OPEN SOURCE APPLICATIONS FOUNDATION

Repository Access Protocol API (RAP)

<OSAF Current Thinking -- 22 Feb 2003>

Draft February 22 2003

The general purpose of RAP is to provide network access to a Chandler repository. A repository can be generalized as a general purpose database consisting of items with an arbitrary number of name value pairs. We were unsatisfied with currently available access protocols primarily due to network inefficiencies and/or complexity issues. We have decided to model RAP on IMAP but using a more generalized and powerful API. Since extending IMAP is not an option due to a large number of email only assumptions within it, we will be building an entirely new protocol, but borrowing some IMAP concepts.

The core transport layer will be based on The Blocks Extensible Exchange Protocol Core (BEEP), as specified in RFC3080. BEEP provides framing, SASL, TLS and very importantly, parrallelism. BEEP will allow RAP to perform multiple repository queries over a single authenticated connection and return answers as they are available. This represents a significant improvement over IMAP as it exists today.

The following API is presented in Python syntax. It is not intended to act as a specification, since we are not even close to finished. We will be working on a specification over the next 6 months. This API should give the reader a strong idea of the scope and direction of the RAP protocol, but it is far from complete.

class RAP([*host*[, *port*]])

This class implements the actual RAP protocol. The connection is created and version and capabilities are determined when the instance is initialized. If *host* is not specified, 'localhost' is used. If *port* is omitted, a standard port, as yet undefined, is used.

Exceptions are defined as attributes of the RAP class:

exception RAP.error

Exception raised on any errors. The reason for the exception is passed to the constructor as a string.

exception RAP.abort

RAP server errors cause this exception to be raised. This is a sub-class of RAP.error.

exception RAP.interrupt

A application triggered interrupt causes this exception to be raised. This is a sub-class of RAP.error.

exception RAP.connection_closed

The connection has been closed by the close_connection() method.

exception RAP.connection_lost

The connection has been disconnected due to loss of the network or an abnormal server termination.

* MORE EXCEPTIONS HERE IN THE FUTURE *

A RAP instance has the following methods:

close_repository()

Close the currently selected repository. * NOT CURRENTLY KNOW HOW TRANSACTIONS ARE HANDLED AT CLOSE TIME. MY GUESS WOULD BE TO FORCE A TRANSACTION AT CLOSE *

Arguments: NONE

Returns: NONE, throws exception on error

close_connection()

Close the connection. Implicitly calls close_repository(), if one is selected. After close_connection() has been called no other repository access functions can be called, if they are called, they will return an exception.

authenticate(*func*)

Authenticate command -- requires response processing.
NOTE: At this point I don't know if we need this method. BEEP provides SASL authentication at a level below RAP, but we may want to be able to have multiple identities per connection.

Arguments: a function for querying string responses from the user.

Returns: NONE, throws exception on error

capabilities()

Arguments: NONE

Returns: a list of repository objects each describing an optional or extention based capability of the currently connected repository server. The list of capabilities is global for all repositories on the server.

select(*repository_name* **)**

Arguments: *repository_name* specifies the name of a repository to be selected. All further operations will be performed on this repository.

Returns: Nothing on successful selection. If another repository is currently selected, this function implicitly calls close_repository() on the currently selected repository. Throws exception on error.

search(*search_type*, *search_arguments,* **[***attribute_list***], [** *retrieval_flags* **]**)

Arguments: *search_type* specifies the type of search to be performed. *search_arguments*represents a list of necessary data to perform the search. *attribute_list* is optional and specifies which attributes of the objects returned from the search are returned. Keywords "ALL" or "NONE" may be used in place of an *attribute_list*. If "NONE" is used, the UID is still returned. If *attribute_list* is omitted only the UID of each object is returned.

retreival_flags is a list of name value pairs that specifies several modifiers that effect the result set returned. *retrieval_flags* can contain the following:

retreive_names_flag: TRUE | FALSE - specifies that all the attribute names should be returned, even if we are not requesting all of the values. This is useful when building a partial object cache so that you can tell when you need to fetch more attributes or if your object is complete.
values_smaller_than: INTEGER - specifies that values equal to or larger the specified size are not returned. Values will be marked accordingly and their size returned
partial_value_range: pair of INTEGERS - specifies a byte range. The byte range is used to return parts of values. If an entire value fits within the range, the complete value is returned and marked accordingly. If the object is larger than the range or the range does not begin at zero, a partial object is returned and marked accordingly. The size of the entire value is returned as well.
limit_result_set: pair of INTEGERS - specifies a range. The range is used to return parts of a result set. If the entire result set fits within the range the whole set is returned, otherwise a partial result set is returned. The size of the entire set is returned as well.

Returns: an iterator object to a list of objects containing the specified attributes. If a given object did not contain an attribute named in the *attribute_list* the attribute will not be part of the returned object.

search_types have yet to be defined, but should include the capability to sort the result set for retrieval.
* Have not yet specified how the objects returned notify the caller that an attribute name is present, but the value has yet to be retrieved *
* Several questions about how to return all the partial value and limited result set information need to be answered *

retrieve(*uid_list,* **[***attribute_list***], [** *retreival_flags* **]**)
Arguments: *uid_list* specifies any positive number of UIDs for retrieval. *attribute_list* is optional and specifies which attributes of the objects listed are to be returned. Keywords "ALL" or "NONE" may be used in place of an *attribute_list*. If "NONE" is used, the UID is still returned. If *attribute_list* is omitted all attributes are returned. *retreive_names_flag:* see the definition above under **search**()

Returns: an iterator object to a list of objects containing the specified attributes. If a given object did not contain an attribute named in the *attribute_list*the attribute will not be part of the returned object.
* WHAT DO WE DO ABOUT INVALID UIDs? *

put(*list_of_objects*)
Arguments: *list_of_objects* is a list of objects containing a UID and any number of attributes per objects. There is no size restriction on object length or number of objects. If an object already exists in the database, the attributes in the put command will overwrite attributes in the existing object. Existing attributes not included in the put will not be deleted. This allows for the rapid

update of a small number of attributes within a large set of objects.

Returns: NONE, throws exception on error

delete (*uid_list,* **[***attribute_list***]**)
Arguments: *uid_list* specifies any positive number of UIDs for deletion. *attribute_list* is optional and specifies which attributes of the objects listed are to be deleted. If *attribute_list* is omitted all attributes are deleted and the object itself is deleted.

transaction("SAVE" | "REVERT")
Arguments: keywords "SAVE" or "REVERT"

Returns: NONE, throws exception on error

Class RAP_iterater()

RAP_iterater has the following methods:

next(**[***number***]**)
Arguments: *number* is optional and specifies the number of objects to be returned. If not specified the default is one (1)
Returns: a list of repository objects.

all()
Arguments: NONE

Returns: a list of repository objects containing all objects within the iterator. Or if some have already been returned via **next()** only the remaining objects are returned.

abort()
Arguments: NONE

Returns: nothing. The current iterator will no longer be valid, all further network data associated with the iterator will be cancelled.

Issues with current spec:

* need negotiable transfer encodings. BEEP will provide this as part of the channel start process, but I haven't worked out the details.
* need some thoughts about per attribute authorization
* need to fill in alot of details around object retrieval

-- LouMontulli - 22 Feb 2003

<End OSAF Current Thinking>

Questions and Answers

locking (for multi-user concurrent access)

- **Question:** Will a RAP client be able to lock an item in a repository? (BrianDouglasSkinner - 26 Feb 2003)

- **Answer:** This is still under discussion, but likely not. I would expect to see a system more like CVS where changes are merged in rather than held locked. Some locking would be necessary during writes and merges, but otherwise not. (LouMontulli - 27 Feb 2003)

change notification

- **Question:** Will RAP provide some kind of mechanism to let a RAP client "subscribe" to get notified about changes to an item? (BrianDouglasSkinner - 26 Feb 2003)

- **Answer:** Yes, definately. A database trigger mechanism is being designed now, and RAP will reflect that functionality. RAP is going to use BEEP for a transport layer which supports asyncronous messaging which allows for notifications. There are some significant challenges though:
    - What happens to notifications when a client isn't connected and can't be reached?
    - How do large servers deal with triggers? It may be unworkable to require a large server to support triggers. In this case what other mechanism can the client use? (LouMontulli - 27 Feb 2003)

change notification

- **Question:**
    - For example, let's say a RAP client uses **search** or **retrieve** to get an event in a calendar, and then the event is displayed to some user (Pat) in the UI. If another user (Chris) on another machine changes the end time of the event, how does the UI code on Pat's client find out that the event needs to be redisplayed? Would it be good to have some kind of mechanism that allowed a RAP client to **automatically** subscribe for notifications about all of the items that the RAP server has returned to the client?
    - Or, not even considering issues about multi-user concurrent access, let me just offer a single-user example. Say I'm using Chandler, and I have a couple views open: a calendar day view and a calendar week view. If I create a new event in the day view, how does the week view get notified that there now exists a new event it should be displaying?
    - And, if we are talking about general mechanism for managing notifications about changes to query results, then is that related to the issue that people are talking about in these posts on the design list: lists: "Knowing when you've read the most recent e-mail in a mailbox" and lists: "Recognizing a response, and group filters". (BrianDouglasSkinner - 26 Feb 2003)

- **Answer:** This is a great example. There are multiple ways this could be implemented. I suspect that we will need to use a few of these methods in order to deal with firewall issues:
    - If a RAP connection is active, database change notifications could be sent
    - triggers could be used to send a notification via:
        - RAP
        - a jabber message
        - email
    - The client could periodicaly poll the server to look for changes. (LouMontulli - 27 Feb 2003)

database triggers

- **Question:** Will the RAP API allow a client to subscribe to be notified when an arbitrary database condition is met?
    - On the Extensibility page, there are a couple bullet points the need for trigger and scripts, "Chandler can detect when an arbitrary database condition is met" and "Execution of a script whenever a particular condition is triggered"? Are there some scripts that run on the server, and others that run on the client? Does the server ever notify a client that it should run a script? (BrianDouglasSkinner - 26 Feb 2003)

- **Answer:** Triggers will be as arbitrary as possible, but we will not support a turing complete language for triggers. Because triggers run on remote servers we cannot allow arbitrary code to run there or else the security of all users could be compromised. (LouMontulli - 27 Feb 2003)

delete semantics

- **Question:** What are the semantics of the delete method?
    - If a RAP client deletes an item that represents an entire calendar, do all of the events that are "in" that calendar get deleted?
    - Is there some notion of garbage collection for items that are "orphaned" as the result of a deletion?
    - What happens when one RAP client makes changes to a calendar that another RAP client just deleted?
    - How is referential integrity handled? If item A has an attribute that points to item B, what happens to item A when item B gets deleted?
    - Does the repository keep a change log for each item, or a series of item versions? Using the **delete** method, when an item is deleted, does that mean that item is just marked as deleted, or is item actually erased on disk, along with all of the previous versions and/or change log entries? (BrianDouglasSkinner - 26 Feb 2003)

- **Answer:** These issues are still being discussed, and depend more on the database than RAP. We would like to use a database with garbage collection to make things easier. (LouMontulli - 27 Feb 2003)

Discussion

Comment on "class RAP ([host[, port]])": A user might want multiple repositories on their localhost for testing purposes. Suppose that someone has made some schema changes and wants to test these on new repository before making changes to their "working" repository. (PaulBHill 3/18/03)

Comment on Capabilities: I don't think that capabilities should be global for all repositories on the server. If you are running on multiple port numbers the capabilities should be specific to the port that the client connected to. Again, suppose that you have two repositories that have different schemas, capabilities might end up reporting information about the schema requirements or version... (PaulBHill 3/18/03)

Question about triggers: Will triggers need to be replicated themselves or will it just be enough that the data generated (if any) by the trigger's execution will be replicated? The reason I ask is that I sense that triggers will essentially be callbacks to a parcel by the data api. If so, then triggers will be unable to run on anything except the computer where the parcel has been installed on. MikeT - 21 May 2003
