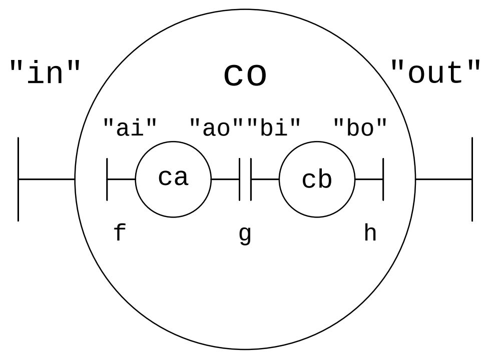
R"""Composite objects

Author : Anders Andersen
Created On : Tue Apr 28 09:34:23 1998
Last Modified By:
Last Modified On: Wed Jul 07 21:14:27 1999
Status : Unknown, Use with caution!

Copyright © 1998, 1999 Lancaster University, UK and NORUT Information Technology Ltd., Norway. See COPYING for details.

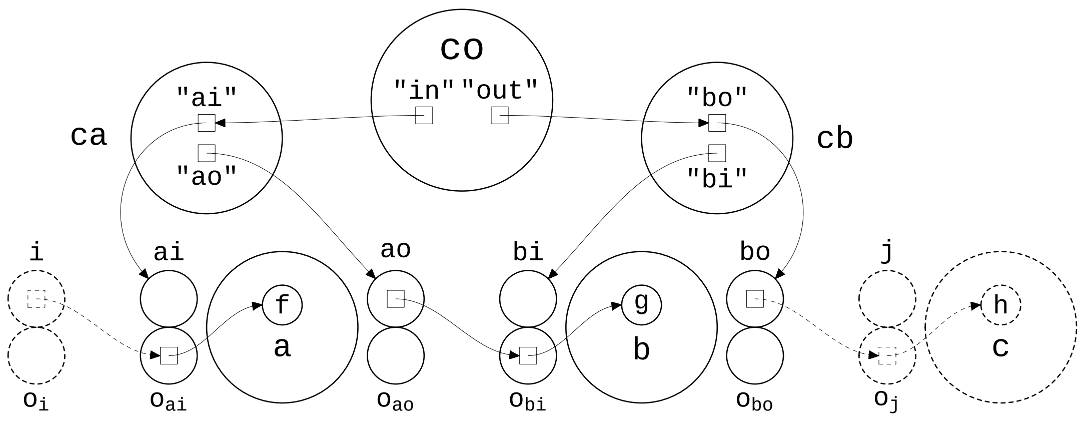
This module implements a class (and a factory) for composite objects based on the component class in `component.py`. The composite object is represented as an object graph with local bindings between the objects in the graph. Objects can be located in different capsules on different hosts (see `capsule.py` and `nodemngr.py`).



```
co = Composite(
    {"in":(ca,"ai"),"out":(cb,"bo")},
    {"comps":[ca,cb,cc],
     "ifaces":{"ao":(ca,"ao"),
               "bi":(cb,"bi")},
     "edges":[("ao","bi")]})
i = IRef(None, [], ["f"])
lbi = localBind(i, co.interfaces["in"])
j = IRef(c, ["h"], [])
lbj = localBind(co.interfaces["out"], j)
```

The figure above shows a simple example of a composite object `co` created from the class `Composite` with two components `ca` and `cb`, and two external interfaces, `"in"` and `"out"`. The `"in"` interface of `co` is defined to be the `"ai"` interface of `ca` (which is the `ai` interface of object `a`), and the `"out"` interface of `co` is defined to be the `"bo"` interface of `cb` (which is the `bo` interface of object `b`). `ca` and `cb` in the example above is either actual (local) object references or a dictionary containing `"capsule"` (a capsule or a capsule proxy) and `"comp"` (the name of the registered component).

The code above also binds the `"in"` interface of `co` to the (empty) interface `i` and the `"out"` interface of `co` to the interface j of an object `c`. The interface `i` can now be used to access the exported method `f` in `co` (which is method `f` in `ca` which is method `f` in `a`). When `co` access the imported `h` method of its interface `"out"`, it will access the `h` method in `c`. The control objects of the local bindings are not shown in the figure below.



"""

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

```
# Use local bindings to connect interfaces
from lbind import *

# Composite objects are an extension of components
from component import *


class Composite(Component):
    R"""Composite component

    This class is used for composite objects. A composite object is represented by a component graph representing its constituent components. Since a composite object also is a component, it also has public named interfaces.

    """

    def __init__(self, interfaces={}, componentGraphSpec={}, dir=0):
        R"""Create a composite component

        A composite component is created with the external interfaces and a component graph specification. The component graph specification is a directory containing a list of the components, a mapping from interface names to actual interface references and a list of edges in the graph. If the optional argument dir is set to 1 (default is 0) the edges in the object graph are directional and represents one-way local bindings. A composite object can also contain stand-alone components (components without edges). The introduction to this module (see above) includes an example of the usage of this class.

        """
        componentGraph = self.bindComponents(componentGraphSpec, dir)
        ifaces = {}
        for (name, (comp, iname)) in interfaces.items():
            if type(comp) is DictType:
                ifaces[name] = comp["capsule"].getIRef(comp["comp"], iname)
            else:
                ifaces[name] = comp.interfaces[iname]
        Component.__init__(self, ifaces, componentGraph)

    def bindComponents(self, componentGraphSpec, dir):
        R"""Binds the components in the components graph

        Parse the component graph and make local bindings between the components.

        """

        # Initial component graph
        componentGraph = {
            "components": componentGraphSpec["comps"], "edges": {}}

        # Loop through all edges
        for (iname1, iname2) in componentGraphSpec["edges"]:

            # Fetch the component of these interfaces
            comp1 = componentGraphSpec["ifaces"][iname1][0]
            name1 = componentGraphSpec["ifaces"][iname1][1]
            comp2 = componentGraphSpec["ifaces"][iname2][0]
            name2 = componentGraphSpec["ifaces"][iname2][1]

            # Fetch interface references
            if type(comp1) is DictType:
                iref1 = comp1["capsule"].getIRef(comp1["comp"], name1)
            else:
                iref1 = comp1.interfaces[name1]
            if type(comp2) is DictType:
                iref2 = comp2["capsule"].getIRef(comp2["comp"], name2)
            else:
                iref2 = comp2.interfaces[name2]

            # Create bindings (directional or not)
            if dir:
                componentGraph["edges"][(iname1, iname2)] = \
                                                  localBindOneWay(iref1, iref2)
            else:
                componentGraph["edges"][(iname1, iname2)] = \
                                                  localBind(iref1, iref2)

        # Return the component graph
        return componentGraph


def compositeFactory(interfaceNames={}, componentGraphDesc={}, dir=0):
    R"""A factory for composite components
```

Creates a composite component including all its constituent components. The component graph is build using local bindings (but components in the graph can be binding objects).

The first argument is a list of the interfaces of the new composite component. Each interface is represented as a mapping from a name to a component/interface name pair. In the mapping `key:(comp,ifname)` is `key` the name of the interface, `comp` the name of the component, and `ifname` the name of the interface in the component `comp` implementing the interface `key`.

The second argument is the component graph specification. The component graph specification is a directory containing a list of the components, a mapping from interface names to actual interface references and a list of edges in the graph. Since the components are not yet created the components are specified by a name and how to create them (class or factory and its arguments). Each component is represented by at tuple, where the first element is the name (key) of the component and the second element is a dictionary with information about how to create this component. `"factory"` is the function (or class) used to create the component, `"args"` and `"kw"` are the arguments passed to the factory (these are optional) and `"capsule"` can be used to specify which capsule this component should be created in (the local capsule is default). The `"ifaces"` and `"edges"` part of the specification are similar to their counter part in the component graph specification of the `Composite` class (the only difference is that `"ifaces"` uses names for components and not their actual references).

The last argument is equal to the last argument of the constructor of the `Composite` class. This is an example of the creation of a composite component with this factory:

```
co = compositeFactory(
    {"in":("ca","ai"),"out":("cb","bo")}
    {"comps":{"ca":{"factory":componentFactory,
                    "args":(["ai", "ao"], A)},
              "cb":{"factory":componentFactory,
                    "args":(["bi", "bo"], B)}},
     "ifaces":"ao":{("ca","ao"),"bi":("cb", "bi")},
     "edges":[("ao","bi")]})
```

```
    """

    # Initialize structures
    components = {}; interfaces = {}
    componentGraphSpec = {"comps": [], "ifaces": {},
                          "edges": componentGraphDesc["edges"]}

    # Create components
    for (key, cinfo) in componentGraphDesc["comps"].items():
        if not cinfo.has_key("args"):
            cinfo["args"] = ()
        if not cinfo.has_key("kw"):
            cinfo["kw"] = {}
        if cinfo.has_key("capsule"):
            components[key] = {
                "capsule": cinfo["capsule"],
                "comp": cinfo["capsule"].mkComponent(
                    cinfo["factory"], cinfo["args"], cinfo["kw"])}
        else:
            components[key] = (
                apply(cinfo["factory"], cinfo["args"], cinfo["kw"]))
        componentGraphSpec["comps"].append(components[key])

    # Generate external interface listing
    for (key, (comp, ifname)) in interfaceNames.items():
        interfaces[key] = (components[comp], ifname)

    # Generate internal interface listing
    for (key, (comp, ifname)) in componentGraphDesc["ifaces"].items():
        componentGraphSpec["ifaces"][key] = (components[comp], ifname)

    # Create composite object
    return Composite(interfaces, componentGraphSpec)


# LocalWords:  Apr Oct UK NORUT aacodefont py nodemngr parbox linewidth pt co
# LocalWords:  includegraphics hfil aaws cb ao bi IRef lbi localBind lbj comp
# LocalWords:  lbind def init componentGraph dir ldots bindComponents ifaces
# LocalWords:  iname DictType getIRef ci TupleType iref localBindOneWay ifname
# LocalWords:  compositeFactory interfaceNames componentList nameGraph args kw
# LocalWords:  cinfo mkComponent
```