Reliable RPC for Java Wrapper

Writing a Client

All methods for reliable RPC are available on the interface object. See `RPCService` for details. The methods are:

- `RPCService.setReliable`
- `RPCService.getReliable`
- `RPCService.reliableCommit`
- `RPCService.reliableRollback`
- `RPCService.getMessageId`
- `RPCService.getStatusOfMessage`

Example (this example is included as source in the *examples/RPC/reliable/JavaClient* folder):

Create Broker object and interface object.

```
Broker broker = new Broker(Mail.DEFAULT_BROKERID, userID);
Mail mail = new Mail(broker);
broker.logon();
```

Enable reliable RPC with `CLIENT_COMMIT`

```
  mail.setReliable(RPCService.RELIABLE_CLIENT_COMMIT);
```

The first RPC message.

```
  mail.sendmail("mail receiver", "Subject 1", "Text 1");
```

Check the status: get the message ID first and use it to retrieve the status.

```
String messageID = mail.getMessageID();
String messageStatus = mail.getStatusOfMessage(messageID);
System.out.println("Status: " + messageStatus + ", id: " + messageID);
```

The second RPC message.

```
mail.sendmail("mail receiver", "Subject 2", "Text 2");
```

Commit the two messages.

```
mail.reliableCommit();
```

Check the status again for the same message ID.

```
messageStatus = mail.getStatusOfMessage(messageID);
System.out.println("Status: " + messageStatus + ", id: " + messageID);
```

The third RPC message.

```
mail.sendmail("mail receiver", "Subject 3", "Text 3");
```

Check the status: get the new message ID and use it to retrieve the status.

```
messageID = mail.getMessageID();
messageStatus = mail.getStatusOfMessage(messageID);
System.out.println("Status: " + messageStatus + ", id: " + messageID);
```

Roll back the third message and check status.

```
mail.reliableRollback();
messageStatus = mail.getStatusOfMessage(messageID);
System.out.println("Status: " + messageStatus + ", id: " + messageID);
broker.logoff();
```

Limitations

1. All program calls that are called in the same transaction (`CLIENT_COMMIT`) must be in the same IDL library.
2. It is not allowed to switch from `CLIENT_COMMIT` to `AUTO_COMMIT` in a transaction.
3. Messages (IDL programs) have `IN` parameters only.

Writing a Server

The server implementation consist of the four classes:

- `Abstract<IDL library name>Server`
- `<IDL library name>`
- `<IDL library name>Server`
- `<IDL library name>Stub`

Add your implementation to the class `<IDL library name>Server`. There are no server-side methods for reliable RPC. The server does not send back a message to the client. The server can run deferred, thus client and server do not necessarily run at the same time. If the server fails, it throws an exception. This causes a cancel of the transaction (unit of work inside the Broker) and the error code is written to the user status field of the unit of work.