OPEN SOURCE APPLICATIONS FOUNDATION

Agents

This area is for discussing issues related to agents and/or end-user scripting.

Andy's Agent Framework Proposal (We really want your feedback!)

Movitation for Agents

Chandler's user interface is based on a landscape of views, arranged in a set of modules, each with an associated URL. Each view presents a collection of items specified by a query from a (possibly remote) data repository, displayed in a spreadsheet-like table or in a view-specific, flexible fashion. While that's fine for browsing and manipulating data, it doesn't provide an intuitive way to express continuous activity or initiative. We'd like Chandler to be able to respond automatically to various situations as they arise, as instructed by the user in a simple, understandable fashion. Chandler should be capable of orchestrating complex, multi-step, asynchronous tasks in the background, like booking reservations for a business trip or arranging a conference call between multiple parties. Non-programming users should be able to specify actions that are performed in response to certain events while programmers should be able to define new types of events and actions, besides the ones built into Chandler.

We propose an agent framework to accomplish the above. Agents are plug-in modules containing Python scripts and other resources like images, that perform tasks on behalf of the user, either when explicitly requested or when the agent notices particular conditions arising. Agents have a compelling visual representation based on an anthropomorphic "occupation" metaphor (postman, janitor, secretary, courier, sleuth, etc), in which we express as much of their internal state as we can visually (they look busy, alert, bored, sleeping, impatient, pleased, etc). They can be instructed by the user without any programming, simply by associating predefined conditions with predefined actions.

So the Chandler user interface might present two different kinds of plug-in modules, views (actually sets of related views, currently called "packages"), and agents. The user will think of views as places in the landscape of the program, sort of like a web page, and agents as active characters who can visit the places and accomplish tasks on their behalf. I think this provides us a with a flexible framework that will allow Chandler to address a wide variety of applications while giving users simple yet powerful ways to extend their will into the software and perform routine tasks automatically.

The rest of this paper discusses aspects of a potential agent framework for Chandler without attempting a detailed design. This is still a new, raw set of unproven ideas, with no prototyping or proof of concept work attempted yet, so we should expect it to evolve and mutate rapidly as we proceed with development.

Revision r1.1 - 07 Mar 2003 - 20:17 GMT - ChaoLam

The Agent Metaphor

A good user interface should find a way to offer lots of power and flexibility behind a simple facade. Its basis should illustrate some organizing principle that aligns complex options, yet aligns them with concepts already familiar to and understood by the user. The Chandler Agent Metaphor makes use of familiar occupational roles to represent the functionality of the program.

This metaphor empowers users because they can apply their lifetime of experience in the real world to how they want to utilize Chandler. Most users have more trouble manipulating abstract ideas than concrete objects. The realization of the agent metaphor should be highly visual if it's going to be effective for a broad range of users, since that makes it more concrete and real to them. We can use the graphical representation of an agent to both distinguish it from other agents and to convey the current details of it's internal state and portray ongoing activity.

Another strength of the agent metaphor is that it is inherently modular, since agents are autonomous, and distinct in the user's perception. It's easy to envision adding more members with different skills to your virtual staff.

Designing agents that generate and respond to events appears to be rich, deep and fun. Once the work begins, it almost seems to start designing itself.

Revision r1.1 -- 07 Mar 2003 - ChaoLam

AgentModularFramework

Chandler is based on a modular architecture, where plug-in modules containing Python scripts can add new functionality on the fly, without restarting the application. The modules typically inherit rich behavior from base classes in the Chandler core, so they can be relatively small themselves. The agent framework will leverage the modular architecture to allow agents to be developed and distributed independently from the core application, and to be added and removed smoothly at runtime.

A standard plug-in module in Chandler is called a Parcel. Parcels usually contain a set of related views, as well as the resources required for the views to render themselves. Parcels can even contain preference information, schemas, or additional data to augment the repository, so they can serve as the distribution format for Chandler applications.

Parcels will often contain one or more agents, sometimes as part of a set of related views. For example, a postman agent may be part of the email parcel. Other parcels may contain agents that are independent from any particular view. The user can also create new agents by cloning an existing agent, and then modifying it.

There are many other aspects of the framework that have to be carefully thought through. For example, agents can probably register with the framework declaring themselves suitable for a set of tasks, for example being called when the user issues a specific command like 'delete'. Agents should also contain a set of cryptographic credentials identifying them and allowing them to have specific privileges. Some more of the framework issues are discussed below, but the detailed design is beyond the scope of this document.

Revision r1.1 - 07 Mar 2003 - 20:22 GMT - ChaoLam

Agent Visualization

Chandler tries to portray as much of the agent's internal state as it can visually. Agents will typically be around 80 pixels square but they will also be rendered both smaller and larger as circumstances permit. They will use text, graphics and animation, and in rare cases sound, to portray their current status.
An agent's visual appearance will be based on a user selectable set of base images supplemented by agent-specific images overlayed onto the base image. Agents will display prominent tokens of their role, like hats or badges, to make their roles clearer and to help them be more easily distinguished from each other. Their graphical appearance is dynamic, changing to reflect their internal state, and their progress toward fulfilling their assigned tasks. We might want to render features with a vector-based format like Flash, which would potentially allow a wider range of expressions than bitmaps.

An agent will express its current level of satisfaction at carrying out its assigned tasks via its appearance, by appearing happy, sad, neutral, frustrated, etc. It will portray its activity level by appearing disabled, sleeping, alert, busy, bored, excited, etc. An agent will illustrate the completion state of the tasks at hand, and show the current step of a multi-step process. They will also be able to show magnitudes like none, one, some or many. Agents use animation to show when they are communicating and if they are currently sending, receiving or waiting for data.
The user will have some control of the appearance and style of their agents. The system will supply a variety of base appearances, from which the user can choose, and third parties will be able to add even more choices.

When you hover the mouse over an agent, a pop-up textbox appears (or possibly a speech balloon), containing even more information about the agent and it's current status. When you click on an agent, a dialog box appears to allow you to interact with it in detail.

Revision r1.1 - 07 Mar 2003 - 20:23 GMT - ChaoLam

Agent User Interface

Agents are integrated into the user interface in a number of different ways. There will be an optional "agent bar" stretching horizontally across the top of a Chandler window, below the other toolbars, that will display the relevant agents to the current view or activities, with the most relevant ones toward the left hand-side. The contents and order of the agents in the bar might change as you navigate between views, since some agents are specific to certain views or packages. There is also an Agents view available, which is a place that allows you to see all of your agents and manage them, as well as creating new ones and deleting agents that are no longer useful. Finally, agents can pop up in dialogs as necessary when conditions arise (see below)

An agent continuously reflects activity and progress via its appearance, but when the user lingers the cursor over it, it displays more detailed information in a pop up text box or speech balloon. When the user clicks on an agent, it pops up a dialog box for a more detailed interaction. Items from the view may also be dragged and dropped on an agent, which will respond according to its scripts.
At their most basic, agents offer users a number of commands that they can perform immediately, when the user selects one from a list, or at a specified time. Sometimes the commands require additional parameters, obtained by interrogating the user in a step-by-step, wizard-like manner. Some commands will complete quickly, but others may execute indefinitely in the background, while the agent offers progress feedback visually.

Agents also contain a set of event/action pairs, some of which are enabled by default. If a pair is enabled, the agent will execute the specified action when it receives the associated event. The user can choose to enable or disable the pairs, or to create new event/action pairs by selecting an event and action from lists.
One of the main functions of an agent is to notify the user when notable events occur, so they will need to get the user's attention with varying degrees of urgency. They will have multiple ways of engaging the user's attention, including altering their appearance, animating and changing colors, making sounds or popping up dialogs as appropriate.

Revision r1.1 - 07 Mar 2003 - 20:25 GMT - ChaoLam

Agent Runtime

Agents are Chandler modules, which contain an XML file and possibly some other resources including Python scripts that usually inherit from a base class defined by the agent framework, but might also inherit from other agent modules as well.

The Agent Runtime provides methods controlling the life-cycle of agents, which are usually accessed through inheritance. These include methods for creation, destruction, notification, cancellation, etc. There are also methods allowing agents to register with the framework to associate themselves with particular activities.

There might be some sort of sandbox-like restricted execution environment for the agent scripts if they're not fully trusted (see below), but otherwise an agent has access to the full resources of the client's Python environment, including performing database operations. There will be a facility for agents to ask the runtime for a description of the services that are available, including version numbers, so they can adapt themselves to multiple versions of the runtime.

One of the main functions of an agent is to generate and respond to events. Agents may call the agent runtime to define new types of events. They also invoke the runtime to signal events as necessary to notify other interested parties when the appropriate conditions arise. Agents can also subscribe to events generated by the system, or by other agents, to be notified each time the specified event occurs, without having to poll for it.

Events are queued by the runtime, and fed to the relevant agents asynchronously from the main thread of execution, so responses can be lengthy without interfering with the main task the user is focusing on in the foreground.

Revision r1.1 - 07 Mar 2003 - 20:26 GMT - ChaoLam

Agent Communications

Agents should be able to communicate with other agents, either running in the same application instance, or between different application instances on a local area network, or even across the Internet, in order for them to negotiate on the behalf of the users they represent. They need to be able to address one another globally, without requiring intermediating servers or fixed IP addresses. An interesting approach is to have them communicate using Jabber, so they can exchange XML-based structured instant messages in order to accomplish a variety of tasks.

Chandler may employ dozens of agents, so it wouldn't be prudent to require a unique Jabber ID for each agent. Instead, they can all use the user's Jabber ID and rely on the Jabber resource ID or an extension block to designate the intended agent. Since Jabber works by exchanging XML fragments, agents can carry out structured conversations using application-defined XML tags to accomplish their tasks. We can leverage the machinery for XML namespaces to provide a way for agents to declare the XML vocabularies that they support, and a way to query if a particular agent supports a particular vocabulary.

There should also be a way to discover the Jabber IDs of agents offering a particular service that you're interested in, and what protocols those agents support. The Chandler agent runtime will allow you to query what agents are registered with a particular Chandler instance, but there should also be a way to query across all known instances for agents that fulfill a particular requirement. Also, we could leverage Jabber chat rooms to creating meeting places for agents to congregate.

There also needs to be a mechanism for agents to converse when one of them is off-line. Chandler will probably have a general mechanism for store-and-forward communications that agent communications could utilize; one possibility is using email for store-and-forward agent conversations in a fashion analogous to the instant messaging approach under discussion, where text/XML MIME parts would contain the XML fragments.

Revision r1.1 - 07 Mar 2003 - 20:26 GMT - ChaoLam

Agent Log

Since agents can perform actions in the background without the explicit consent or awareness of the user, it's important to provide a way for the user to find out everything that's taken place. So, all actions performed by agents are logged for the user's inspection. Each agent has an individual log of the actions it has taken, and there's also a global log available in the Agents view where you can see everything that's happened. The logs will also be useful for debugging agents; there will probably be a verbose logging preference, with more details logged to facilitate agent debugging.

Revision r1.1 - 07 Mar 2003 - 20:27 GMT - ChaoLam

Agent Programming

Agents contain actions defined by scripts that can be executed at the user's request, or when the agent receives a specified event. There is a simple user interface for enabling or disabling predefined event/action associations, and creating new ones by associating an event with an action.

There will also be a way for power users to create new actions by combining existing actions and possibly using conditionals and iteration. Eventually, it will would be nice to have a simple, visual programming language with conditionals and iteration, possibly based on a subset of Python, but probably not in our first release.

Programmers will be able to add new types of events and actions to an agent, or create entirely new types of agents, by writing methods in Python.

Python, eh? Not AIML? (of course, if designing Agent Poindexter, one would have to write it in DAML, yes?)

Revision r1.2 - 13 Mar 2003 - 22:09 GMT - JonathanSmith

Agent Reputation

Since agents may be independently developed plug-in modules, there will eventually be a wide variety of agents available from a wide variety of sources. Some agents might be unruly, or even malicious, either by accident or intent. We should provide a reputation mechanism for rating agents and sharing your ratings with other users.

Chandler's UI should provide an easy way to rate an agent by picking attributes from a list, as well as capturing free-form text feedback. The ratings will be kept in the user's repository and also conveyed to a server that associates the ratings with agents via their MD5 digest. Chandler can query the server to determine the average ratings associated with an agent and display them to the user. We might also want to include a peer-to-peer way of querying ratings to obviate the need for a server.

See also SharingIssues -- ChaoLam - 07 Mar 2003

Revision r1.1 - 07 Mar 2003 - 20:31 GMT - ChaoLam

Security Issues

Since Chandler agents are independently distributed plug-in modules, capable of performing arbitrary operations on a data repository and sending the results across the network, there are lots of security issues to consider.

Agents should carry a set of credentials, authenticated by digital signatures, declaring the owner of the agent and granting it specific privileges. One possibility is using a relatively fine-grained capability-based security architecture where agents must possess specific capabilities to perform specific operations; the relevant capabilities would be passed as parameters to the restricted methods. Another possibility is restricting access to methods or primitives at the Python language or library level, executing the agents in a restricted runtime influenced by the agent's credentials. The reputation mechanism mentioned above will also help guard against rogue agents.
It's too early in the design cycle to commit to specific security mechanisms, since we're still not sure how agents will be used, but it's clear that we'll have to pay close attention to security issues as development proceeds.

See also SecurityIssues

Revision r1.1 - 07 Mar 2003 - 20:37 GMT - ChaoLam

Sample Agents

It's easy to think of lots of potential agents; it will be tricky to decide the best set of them to include in the initial implementation. Here are some brief descriptions of potential agents to consider implementing, but I'm sure the list will be revised as our thinking evolves:

- Postman - collect and classify email
- Secretary - arrange meetings between multiple parties
- File Clerk - file items and enforce filing policies
- Janitor - system housekeeping tasks, warn about resource limitations
- Sleuth - search for items, both locally and remotely
- Switchboard Operator - arrange conference calls or IM meetings
- Social Director: manager personal calendar, arrange recreational activities
- Security officer: enforce security policies
- Travel agent: book airline and hotel reservations for a business trip

Revision r1.2 - 09 Mar 2003 - 19:16 GMT - JonathanSmith