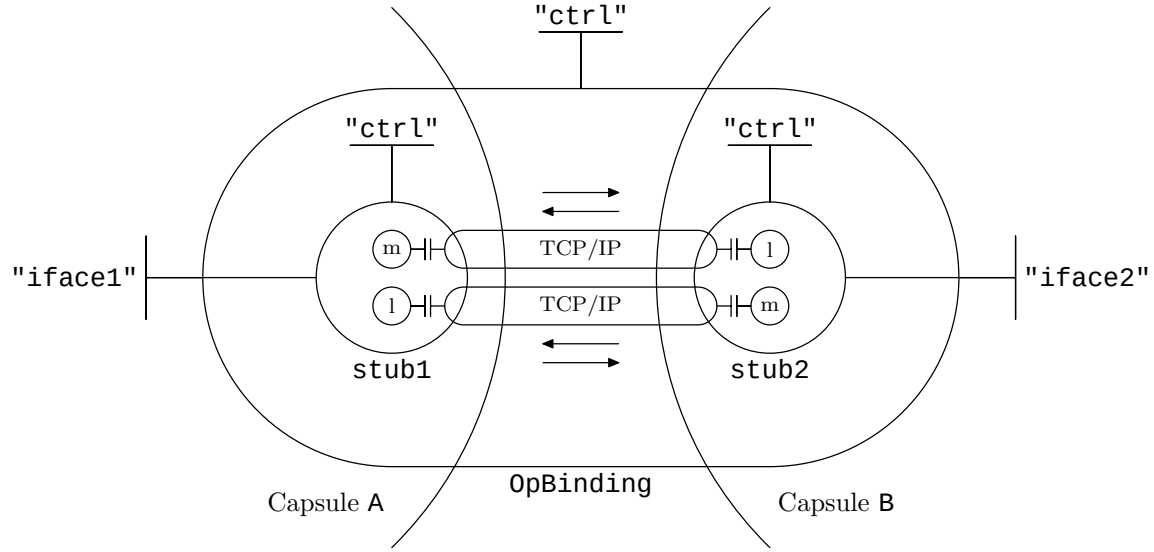
R"""Remote operational bindings

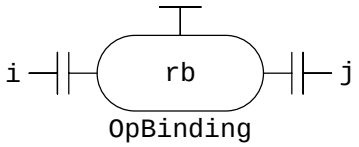
Author : Anders Andersen
Created On : Thu Aug 27 09:25:56 1998
Last Modified By: Anders Andersen
Last Modified On: Fri Sep 10 12:38:01 1999
Status : Unknown, Use with caution!

This module implements the `OpBinding` class and the `remoteBind` function. The `OpBinding` class implements a two way operational binding that tries to emulate the behavior of a local binding that can be created between interfaces in two different capsules.



The example above shows an operational binding with one interface in capsule `A` and one interface in capsule `B`. The `OpBinding` class implements the operational binding with two TCP/IP channels. One for request/reply from left to right and back (from capsule `A` to casule `B` and back in the figure above) and one for request/reply the oposite way (from capsule `B` to casule `A` and back in the figure above). The `remoteBind` function is used to create an operational binding between two interfaces in different capsules. The example below creates an operational binding between interface `i` in capsule `A` and interface `j` in capsule `B`.

```
rb = remoteBind(i,j)
```



Note that `rb` returned from the `remoteBind` function is *not* an instance of the `OpBinding` class but a reference to the registered operational binding component in the local capsule (the `remoteBind` function automatically registers the operational binding in the local capsule).

```
"""

# We need to check the type of some attributes
from types import *

# For low level communication
from socket import *

# Listen on different connections at the same time
```

```
import select

# Need sys to access information about an exception
import sys

# Misc values for the Open-ORB core
from misc import *

# Local bindings (and interface references)
from lbind import *

# Components
from component import *

# Composite components
from composite import *

# Use the message objects
from msg import *

# The capsule
import capsule


class StubMethod:
    R"""A stub method

    Used to call a message type of method (see for example the msg.message method of the Stub class).
    Save the message method and the name of the actual method called when you initialize this object.
    """

    def __init__(self, method=None, op=""):
        R"""Initialize the method

        Save the message method (usually Stub.msg.message) and the name of the actual method called
        (op).
        """
        self.method = method
        self.op = op

    def __call__(self, *args, **kw):
        R"""Call the method

        Call the method we previously saved information about with the given arguments.
        """
        rep = apply(
            self.method, ({"op": self.op, "args": args, "kw": kw},), {})
        debug("StubMethod rep: %s" % ('rep',))
        return rep


class Stub(Component):
    R"""A stub

    A stub in a remote binding. A stub is a component with two main interfaces, one is bound to an interface
    of a component in the local capsule ("iface") and one is bound to an interface in the oposite stub in the
    other capsule ("oface").. This last binding is *not* a local binding. The stub also has a control interface.
    """

    def __init__(self, iref=None, otherNode="", lport=0, oport=0, cport=0):
```

```
    R"""Initialize the stub                                                    127

    Create the different message objects (for out-going, in-coming and control messages) and create
    forwarding methods (using the out-going message object) for the exported methods of the given
    interface reference. Also initialize the component part (interfaces) of the stub.

    """                                                                        136

    # Initialize the message objects                                           137
    self.serving = 0                                                           138
    self.msg = Msg(otherNode, oport)                                           139
    self.listen = Msg(gethostname(), lport, 1)                                 140
    self.ctrl = Msg(gethostname(), cport, 1)                                   141
    self.reply = 1                                                             142
                                                                               143
    # Insert forwarding methods                                                144
    fref = IRef(None, iref.__impID__, iref.__expID__)                          145
    fref.__local__["iobj"] = IObj()                                            146
    for key in fref.__expIDKeys__():                                           147
        fref.__local__["iobj"].__dict__[key] = StubMethod(                     148
            self.msg.message, key)                                             149
                                                                               150
    # Initialize the stub as a component                                       151
    Component.__init__(                                                        152
        self,                                                                  153
        {"iface": fref,                                                        154
         "ctrl" : IRef(self, ["serve", "servethread", "stopserve"], [])},      155
        self)                                                                  156
                                                                               157
def __del__(self):                                                             158
    del self.msg                                                               159
    del self.listen                                                            160
    del self.ctrl                                                              161
                                                                               162
def __serveloop__(self):                                                       163
                                                                               164
    # Start serving                                                            165
    self.serving = 1                                                           166
    debug("Stub serve: ready (%d, %d)" % (self.listen.port,                    167
                                          self.ctrl.port))                     168
    while 1:                                                                   169
        debug("Stub serve: waiting (%d, %d)" % (self.listen.port,              170
                                                self.ctrl.port))               171
                                                                               172
        # Wait for a request (either server or control)                        173
        inreqlist, o, e = select.select(                                       174
            [self.listen.listensocket, self.ctrl.listensocket], [], [])        175
                                                                               176
        debug("Stub serve: something (%d, %d)" % (self.listen.port,            177
                                                  self.ctrl.port))             178
        # Traverse the requests                                                179
        for inreq in inreqlist:                                                180
            if inreq == self.ctrl.listensocket:                                181
                msg = self.ctrl                                                182
                obj = self.interfaces["ctrl"].__local__["iobj"]                183
                isctrl = 1                                                     184
            else:                                                              185
                msg = self.listen                                              186
                obj = self.interfaces["iface"]                                 187
                isctrl = 0                                                     188
```

```
                # Recieve requests
                connection, requests = msg.recvreq()

                for req in requests:

                    debug("Stub request: %s" % (‘req‘,))
                    if isctrl and req["op"] == "stopserve":
                        debug("Stub stopserve: %d (%d)" % (
                            self.listen.port, self.ctrl.port))
                        self.serving = 0
                        return

                    # Perform the request and send a reply (possible an error)
                    try:
                        if not req.has_key("args"): req["args"] = ()
                        if not req.has_key("kw"): req["kw"] = {}
                        rep = apply(getattr(obj, req["op"]),
                                    req["args"], req["kw"])
                    except Exception:
                        (exc, val, tb) = sys.exc_info()
                        debug("Stub: serve error: %s (%s)" % (req["op"], val))
                        rep = ErrorObject(exc, val, tb)
                    if self.reply:
                        msg.sendrep(connection, rep)

    def serve(self):
        self.__serveloop__()

    def servethread(self):
        import thread
        thread.start_new_thread(self.__serveloop__, (), {})

    def stopserve(self):
        if self.serving:
            try:
                self.ctrl.announce({"op": "stopserve"})
            except error, str:
                debug("Couldn’t connect ctrl interface: %s" % (str,))
                raise error, str


class OpBinding(Composite):

    def __init__(self, iref1, iref2, serve="servethread"):

        # Get access to both capsules
        if type(iref1.__local__["object"]) is DictType:
            if iref1.__local__["object"]["capsule"] == capsule.local:
                self.capsule1 = capsule.local
            else:
                self.capsule1 = capsule.CapsuleProxy(
                    iref1.__local__["object"]["capsule"].message.node,
                    iref1.__local__["object"]["capsule"].message.port)
        else:
            self.capsule1 = capsule.local
        if type(iref2.__local__["object"]) is DictType:
            if iref2.__local__["object"]["capsule"] == capsule.local:
```

```
            self.capsule2 = capsule.local
        else:
            self.capsule2 = capsule.CapsuleProxy(
                iref2.__local__["object"]["capsule"].message.node,
                iref2.__local__["object"]["capsule"].message.port)
    else:
        self.capsule2 = capsule.local

    # Fetch communication ports
    lport1 = self.capsule1.newPort("op binding")
    cport1 = self.capsule1.newPort("op binding ctrl")
    lport2 = self.capsule2.newPort("op binding")
    cport2 = self.capsule2.newPort("op binding ctrl")

    # Create stubs
    self.stub1 = self.capsule1.mkComponent(
        Stub,
        (iref1, self.capsule2.message.node, lport1, lport2, cport1),
        {})
    self.stub2 = self.capsule2.mkComponent(
        Stub,
        (iref2, self.capsule1.message.node, lport2, lport1, cport2),
        {})

    # Initialize the componet
    Component.__init__(
        self,
        {"iface1": self.capsule1.getIRef(self.stub1, "iface"),
         "iface2": self.capsule2.getIRef(self.stub2, "iface"),
         "ctrl": IRef(self, ["servethread", "serve", "stopserve"], [])},
        {"comps": [self.stub1, self.stub2], "ifaces": {}, "edges": []})

    # Start server threads
    if serve in ["servethread", "serve"]:
        self.__serve__(serve)

def __serve__(self, serve):
    if self.interfaces["iface1"].__impID__:
        self.capsule1.announceMethod(
            self.stub1, "ctrl", serve, (), {})
    if self.interfaces["iface2"].__impID__:
        self.capsule2.announceMethod(
            self.stub2, "ctrl", serve, (), {})

def serve(self):
    self.__serve__("serve")

def servethread(self):
    self.__serve__("servethread")

def stopserve(self):
    debug("OpBinding stopserve")
    self.capsule1.announceMethod(
        self.stub1, "ctrl", "stopserve", (), {})
    self.capsule2.announceMethod(
        self.stub2, "ctrl", "stopserve", (), {})
```

```
class OpBindCtrl(IRef):                                                    305
    def __init__(self, obj=None):                                         306
        IRef.__init__(self, obj, [], ["servethread", "serve", "stopserve"]) 307
                                                                           308
                                                                           309
def remoteBind(iref1, iref2, serve="servethread"):                         310
    rob = capsule.local.registerComponent(OpBinding(iref1, iref2, serve)) 311
    localBind(iref1, capsule.local.getIRef(rob, "iface1"))                 312
    localBind(capsule.local.getIRef(rob, "iface2"), iref2)                 313
    return rob                                                             314
                                                                           315
```