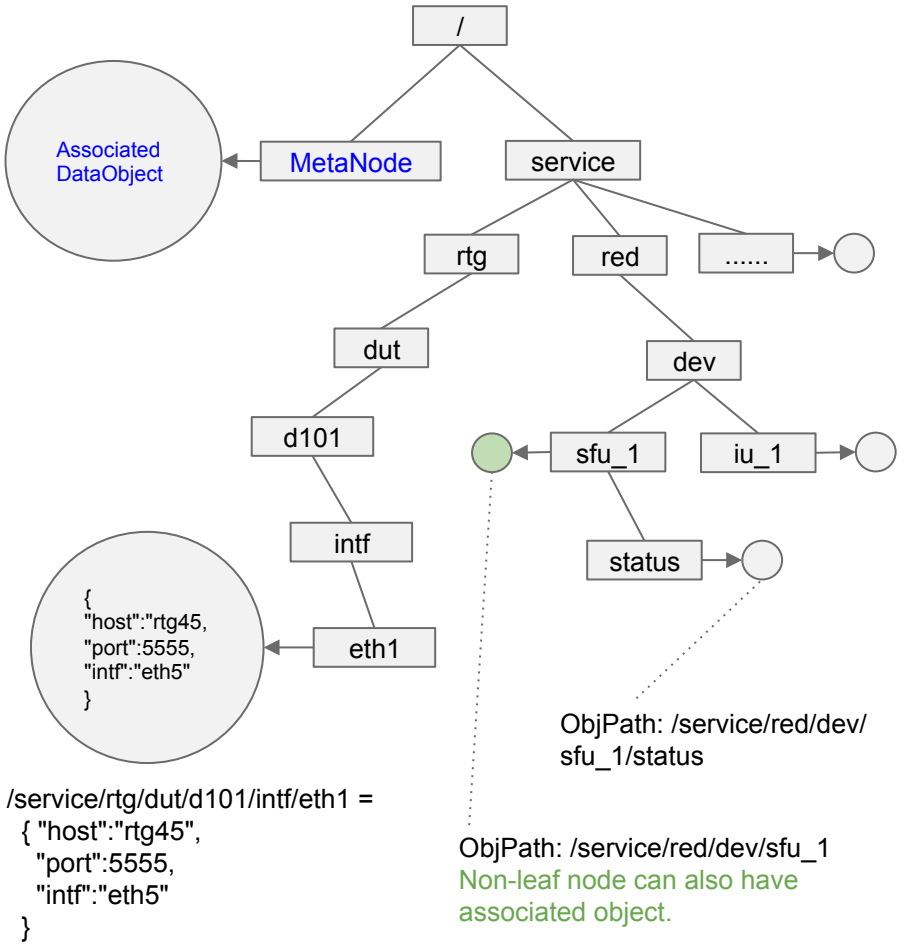
ObjXp

Obex Tree

ObexTree

- ObexTree is composed of MetaNodes and associated DataObjects.
- Each MetaNode can have at most one SPtr to its associated DataObject. (The associated data object can be a collection type such as set, list, vector, map, etc.)
- Not only leaf node but also non-leaf node can have associated object.
- User doesn't have to care about MetaNode which will be created/updated/removed transparently by the framework.



ObexObject

- ObexObject is the base class of all ObexTree associated data objects.
- User can extend the ObexObject and make subclasses to customize user-specific data.
- A MetaNode can hold any subclass type of ObexObject. The type can be converted using DynamicPointerCast or StaticPointerCast at use time.
- Subclasses may override the following methods:
  - virtual String marshal() - serialize the object to octet string
  - virtual void unmarshal(String& data) - deserialized octet string to obex object
  - virtual String toString() - human readable description of the object
  - virtual bool equals(ObexObject& obj) - compare with a given object if it's equal or not

  The usage of the methods will be further explained later.

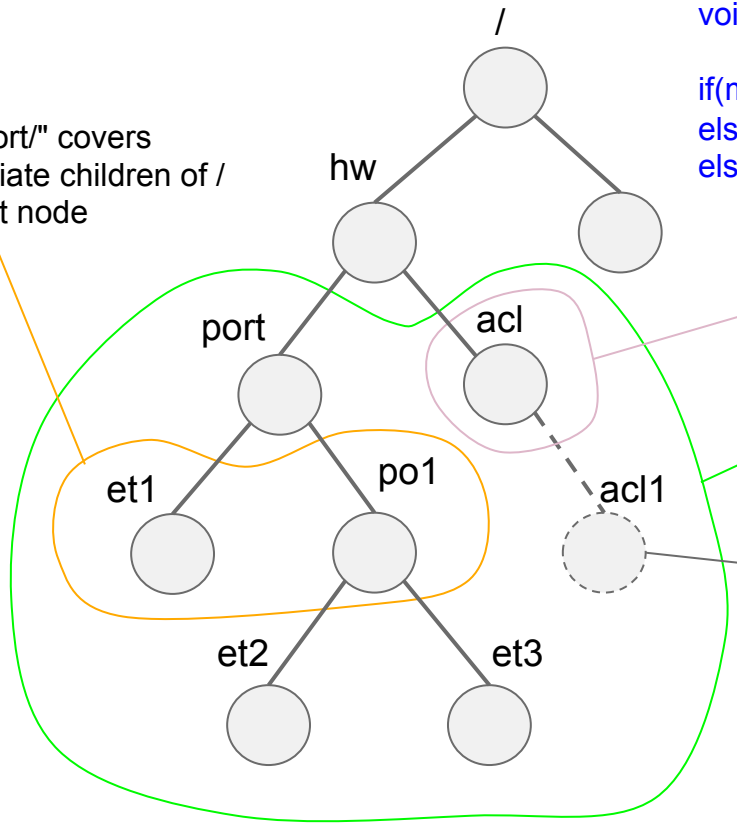
ObexTree Operations

- void putObject(String path, SPtr<ObexObject> oObj);
- SPtr<ObexObject> getObject(String path);
- void delObject(String path);
- void registerCallback(String path, SPtr<ObexCallback> cbObj);
- void unregisterCallback(String path, SPtr<ObexCallback> cbObj);
- SPtr<ObexObjectMap> find(String topDir, String matchStr);

ObexCallback Interface

```
YAIL_BEGIN_INTERFACE(ObexCallback)
        // Friendly name to identify the callback object
        virtual String callbackName() = 0;

        // Called when object is created or updated
        virtual void onUpdated(String path, SPtr<ObexObject> newObj, SPtr<ObexObject> oldObj) = 0;

        // Called when object is deleted
        virtual void onDeleted(String path, SPtr<ObexObject> oldObj) = 0;
YAIL_END_INTERFACE
```

ObexTree registerCallback() - Matching Scope



"/hw/port/" covers immediate children of /hw/port node

void registerCallback(String matchStr, SPtr<ObexObject> callbackObj);

if(matchStr is ending with "/*") { match all descendants }
else if(matchStr is ending with "/") { match immediate children }
else { do exact match }

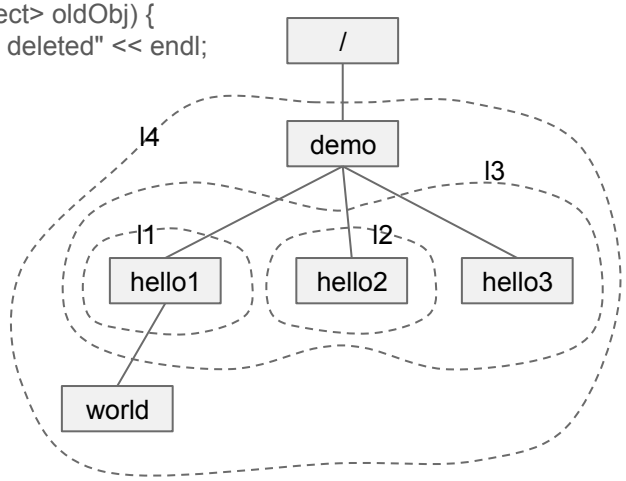
"/hw/acl" matches this node only

"/hw/*" covers all descendants of /hw node

/hw/acl/acl1 is not in the tree at the moment but the callbacks can be registered even before the object is put to the tree.

DemoListener

```
YAIL_BEGIN_CLASS(DemoListener, EXTENDS(YObject),
                                IMPLEMENTS(ObexCallback))
  public:
    void init(String name) { name_ = name; }

    String callbackName() { return name_; }
    void onUpdated(String path,
          SPtr<ObexObject> newObj, SPtr<ObexObject> oldObj) {
      cout << path << " is updated" << endl;
    }
    void onDeleted(String path,
          SPtr<ObexObject> oldObj) {
      cout << path << " is deleted" << endl;
    }
  private:
    String name_;
YAIL_END_CLASS
```




```
int main(int argc, char** argv) {
  SPtr<ObexTree> myTree = CreateObject<ObexTree>();
  SPtr<DemoListener> l1 = CreateObject<DemoListener>("l1");
  SPtr<DemoListener> l2 = CreateObject<DemoListener>("l2");
  SPtr<DemoListener> l3 = CreateObject<DemoListener>("l3");
  SPtr<DemoListener> l4 = CreateObject<DemoListener>("l4");

  myTree->registerCallback("/demo/hello1", l1);
  myTree->registerCallback("/demo/hello2", l2);
  myTree->registerCallback("/demo/", l3);
  myTree->registerCallback("/*", l4);

  SPtr<ObexStringObject> demoObj =
       CreateObject<ObexStringObject>("Hello World!");

  myTree->putObject("/demo/hello1", demoObj); // trigger l1, l3, l4
  myTree->putObject("/demo/hello2", demoObj); // trigger l2, l3, l4
  myTree->putObject("/demo/hello3", demoObj); // trigger l3, l4
  myTree->putObject("/demo/hello1/world", demoObj); // trigger l4

  myTree->delObject("/demo/hello1");
  myTree->delObject("/demo/hello2");
  myTree->delObject("/demo/hello3");
  myTree->delObject("/demo/hello1/world");

  return 0;
}
```

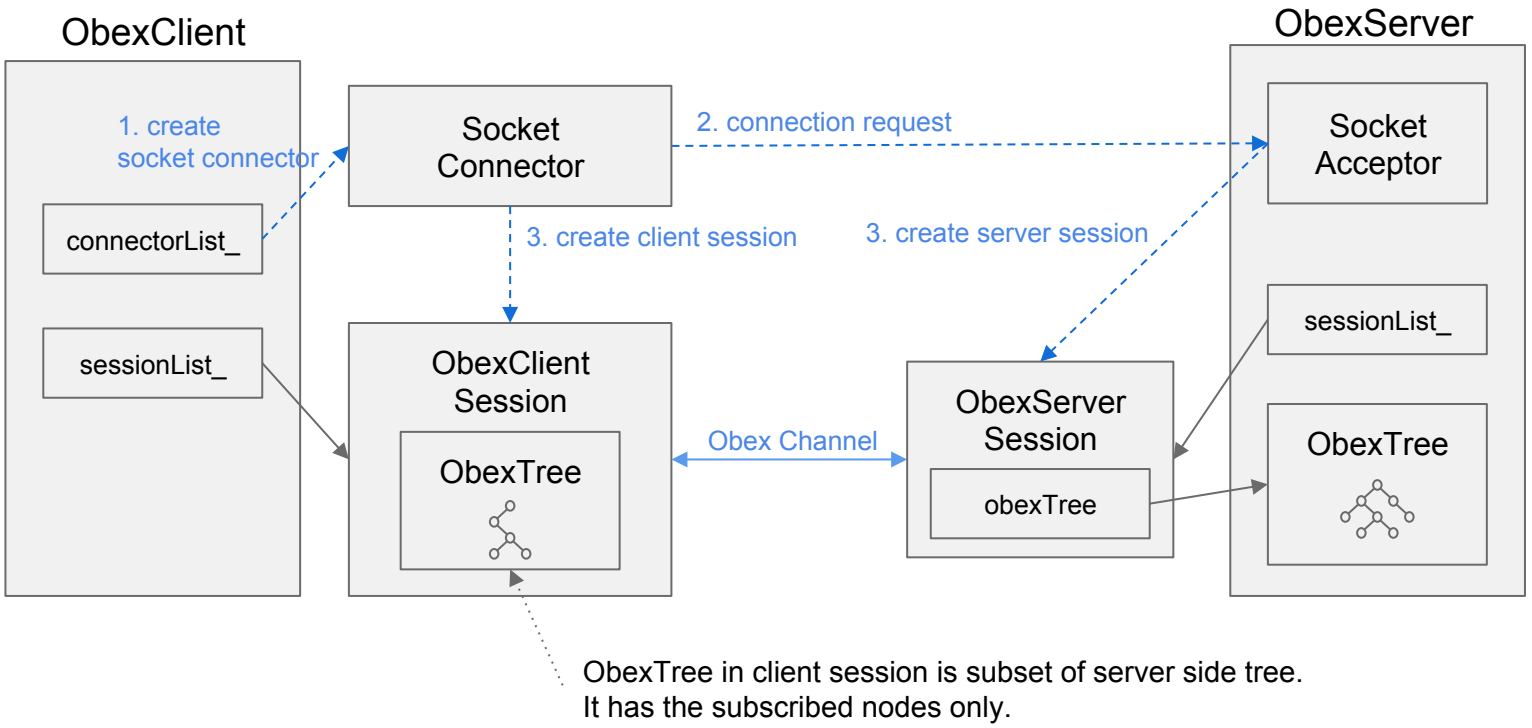
Object Path Util - MkYPath(), MkYPathArg()

```
// Path Template Format: ${VAR_NAME[=DEFAULT_VALUE]}

// See "${YHOME}/yail/include/YPath.h" for more detail

#define YpRtgTopo__DUT__INTF   "/service/rtg/topo/${DUT=d101}/intf/${INTF}"

String objPath1("/service/rtg/topo/d101/intf/eth1");

String objPath2 = MkYPath(YpRtgTopo__DUT__INTF, "DUT=d101", "INTF=eth1");

String objPath3 = MkYPath(YpRtgTopo__DUT__INTF, "INTF=eth1", "DUT=d101"); // Args order doesn't matter

String objPath4 = MkYPath(YpRtgTopo__DUT__INTF, "INTF=eth1"); // Use default value for DUT

String objPath5 = MkYPath(YpRtgTopo__DUT__INTF, MkYPathArg(DUT, "d" << 101), "INTF=eth1");

// objPath1 == objPath2 == objPath3 == objPath4 == objPath5
```

ObjectPath Util - ParseYPath(), GetYPathArg()

```
YPathArgVector  tokens;

ParseYPath(tokens, "/service/rtg/topo/d101/intf/eth1");

// tokens.size() == 6, tokens[0] == "service", tokens[3] = "d101", tokens[5] == "eth1"


YPathArgMap args;

ParseYPath(args, YpRtgTopo__DUT__INTF, "/service/rtg/topo/d101/intf/eth1");

// args.size() == 2, GetYPathArg(args, "DUT") == "d101", GetYPathArg(args, "INTF") == "eth1"
```

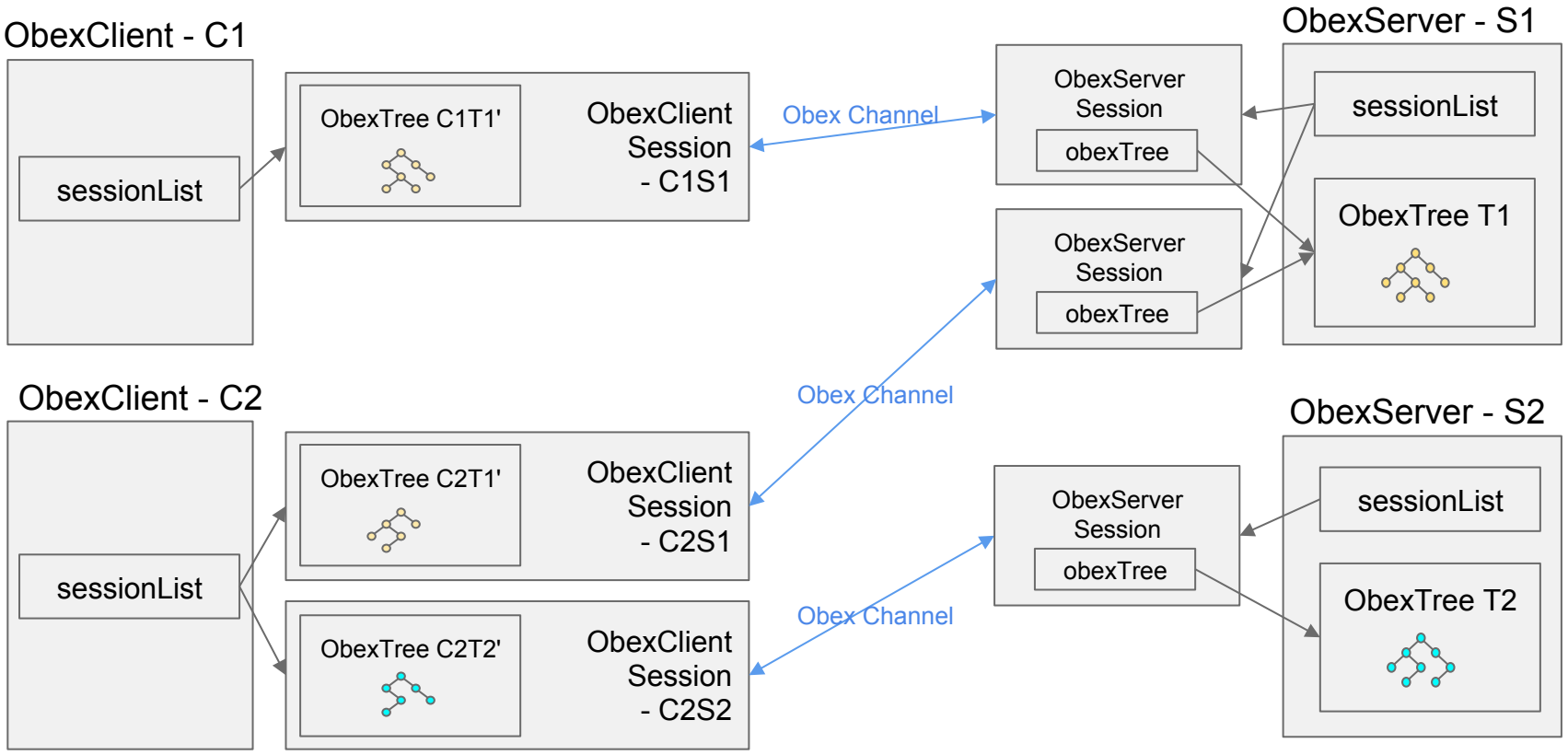
Obex Client-Server Extension (1)



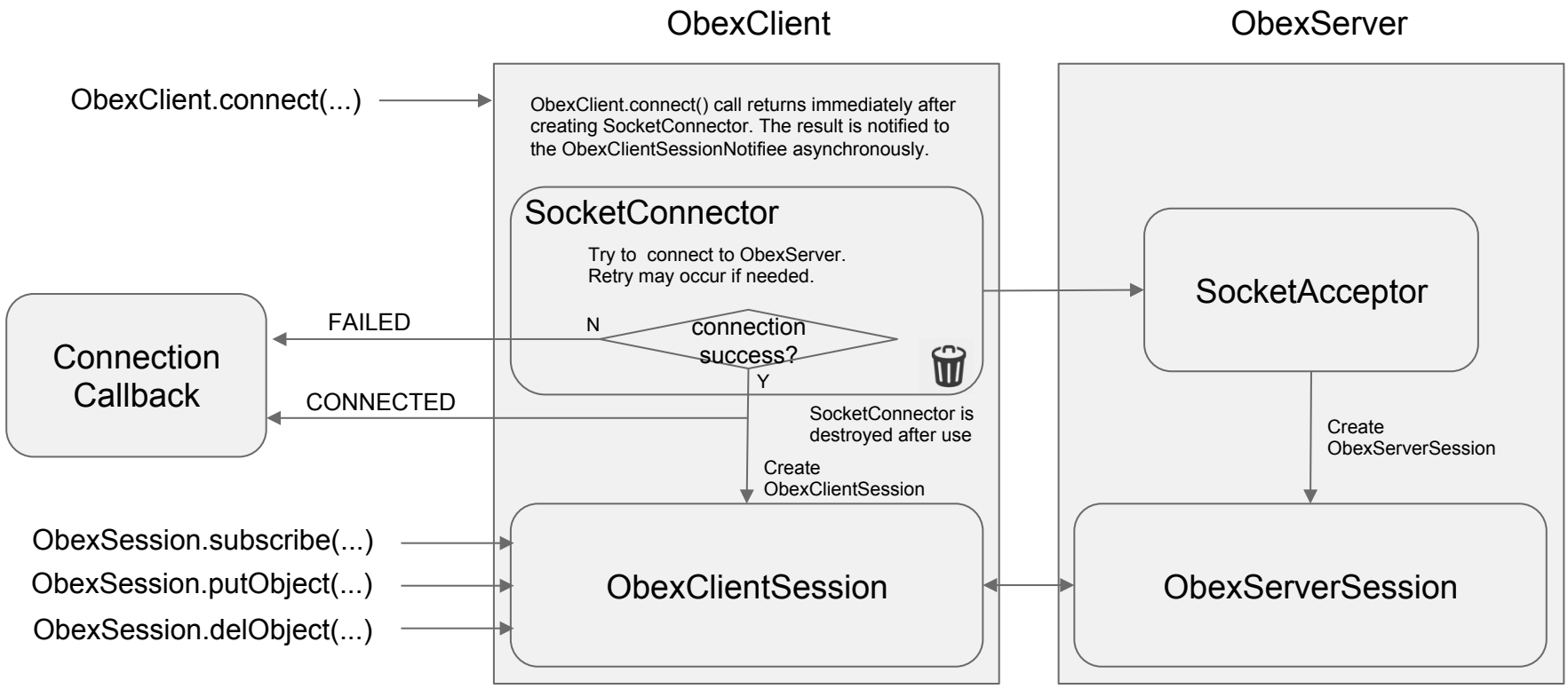
Obex Client-Server Extension (2)

- ObexClientSession and ObexServerSession pair makes the remote ObexTree, which is maintained by ObexServer, look like a local tree from ObexClient's standpoint.
- ObexClientSession receives object manipulation requests from user
- ObexServerSession acts like a 'Proxy'

Multi-site Client-Server Model



Connection Procedure



ObexServer

- Keeps UnixSocketAcceptor and/or TcpSocketAcceptor
- Accepts connection request from ObexClient
- Creates ObexServerSession per successful connection

- void init(String serverName, String sockName, int svcPort, SPtr<IoService> ioSvc, bool fwdOnly=false);
  - serverName: name to identify this ObexServer object
  - sockName: if not empty, opens a unix socket under /var/yail/<sock/sockName> and accepts incoming connections
  - svcPort: if non-zero, opens a tcp socket on this port and accepts incoming connections
  - ioSvc: IO service scheduler this server will be registered with.
  - fwdOnly: if true, data objects are kept in serialized form. No marshal/unmarshal is done. (Will be explained later.)

ObexServer

- Default Unix Socket Path: /var/yail/sock

- Override with the shell env variable: YAIL_SOCK_PATH

bash$ export YAIL_SOCK_PATH=${HOME}/tmp

ObexClient API

- void init(String clientName, SPtr<IoService> ioSvc, SPtr<ConnectionCallback> connectionCallback)
  - clientName: the name to let server know who this client is.
  - ioSvc: IO service scheduler this client will be registered with.
  - connectionCallback: callback object to which the connection results will be notified.
- void connect(String sessionKey, String dest, int retryPeriod, int retryLimit)
  - sessionKey: key to identify the session.
  - dest: where to connect. "ipAddr:port" for TCP, or unix socket at "/var/yail/sock/<dest>"
  - retryPeriod: retry interval in millisecond. 0 means no retry.
  - retryLimit: retry count until giveup. 0 means unlimited.
- void connect(String sessionKey, SPtr<ObexServer> cohabitingServer)
  - can directly connect to the ObexServer cohabiting in the same process.
- void disconnect(String sessionKey)
- SPtr<ObexClientSession> getSession(String sessionKey)

Cohabiting ObexClientSession

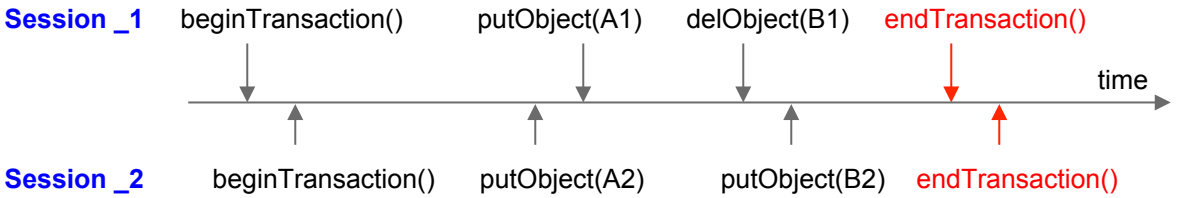
- If ObexServer is running in the same process space, directly access to the ObexTree of the ObexServer is possible without a real socket connection. A dummy client session called "Cohab Client Session" can support this.
- Provides transparency about the location of ObexServer.

ObexClientSession API

- Same API as ObexTree
  - void putObject(String path, SPtr<ObexObject> oObj, bool delOnDisconnect=false)
    - optional argument ‘delOnDisconnect’ - the object will be automatically deleted when the client session is disconnected.
  - void delObject(String path)
  - SPtr<ObexObject> getObject(String path)
  - void registerCallback(String path, SPtr<ObexCallback> cbObj)
  - void unregisterCallback(String path, SPtr<ObexCallback> cbObj)
  - SPtr<ObexObjectMap> find(String topDir, String matchStr)

ObexClientSession API

- Additional API
  - void subscribe(String paths[, SPtr<SubscriptionCallback> callback])
    - multiple paths separated by comma(,).
    - Callback will be called on sync completion for the subscription.
  - void disconnect()
  - void beginTransaction(), void endTransaction(), void abortTransaction()
    - The putObject(), delObject() operations will take effect all at the same time when endTransaction() is called.
    - The delOnDisconnect option in putObject() API is still supported during transaction processing.



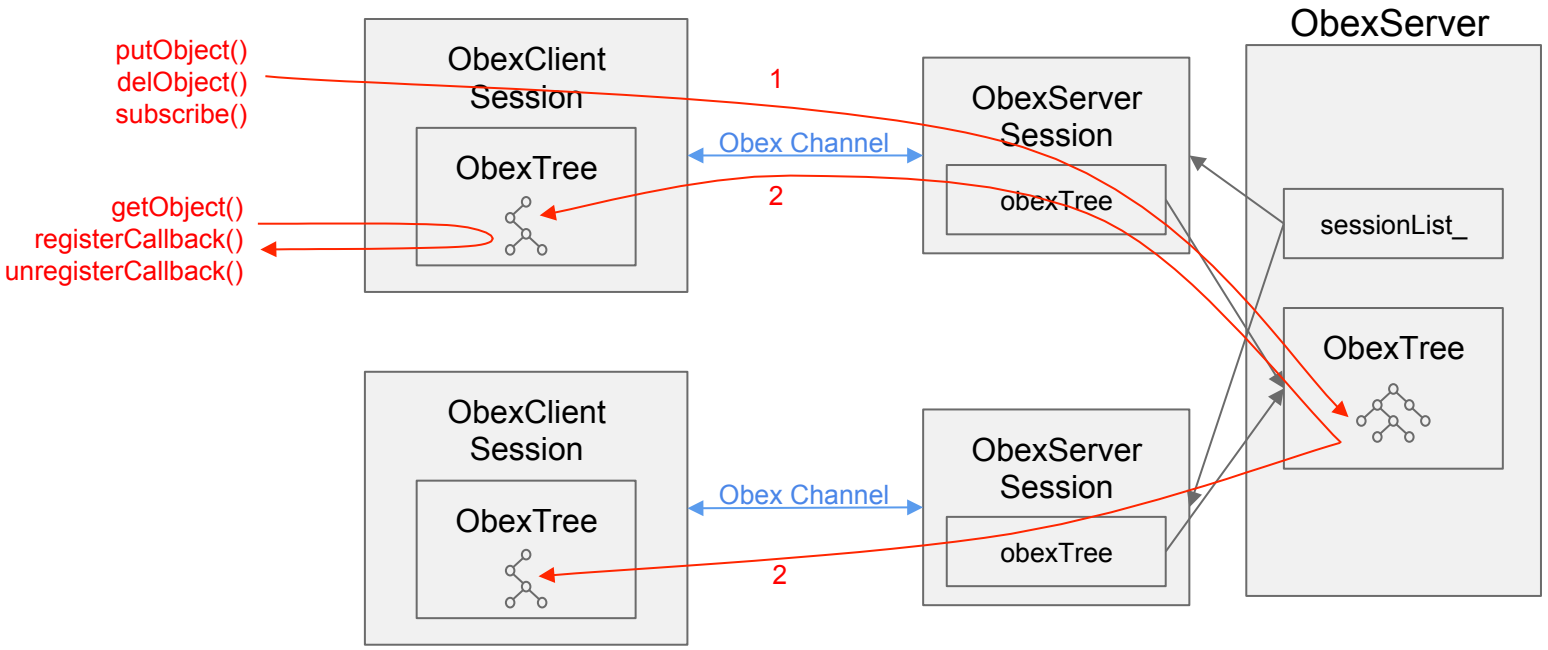
Operation order will be:

put A1 -> del B1 -> put A2 -> put B2

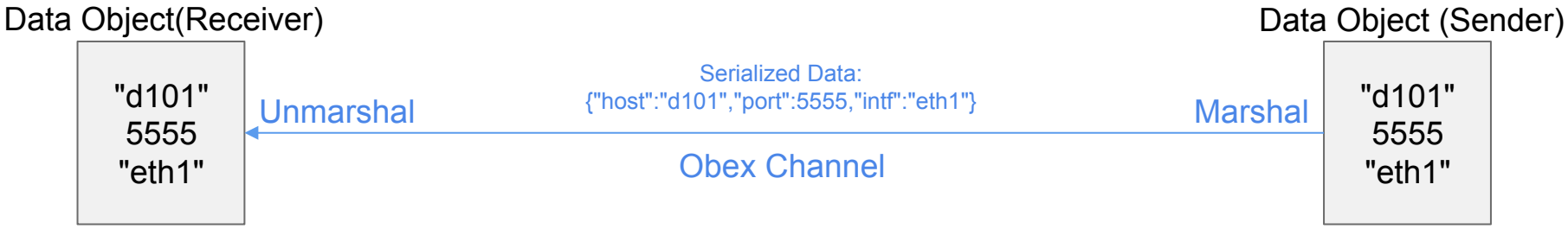
ObexClientSession - Caveats (1)

- Primary copy of a tree node is the one in the ObexServer side.
- In order to avoid complicated sync issues, putObject() updates the server side tree (not the client side tree directly), then the updated object on the server will be propagated to the clients that have subscribed the node. Same rule applies to delObject().
- On the contrary, getObject() looks for the node in client side tree.
- If user calls putObject() and then calls getObject() immediately, the returned object will not be the one that just put in. In this case, the 'sleep()' kind of idling functions between putObject() and getObject() won't help because the propagation will take effect asynchronously when the IoService runs the scheduler.
- Client session can put an object which is not subscribed by that client session. (i.e. blind update is allowed)
- registerCallback() will be bounded within the subscription scope. Setting callbacks out of the scope doesn't do harm but won't trigger the callback event.

ObexClientSession - Caveats (2)



ObexObject Transfer



- Marshal: Object -> Serialized Data
- Unmarshal: Serialized Data -> Object
- Serialized Data format can be of any type such as JSON, XML, even proprietary binary data as long as marshal/unmarshal pair can encode/decode the data.

Ready-made ObexObjects

- ObexStringObject
  - CreateObject<ObexStringObject>("Hello World!");
- ObexIntObject
  - CreateObject<ObexIntObject>(1024);
- ObexFloatObject
  - CreateObject<ObexFloatObject>(1.5f);
- ObexBoolObject
  - CreateObject<ObexBoolObject>(true);
- ObexJsonObject
  - CreateObject<ObexJsonObject>(jsonStr);