ADD OR UPDATE A DYNAMIC LINK BETWEEN TWO SERVICES

A link can be used to associate two services based on some criteria and represents a reference to some type of object or resource. The links can be created or destroyed dynamically depending on the system use. With the link, there is a descriptive path, which is made up of a list of concepts related to the link. This concept list is arbitrary and can represent anything, keywords in a query, for example. If you update a link then a weight value associated with it can be reinforced. If the weight value passes a threshold, then the link can be considered as reliable. If the link exists (path description and URI) but the association subsequently does not, the link can be decremented again until it gets removed. The 'licasArchitecture' guide, book, or 'licas' research papers describe this further.

CREATE/ADD/UPDATE DYNAMIC LINK

If you add/update a dynamic link using the service methods, if the link already exists, then its weight value can be increased. The purpose of the links is to represent the current situation by changing dynamically over time. There can be several sources that relate to a particular path, but over time, one or other of them may become more popular. The linking structure therefore includes a descriptive path and a set of sources. For an update, the service then retrieves all sources that it has stored under that path. If a stored source reference is part of the current update, the source weight can be incremented. If it is not part of the current update, the weight value can be decremented. So adding or updating link information will only reinforce the current result set and decrement all of the other ones that are part of the same path structure. You only need to add a linking service to your own service and then invoke the `addRelatedSources` method on it to maintain the dynamic links.

```
//some variables that might be required
String serverPassword;                  //server password
String servicePassword;                 //service password
Element serverUri;                      //server uri
Element serviceUri;                     //service uri
ArrayList sources;                      //list of source references
ArrayList concepts;                     //list of concepts in the link path
ArrayList negativeConcepts;             //list of concepts to block a link
MethodInfo methodInfo;                  //method info
CallObject callObject;                  //call object

//you need to know what address the remote server is running on
serverUri = remote server handle to call.

//you can create the service uri from the server uri and the service uuid as follows
serviceUri = Handle.createNewUrlHandle(Handle.getURI(serverUri));
serviceUri = Handle.addToHandle(serviceUri, Handle.asHandleElement(serviceUuid));
```

```
serviceUri = Handle.addToHandle(serviceUri,
                                Handle.asHandleElement(ServiceConst.LINKER));
servicePassword = passwordHandler.getServicePassword(serviceUuid); //or serviceUri

//a hopefully simpler way to add the links is through single lists for each path separately
//this is the path description in order, for example query conditions, or metadata values
concepts = new ArrayList();
concepts.addElement(concept1);
concepts.addElement(concept2);

//retrieve the references to the sources that you want to include as links
//this could be a direct reference or a uri path, or mostly, a Handle description
sources = new ArrayList();
sources.addElement(source URI 1);
sources.addElement(source URI 2);

//negative concepts is part of a test case and can usually be empty
negativeConcepts = new ArrayList();

//make a method call to add the new linking information
methodInfo = new MethodInfo();
methodInfo.setName(MethodConst.ADDRELATEDSOURCES);
methodInfo.setRtnType(TypeConst.ARRAYLIST);
methodInfo.setServerURL(serverUri);
methodInfo.setServiceURI(serviceUri);
methodInfo.setServerPassword(serverPassword);
methodInfo.setPassword(servicePassword);
methodInfo.addParamInfo(sources);
methodInfo.addParamInfo(concepts);
methodInfo.addParamInfo(negativeConcepts);
call.call(methodInfo);
```

This should create a structure like: concept1->concept2->all source references

RETRIEVE DYNAMIC LINK INFORMATION

You can then ask a particular service if it has any dynamic links that relate to a certain linking path, or list of concepts. Only the links that are at the top level (pass the top threshold value, see the papers) in the linking structure will be returned. You use the `getLinkedSources` method to do this. Other more local or admin-related methods can retrieve the whole structure.

```
//some variables that might be required
String serverPassword;          //server password
String servicePassword;         //service password
Element serverUri;              //server uri
Element serviceUri;             //service uri
ArrayList sourcePaths;                  //list of source paths
ArrayList concepts;             //list of concepts in the link path
```

```
ArrayList negativeConcepts;          //list of concepts to block a link
MethodInfo methodInfo;               //method info
CallObject callObject;               //call object


//initialise the call invocation object and retrieve the server passwords.
//the server URL is appended with Const.HTTPSERVER as part of the handle address.
callObject = new CallObject();
serverUri = remote server to call.

//you can create the service uri from the server uri and the service uuid as follows
serviceUri = Handle.createNewUrlHandle(Handle.getURI(serverUri));
serviceUri = Handle.addToHandle(serviceUri, Handle.asHandleElement(serviceUuid));
serviceUri = Handle.addToHandle(serviceUri,
                                    Handle.asHandleElement(ServiceConst.LINKER));

//add the concepts in the link path
concepts = new ArrayList();
concepts.add(concept1);
concepts.add(concept2);

//make negative concepts empty for default use
negativeConcepts = new ArrayList();

//also need the server uri and password, and the service password.
//Other code examples show how these can be saved and retrieved from the PasswordHandler
serverPassword = passwordHandler.getServicePassword(serverUri);

//the password can be saved under different formats – local uuid or full handle address
servicePassword = passwordHandler.getServicePassword(serviceUuid);

//make the method call to retrieve any link information
methodInfo = new MethodInfo();
methodInfo.setName(MethodConst.GETLINKEDSOURCES);
methodInfo.setRtnType(TypeConst.ARRAYLIST);
methodInfo.setServerURL(serverUri);
methodInfo.setServiceURI(serviceUri);
methodInfo.setServerPassword(serverPassword);
methodInfo.setPassword(servicePassword);
methodInfo.addParamInfo(concepts);
methodInfo.addParamInfo (negativeConcepts);
sourcePaths = (ArrayList)callObject.call(methodInfo);
```

The mechanism works exactly as expected. If, for example, you increment by 0.1 three times and the threshold is 0.25, then it should become a reliable top level source. If you then decrement once, it should be moved down a level and not be returned.