit as part of the design. The code will be audited for general security problems. Security considerations will be designed into user interface implementations, communicated clearly where necessary and transparent to the user where achievable. This will ensure that security is not just theoretical but practiced in every day usage.

Some features, such as email attachments, data sharing, scripting, will always carry some security risks. System administrators will be able to preconfigure Chandler to obtain desired mix of features and functionality.

Using existing security standards

Chandler will be a client of many established protocols: IMAP, POP, LDAP, and CAP. For these, we plan on implementing existing standards.

Our networking layer will be based upon SASL & TLS. SASL will provide authentication, and TLS over-the-wire encryption. Obsolete unsecured authentication mechanisms such as unencrypted passwords will not be enabled by default. Choosing to use TLS and SASL was easy, since they are widely used in academia, and mature open-source implementations exist for both. They are also regularly extended with new security mechanisms, making it easy for us to keep up (e.g. Shibboleth).

IMAP/POP connections will be authenticated through SASL mechanisms (Plain/GSSAPI) and optionally encrypted with TLS.

Secure email will be supported as S/MIME with X.509 certificates. The import/export mechanism for X.509 email certificates & Certificate Authorities is to be determined.

No authentication/authorization solution will satisfy everyone. Our libraries will be extensible, so those whose needs are not met by default will be able to implement their own extensions.

Data sharing security architecture

Introduction

There are no pre-existing designs for data sharing that meet our requirements, so we will build our own. However, in building Chandler's data sharing mechanisms we will employ two guiding principles:

1. Reuse existing point solutions where applicable.
2. Make it simple for the user.

Data sharing is easy to define from a user's perspective: Meg meets David and exchanges email addresses with him. She would like David to see her calendar for the next week. In Chandler, Meg views the next week's calendar, and selects the 'share' button and types David's email address. David receives the resultant email, clicks on a link, and loads up Meg's calendar into his own.

There are many engineering questions hidden in this simple scenario. The main ones are:

1. authentication
2. authorization
3. over-the-wire encryption
4. general safety concerns
5. groups
6. designation
7. infrastructure-mode specific issues

Universities have extra security requirements to interoperate with existing infrastructure. For this section, we refer to security interactions with such existing infrastructure, *infrastructure-mode*.

This document describes the general design, and skips over many details for clarity. The full design discussion can be found at:

Data Sharing in Canoga: (discusses the p2p scenario) and **Data Sharing in Westwood:** (discusses Chandler usage within existing authentication/authorization/directory infrastructures) can be found at: <http://wiki.osafoundation.org/bin/view/Main/CSGreferenceDOCS>

First, we will define some terms and then summarize of our design for each of the main problems.

Sharer = repository owner. Sharer grants permissions to grantees to access the repository
Grantee = user that was authorized by sharer to access sharer's repository

Authentication

Authentication needs in data sharing are varied, and so are our solutions. The mechanisms supported will be:

Nonce (obfuscated URL)	Nonce is used to give access to a grantee when we don't have any credentials. It is a temporary, one-time pass. The intention is to email the grantee an obfuscated URL to log in for the 1 st time.
Username/password	Sometimes, a grantee does not have access to his credentials from the repository, but wants to access a sharer's data. Username/password mechanisms can be used in this case. The Chandler client will enforce strong password policy.
X.509 certificates	In the absence of a central authentication authority, some features, e.g. rights delegation and groups are only possible with public-key cryptography. Chandler will automatically generate and exchange certificates on 1st login with nonce. If a certificate is lost, a grantee will have to re-authenticate using a different mechanism (nonce, username/password)
Infrastructure mode (SASL)	If there is already a common authentication authority for both sharer and grantee, we'd like to take advantage of it. This will be done through SASL. In particular, Kerberos is a good fit.

How are these authentication mechanisms used and credentials exchanged? In peer-to-peer usage, we have two separate cases:

Initial login:

Before the initial login, a sharer does not have the grantee's credentials. A sharer emails the grantee the obfuscated URL. The grantee uses it for an initial login. As part of the initial login, a grantee and sharer will exchange their permanent credentials for future communication. A grantee will supply username/password, and his public key, and the sharer will supply only his public key.

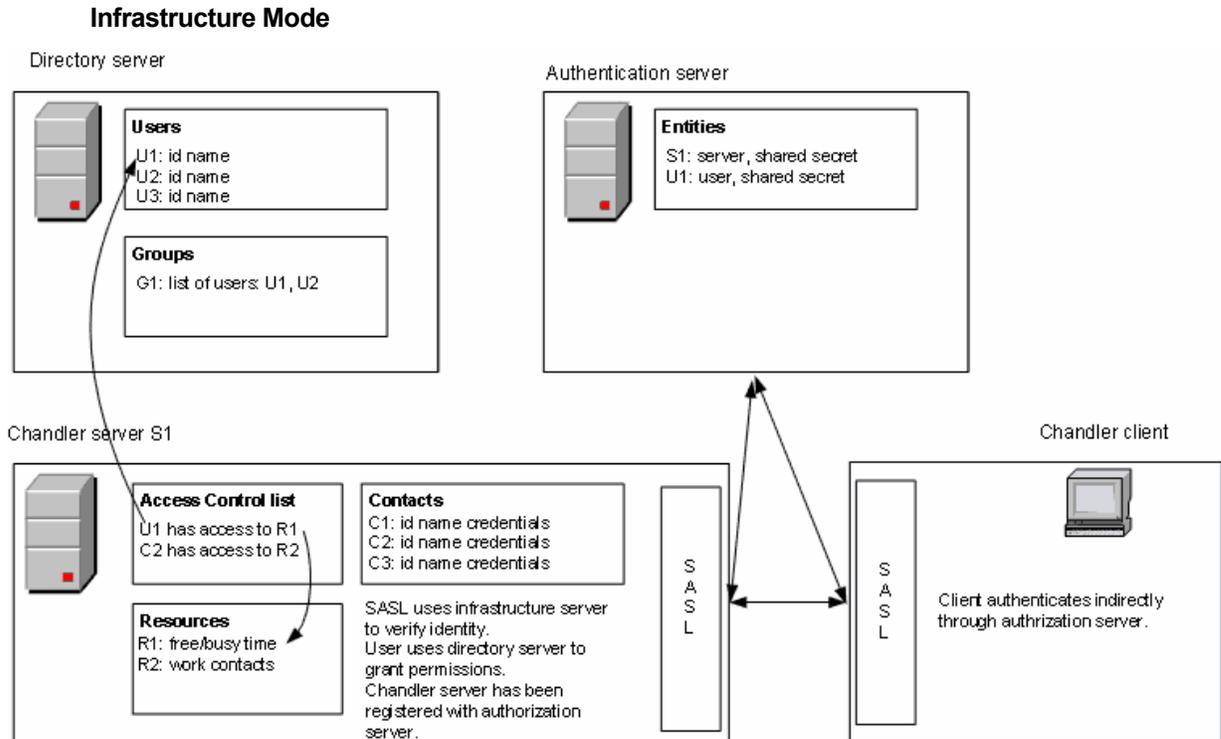
Future logins:

The grantee will log in using his credentials supplied on the initial login. Public key will be the primary login mechanism, with username/password used as a fallback if keys are unavailable.

In the infrastructure mode, things get complex. Chandler is both a server and a client, and as such has infrastructure requirements that go beyond most other applications. The requirements are:

- A sharer needs a directory from which to select a grantee.
- A grantee needs to authenticate to the sharer as the person from the directory.
- A grantee should not reveal any sensitive credentials to the sharer, such as username/password. If username and password are used for authentication, then we need a Kerberos-like system, where the sharer presents a ticket.

This means that directory servers and authentication servers have to be tightly integrated. A common deployment scenario is a directory server, e.g. LDAP, and authentication server, e.g. Kerberos. For Chandler to be fully integrated in this setup, the LDAP server would authenticate users through Kerberos.



Unfortunately, tight Kerberos/LDAP integration is not available in many campuses today. Some alternatives to this ideal scenario are:

- Allowing Chandler servers to authenticate users to an LDAP server directly using username/password. This requires crypto to hide passwords in transit through the Chandler server. The LDAP server would somehow have to be extended to decrypt Chandler's passwords (as a SASL extension).
- By storing Chandler public keys on LDAP directories they could also function as authentication servers, but we are flirting with problems inherent in full-scale PKI deployment.

The infrastructure deployment story is not as straightforward as we'd like it to be, but infrastructure deployment never is. We do know that LDAP will be the preferred directory provider in the higher education environment.

Authorization

Authorization information will be stored inside the Chandler repository. Each authenticated user has a list of capabilities associated with her account. Users have to be aware of what permissions they grant and to whom. This is both a user-interface problem, and a database architecture problem. The current proposal is for users to share views of the data.

Keeping all authorization information inside the Chandler repository might not satisfy organizational infrastructure needs. Institutions may want to manage authorization from a centralized LDAP server. OSAF will provide an extension API for plugging in external authorization sources. We'll look at GAA-API and AZN-API for inspiration.

Shibboleth appears to be an emerging framework for distributed authorization, especially in inter-institutional scenarios.

Over-the-wire encryption

Our favorite problem is solved by one word: TLS (aka SSL). TLS will be used to encrypt all data going over the wire: usernames/passwords/repository data. Our repository access protocol (RAP), BEEP already supports TLS. In high-performance situations, we'll support session resumption.

General safety concerns

Data sharing is inherently risky activity. A sharer or grantee could turn malicious; hackers can try to break passwords. We will take reasonable precautions to defend against these attacks:

- When a sharer's data is not what the grantee expects: The danger is similar to a CGI script that processes form data. Any data that comes from the wire will be validated. (If the grantee does not do data validation, it might be possible for a sharer to pollute the grantee's repository, or worse.) An example of such an attack would be automatically scheduling a meeting with a grantee when he looks at the sharer's free/busy time.
- When a grantee is abuses write privileges: If a grantee abuses his or her write privileges by filling up our database with garbage entries, we will have the ability to roll back the unwanted changes.

Other commonsense security measures will be implemented:

- Westwood will log sharing activity.. A logging agent could present a summary to the user, flagging suspicious activity (an account accessed from hundreds of IP addresses might indicate a leaked password/key)
- Westwood will take reasonable precautions against common attacks: for example, ten wrong passwords could lock out an account.
- System administrators should be able to evaluate the safety of risky Chandler features and create their own security policy. This way, each institution can pick their safety vs. convenience threshold.

Groups

Chandler will treat groups as first-class objects with behavior equivalent to individuals for addressing, access, and delegation. There are two kinds of groups:

- Private.** These are defined by the sharer, and all the information about group membership is contained inside the sharer's repository
- Public.** The group is defined by the group owner, who is not necessarily the sharer. Group membership information needs to be communicated between the sharer and group owner.

Private groups are easy; everything is on a single machine. Public groups are more complex, as group membership information is separate from the sharer's repository. Our group solution for peer-to-peer and infrastructure scenarios will be very different:

- In the peer-to-peer scenario, we'll use public key certificates. A group will have a key. Group members will prove their membership by presenting their certificate signed with the group's key. The compromises in picking this method of proving group membership are "revocation by broadcast", and "if you lose your key, you have to reapply".
- In the infrastructure scenario, we will rely on an authentication server for group membership information. For example, Kerberos already supports groups.

Designation

Designation is a solution to a common problem: a user (sharer) would like another user to perform actions inside the sharer's repository on the sharer's behalf. Calendar scheduling with an admin working for the boss is a common example. This feature is known as designation in Corporate Time, and delegation in Outlook

With designation, the sharer can let a grantee modify the sharer's repository on the sharer's behalf. The grantee can then log into the sharer's repository and act as the sharer.

How is this different from just giving a grantee write access to the sharer's repository? Some repository changes will generate notifications to other users (for example, meeting scheduling confirmation). With simple sharing, this notification would go out with the grantee's name. With designation, the emails will have headers like "From: sharer (by grantee)"

Designation is not a perfect solution to all "perform on sharer's behalf" problems. For that, we'd need to implement rights delegation. This will be addressed in subsequent Westwood releases.

Infrastructure mode specific issues

Chandler users should be able to designate with what users and groups they'd like to share data. Our current plan is to designate users through Chandler's Contact Items. Our current plan for Contacts in the infrastructure mode is to retrieve them from the address book. We do not know what, if any, standards are used for retrieving groups and roles information from LDAP.

Each Chandler repository is also a potential server. In Kerberos, each server needs to have a shared secret with the master server. How we create the shared secret for every Chandler client is still to be determined.

Conclusion

The foundation of our security architecture will be well-proven standards based solutions. Chandler will utilize TLS for session security and will employ X.509 certificates and SASL (and minimally GSSAPI) for authentication with other Chandler clients and standards based servers.

OSAF will perform security audits for major design decisions and code reviews. We will support pluggable security extensions and allow policies to be configurable so that each institution can determine its own safety / convenience trade offs. Security considerations will always be designed into user interface implementations, communicated clearly where necessary, and transparent to the user where achievable. This will ensure that security is not just theoretical, but practiced in every day usage.

V. Centralized Deployment Issues and Usage Scenarios

Introduction

Chandler's initial target market with Canoga is a server-optional configuration. We recognize that the higher education institutional setting will require more robust deployments with a spectrum of server configurations that add administrative control, availability, reliability, and robustness not expected in the server-optional, peer-to-peer configuration. The following sections will outline the variety of usage scenarios and configurations we expect to support with Westwood.

Chandler has a flexible network repository architecture for data storage. There are methods for accessing your own repository data on your local machine, remote data on other machines, or data on central servers. There are also ways for sharing your repository data and accessing other people's data. Repositories can also be synchronized either fully or partially. How people will use their repositories will depend on their own needs, their need to share information, and the available network and server infrastructure. This document will highlight some usage scenarios to explain this in more detail.

Chandler Centralized Deployment Issues

Central Server Issues

One major functional difference between the higher education Westwood version and standard Canoga version of Chandler will be Westwood's additional support for centralized servers. A key element in our design for Westwood will be to split the front-end 'client' and back-end 'server' using our Repository Access Protocol (RAP) to communicate between the two. This will allow the development of specialized, scalable, high-performance servers that can augment or replace the data store for a standard Chandler front-end client.

In a higher education environment, central servers for Chandler will support and provide a scalable centralized Chandler network repository including:

- Stability
- Speed (efficiency)
- Parallel processing
- Administration, configuration, and management
- Interoperability with existing IT infrastructure
- Quota management
- Server hardware platforms

Chandler Central Server Requirements

Chandler as an application is designed to have very flexible network capabilities. Chandler can exist in a purely peer-to-peer environment because it contains both client and server components, but can also act as a client in a client/server environment. P2P in Chandler is very useful because it enables groups of users to interact and share information without the aid of central servers. This allows for easy adoption and low setup overhead for small groups. In large organizations the use of central servers will allow for centralized resource management, network efficiency and easier resource discovery.

For higher education usage, Chandler will need some level of centralized services, because of the numbers of people involved and current operational methodology. There are many existing central services that Chandler will be compatible with, namely IMAP, POP3, SMTP, LDAP, Jabber, and HTTP. The Chandler network repository is a new service that doesn't have existing implementations. The initial Chandler application, Canoga, will ship with a network repository that is suitable for small work groups but will not

serve the needs of a large organization. The subsequent Westwood version will provide a network repository and be more scalable.

Scalable Chandler Network Repository

The Westwood version will ship with a scalable network repository. This implementation will differ from Canoga in several key areas: stability, speed, parallel processing capabilities, management capabilities, interoperability, quotas, and platforms.

Stability

The server application will be expected to run for months at a time without crashing. This requires a very different level of stability than most people expect of PC applications. To accomplish this goal, the architecture should be as simple as possible to reduce overall code size. It should also be modular, so that sections can be stress tested individually and so that we can reduce interaction complexity. Code profiling tools such as Purify will need to be used to eliminate memory leaks and runtime memory problems. Our target is to make something as stable as IMAP servers such as Cyrus and UW.

Speed (Efficiency)

A large-scale network repository will need much greater efficiency per operation than a single user repository, to keep hardware costs to a minimum. The server will be written in C and C++ with no interpreted code. Optimizations to the architecture and critical code sections will be necessary.

Performance targets:

- Several hundred simultaneous users accessing a high-end single processor PC class machine with a performance IDE RAID subsystem.
- .2 second maximum lag time for an average network transaction.

Parallel processing

The architecture of the server should be capable of running efficiently on more than one processor and should have functions that make it possible to split the load across multiple servers.

Targets:

- Per-processor scalability of 90% efficiency per added CPU up to 4.
- The initial release will have hooks for clustering, but not all the necessary components.

Administration, Configuration, and Management

Managing a large server has a much larger set of problems than managing a single user database. Tools for efficiently dealing with large numbers of users and data will need to be developed. For example, we will need configuration files and tools for system administrators. GUI methods are not acceptable. In addition, institutions will be able to create pre-configured clients for standard distribution (e.g. kiosk installations with no local storage, settings to keep the client from remembering passwords, IMAP / LDAP server addresses, etc.).

Interoperability with existing IT infrastructure

When integrating Chandler into the preexisting IT infrastructure, not only will Chandler be able to be preconfigured with appropriate centralized services information, but there should be a consistency and therefore coordination with account setup of other central servers including:

- IMAP servers
- LDAP servers
- Kerberos authentication servers
- CAP servers (when available)

Quota Management

Per-user quotas of some sort will be a necessary feature. We will need to provide administrative support to notify users when they are approaching quota limits, the ability for the administrator to increase or decrease limits on a per-user, or per-group basis, etc.

Server Hardware Platforms

The server would likely be developed on and for UNIX and Linux variants. We believe that while CSG might like the code portable to Windows, it not a hard requirement, since universities rarely run large servers on Windows. .

Higher Education Usage Scenarios

Isolation Scenarios — No sharing of information, but still multiple machines use cases.

Example 1. A user who only ever uses one machine. (The simplest example)

Example 2. A user with two machines, e.g. a desktop and a laptop.

Example 3. A user with a high-availability network repository

Example 4. A user with a few machines and a network repository.

Example 5. A user who uses many machines for short intervals. (Kiosk usage, Web Access)

Example 6. Power user.

Sharing Scenarios — Sharing information, multiple machine use cases.

Example 1. Peer-to-peer

Example 2. Workgroup - Relay Server(s)

Example 3. Kiosk Usage

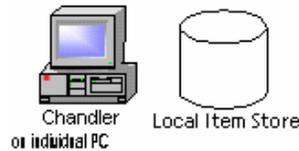
Example 4. Nomadic Usage

Example 5. Scalability of a server based Chandler repository

Example 6. Web access and thin clients

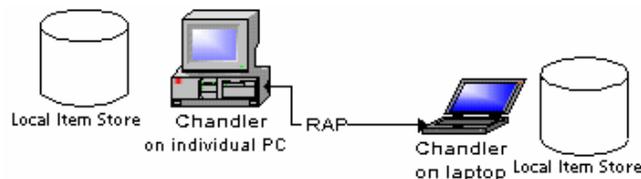
Isolation Scenarios — No sharing of information, but still multiple machines cases.

Example 1. A user who only ever uses one machine. (The simplest example)



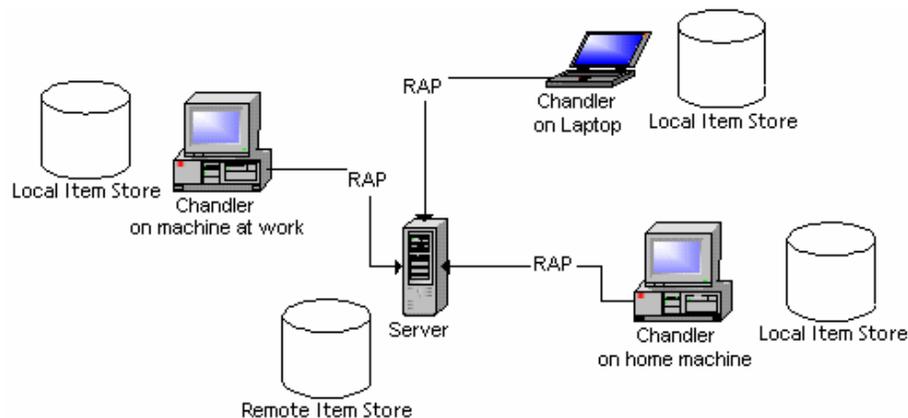
A user who only ever uses one machine and doesn't care about access from others is the simplest case. Chandler in this case is reminiscent of most PC desktop applications. All data will be stored on the local machine and no network repository accesses will take place. Lots of network data may be imported or synchronized, such as IMAP and POP3 mail, but all repository access will be local.

Example 2. A user with two machines, e.g. a desktop and a laptop.



A user with two machines usually wants to keep both computers synchronized. Whenever switching over to the other machine, the user would perform a synchronization step that brings both machines to the same state. This case is very similar to the PDA model. If both the machines are used often, the time to do a synchronization should be small. This example extends to almost any user who uses a few machines often, or who wants full synchronization for offline use. It is also possible to schedule automatic synchronizations if both machines are left on for predictable intervals. Note that in some cases where conflicts are detected that require user intervention, automatic synchronization may be impossible.

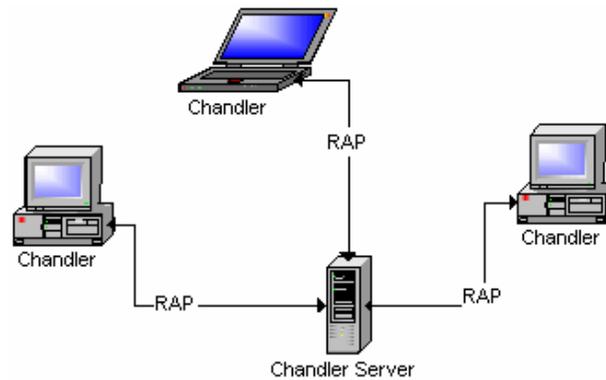
Example 3. A user with a high-availability network repository



There are many reasons why a user may want to use a network repository. A repository hosted on a high availability server with automatic backups and guaranteed uptime is a powerful tool. If backups are the only need, a user may choose to synchronize their repository to the network. They could also use the server as a relay. By synchronizing changes from one computer to the network repository, another computer could later synchronize with the network repository. (This is an effective strategy for bypassing firewalls.) A user could also use the network server to access data from an information kiosk (as in Example 5 below).

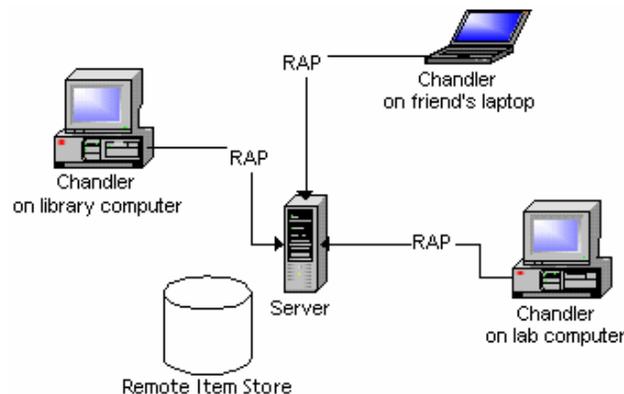
Note: Using a network repository is also a way to reliably share information with others. The network server can provide 24/7 access for other users to shared parts of a repository, like free/busy time in the calendar. Examples in the Sharing Scenarios sections below will go into more detail.

Example 4. A user with a few machines and a network repository.



A user with a few machines and a high availability network repository has several choices. They could choose to keep a local repository and synchronize with the network repository at regular intervals, or they could choose to keep all data on the network and access it all remotely. The choice would largely depend on the speed of the network server and the bandwidth available. If they are running on a high-speed network with a very fast server there should be little difference in speed between the two scenarios. If the network server has limited speed or is behind a slow network, the user would find synchronization to be a faster option. In either scenario, the user will have a copy of her data on a network server that can provide backups and/or 24/7 access for herself and others.

Example 5. A user who uses many machines for short intervals. (Kiosk usage, Web Access)

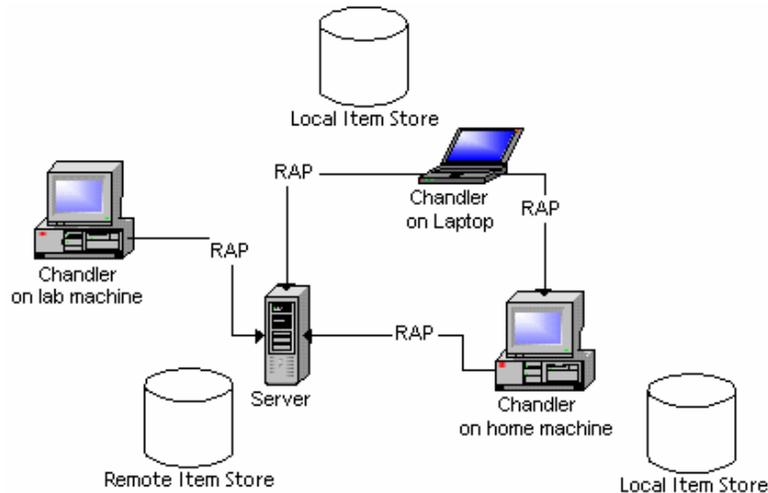


Keeping all data on a network repository and accessing it remotely best serves a user who is highly mobile and uses many different machines for short periods of time. Since synchronizing a repository that is new or severely out of date can be an expensive and time-consuming operation, it would be much faster to access a

network repository and skip synchronization. This scenario would also apply for someone using a kiosk machine that is publicly accessible and/or doesn't allow disk access. Note that the user from Example 4 may also use this method part of the time if they wish to use a kiosk machine.

Another method of accessing your personal information from a remote machine would be web access. This implies a browser interface to a web service application running on a machine hosting your data store. While this is feasible, it is not planned as a solution in the Westwood timeframe.

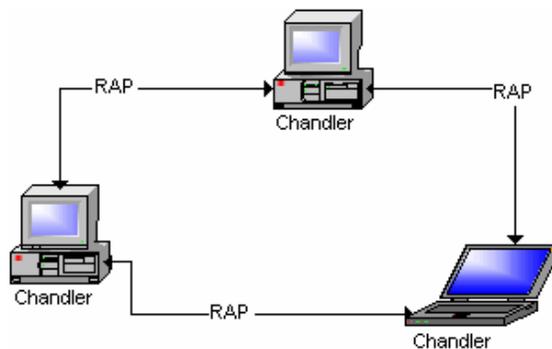
Example 6. Power user.



A power user may choose to use parts of all of the above example usages. For instance, one could synchronize a desktop and a laptop as well as synchronize with a network server. At various times the user may use a kiosk for short intervals and use remote access to the network repository or even their own desktop. Westwood will include synchronization with Palms and Pocket PC's for calendar events and tasks, as we understand this is a must-have feature for universities.

Sharing Scenarios — Sharing information, multiple use cases.

Example 1. Peer-to-peer



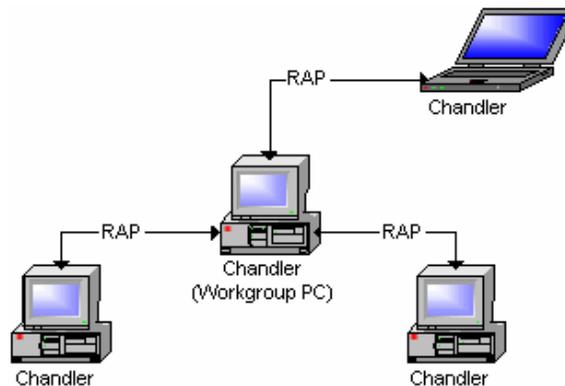
Chandler will offer a server-optional group calendaring for small workgroups. We do not necessarily mean that workgroups will not use any servers, but that a 100-person workgroup would not have to hire an administrator to run a special server to use group calendaring. Users will be able to share calendar

information and invite others to meetings by publishing calendar data to a common “relay” server, which is just another instance of Chandler that remains available. A Chandler user will be able to invite any other Chandler user to a meeting, whether or not the users are in the same workgroup, as long as the Chandler users can access each other's repositories.

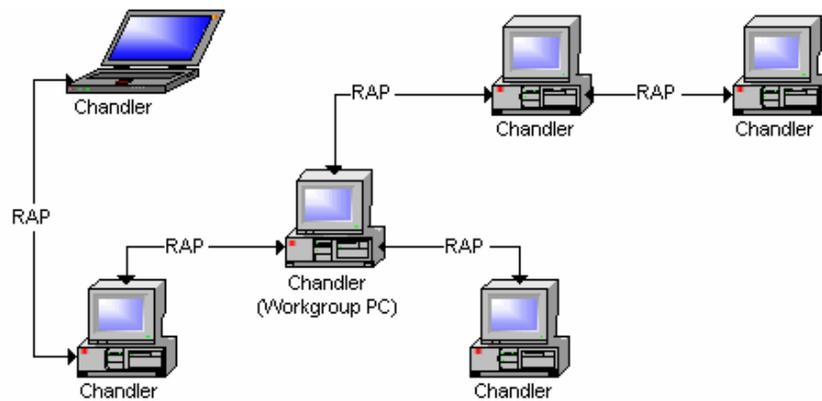
The simplest use case is an individual that uses Chandler in an isolated mode, with only ad-hoc connections to others to share data or to receive subscribed information. Chandler’s design will allow discovery of other Chandler presences on the network in the manner in which instant messaging services discover that “buddies” are online. When two Chandler clients are concurrently online, the users may reveal and provide access to any selected information in their local repository. Each user determines what information she wants to share or to keep private. This mode allows for the most efficiency if a user uses a few machines often. All the data in a remote repository can be synchronized at once with a local instance. With any form of synchronization, conflict detection will be implemented with a user interface for conflict resolution.

In a university setting, when people use Chandler as an email client, there is no “pure” peer-to-peer email use. With a POP server, the email information is often downloaded to the user’s local machine and stored there. Once that occurs, there is no further need to be connected to the email server to access the content, attachments, annotations, and sharing of the downloaded email. In the case of IMAP, Chandler will normally use a compositing approach, leaving the original content (except for selected duplicated items) on the IMAP server and only storing meta-data, markups, annotations, and perhaps the attachments on the local machine for access and sharing.

Example 2. Workgroup - Relay Server(s)



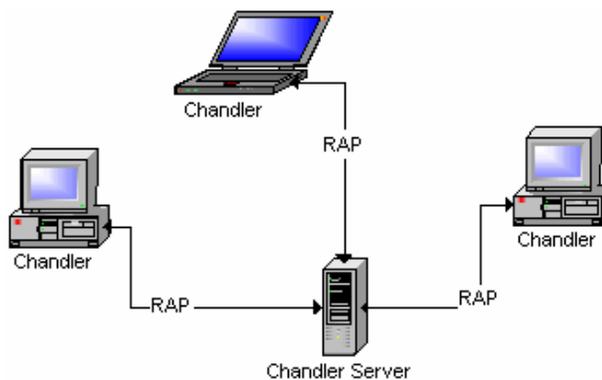
Or in a more complex workgroup configuration:



In the workgroup scenario, a Chandler user's primary data store is the user's own local repository. The Chandler application client 'front-end' talks to the local repository 'back-end' using Chandler's RAP protocol. The RAP protocol is a general "item" protocol; the same protocol is used for all of Chandler's PIM data, regardless of whether the repository is local or remote.

In this scenario, the user's data is stored locally and can be shared in P2P fashion with others, as in the P2P Example 1 above. But in this case, to ensure reliable access to the shared information, the local data is synchronized with secondary repository on a 'relay server'. The relay server can be just another machine running Chandler, perhaps a PC that is left running under the departmental secretary's desk. This instance of Chandler requires no administrative management such as setting up accounts, passwords, user quotas, etc. (although backing up the data in the Chandler repository could be a nice benefit of a workgroup server). Because this workgroup server machine remains up 24/7, free/busy times for calendar information and other high demand personal information can be reliably accessed even when the local user machine is offline. This allows users to set up meetings, look at contacts, or other data each user has specified as public. The next time the user synchronizes with the relay server, the local is updated with the new meeting.

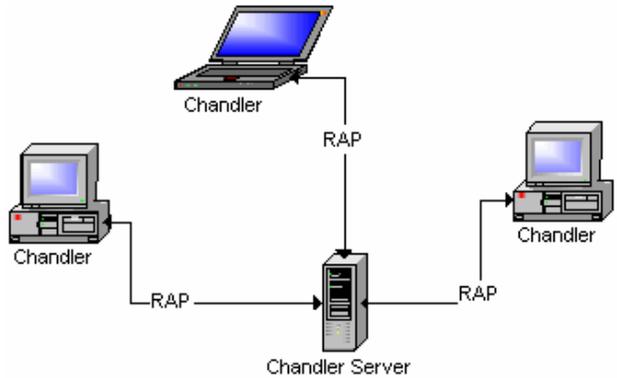
Example 3. Kiosk Usage



For security reasons, kiosk usage (e.g. machines in a computer lab or library) requires that no local data storage remain in the Chandler repository on the kiosk machine. Westwood will be configurable so that after login, the local instance of Chandler will use the RAP protocol to access the user's data store only from a remote repository and not synchronize the remote data with the local repository. If there is sufficient bandwidth in the network connection, the remote server should be able to provide data to the kiosk client with little noticeable performance degradation. In this case, all user data will be eliminated after the user logs out or the session times out, but of course any changes or additions will be saved in the remote server's repository for later access.

This mode will allow for efficient use from a machine that may only be used once or infrequently. Chandler's real-time access protocol RAP will allow network access to the repository. For efficiency, searching will occur at the remote repository.

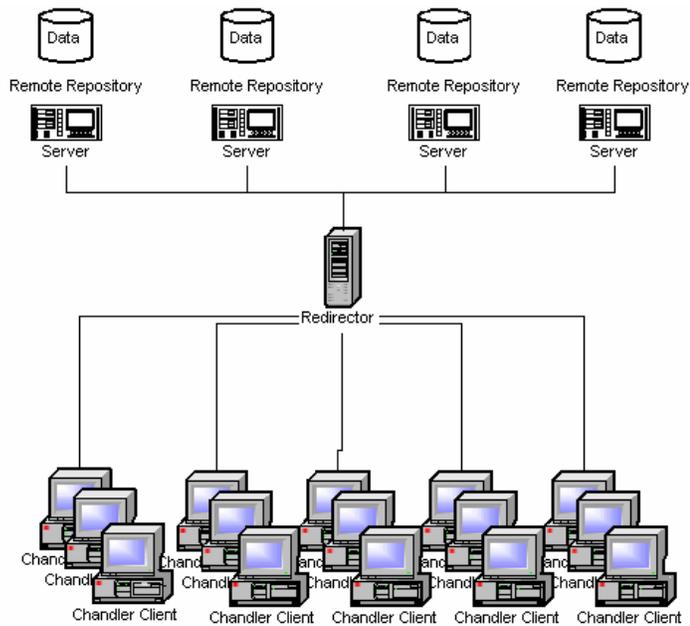
Example 4. Nomadic Usage



Chandler's kiosk mode will help meet the needs of nomadic campus users who may use multiple machines throughout the day—some personal, some public, some low-bandwidth, some high-bandwidth. The nomadic case differs from the pure kiosk mode in Example 3. The primary storage may either be local or central, but there also can be multiple data stores. Chandler will synchronize the multiple repositories, allowing a user to see up-to-the-minute information regardless of which machine they are on.

If the user connects to the Chandler server from her own PC through a high-bandwidth connection, she could choose to synchronize the local repository with the central server, passing the latest information bi-directionally. If however, she has a slow, low-bandwidth connection, she might choose a 'kiosk-style' connection to see just the information she needs at the moment. Later, when a high-bandwidth connection is available, she can synchronize her personal repository and the server repository.

Example 5. Scalability of a server based Chandler repository



In a large-scale configuration, Westwood could be configured so that multiple users' data is not intertwined. Separate users can use separate servers. A redirector could be set up to dynamically partition users on multiple servers with one IP address. Users could be repartitioned to other machines

for efficiency or fail-over. Instead of each server having its own storage, there could be a configuration with multiple Chandler servers reading and writing to a single network-attached storage device. Of course, these configurations would require central administration tools to manage the Chandler server cluster.

The above paragraph represents just an example of central deployment. Chandler will be deployable in multiple configurations. Also, Chandler will of course retain its peer-to-peer nature and not require a central server.

Example 6. Web access and thin clients

While we believe that our users will want to be able to read and write their data from any web browser, and even to be able to read and write from portable devices like PDA's and cell phones, we are not committed to developing a web interface or PDA conduit for the Canoga or Westwood releases. However, these are exactly the kind of features that we expect third-party developers to develop, and we will encourage this effort.

Conclusion

The higher education version of Chandler will add administrative control, availability, reliability, and robustness not available in the standard server-optional, peer-to-peer configuration. The Westwood version of Chandler will support centralized servers by splitting the front-end 'client' and back-end 'server' and using our Repository Access Protocol (RAP) to communicate between the two. Westwood will deliver specialized, scalable, high-performance servers that can augment or replace the data store for a standard Chandler front-end client. Future versions will extend the functionality of the Chandler servers to enable scaling to support campus-wide installations. These servers will meet the needs of higher education institutions by providing the required stability, speed, parallel processing capabilities, centralized management, interoperability with existing centralized IT infrastructure, quota management, and deployment on appropriate server platforms.

VI. CSG Member Volunteers

For approximately six months, OSAF and various CSG representatives have been exploring ways in which Chandler could be valuable in a higher education setting. We began with two all-day meetings, one in December 2002 and one in January 2003. From these meetings, CSG representatives concluded that Chandler could be a compelling application for higher education. OSAF's original development scenario for Chandler targeted the needs of individuals and small- and medium-sized organizations, rather than the large, complex mix of existing infrastructure found in higher education settings. As a result, there are a set of architecture, design, and implementation issues that must be addressed before the CSG members can determine Chandler's usefulness for higher education.

In March 2003, encouraged by the response from the initial meetings between CSG and OSAF, the Andrew W. Mellon Foundation awarded a grant to OSAF to develop the incremental requirements for a higher education version of Chandler and to present the resulting plan to CSG members at the May 2003 CSG meeting to evaluate the probability of deployment at CSG member sites.

To create the requirements plan for a higher education version of Chandler, OSAF worked with a number of CSG members including both CIOs and technical domain specialists. We identified particular areas of concern and conducted a series of discussions with domain specialists in each identified area. These discussions formed the basis for a preliminary set of recommendations circulated to the domain experts. More discussions followed, and refined sets of recommendations were developed, discussed and re-circulated. The various CSG members have contributed a great deal of their time and expertise to this process, which has been instrumental in creating the enclosed plan.

OSAF especially wants to thank the following members of CSG.

Jim Bruce	MIT
Tracy Futhey	Duke
Michael Gettes	Duke
Terry Gray	U. Washington
Paul Hill	MIT
Vace Kundakci	Columbia
Larry Levine	Dartmouth
Jack McCredie	UC Berkeley
Jeff McCullough	Berkeley
Bob Morgan	U. Washington
Andy Newman	Yale
Dan Oberst	Princeton
Mark Poepping	Carnegie Mellon
Joel Smith	Carnegie Mellon
Sean Smith	Dartmouth
Oren Sreebny	U. Washington
David Wasley	UC System Office