R"""An automata component class

Author : Anders Andersen
Created On : Mon Mar 8 11:02:17 1999
Last Modified By:
Last Modified On: Mon Jul 05 23:31:58 1999
Status : Unknown, Use with caution!

This module implements a class that can be used to create automata components. The `AmComp` class is implemented with the `Automata` and the `Component` class (and it inherits the API of both these classes). An object of the `AmComp` class provides three sets of interfaces: (i) control interfacess contains only one interface `a_ctrl` that exports the methods `print_state`, `run` and `stop`, (ii) input interfacess that contain all the input (signal) interfaces of the automaton, and (iii) output interfacess that contain all the output (signal) interfaces of the automaton. The prefix `a_in_` is added to the name of all the input interfaces and the prefix `a_out_` is added to the name of all the output interfaces.

An automata component can be created like this (`ex.fc2` is an FC2 description of the automaton):

```
from fc2 import FC2
from amcomp import AmComp
a = AmComp(FC2(open("ex.fc2")).fc2py)
```

If the automaton described in `ex.fc2` accepts the input signals `in` and `out` and can produce the output signal `overflow`, the following interfaces are available (in the `a.interfaces` dictionary):

`"a_ctrl"`: The control interface
`"a_in_on"`: The `on` input signal
`"a_in_off"`: The `off` input signal
`"a_out_overflow"`: The `overflow` output signal

The control interface exports the `print_state`, `run` and `stop` methods, and the input and output signal interfaces respectively export and import the `event` method.

"""

We need to create interfaces for the automata components
from *lbind* import *
from *sigbind* import *

The automata component is a combination of an automata and a component
from *component* import *Component*
from *automata* import *Automata*

class *EventObj*:
    R"""Forwarding input events

    Each input interface use an instance of this class to forward input events with the right argument to the automaton.

    """

    def *__init__*(self, amc, msg):
        R"""Save automaton and event

        Save a reference to the automaton and the name of the event of this interface.

        """
        self.amc = amc

```
        self.msg = msg

    def event(self):
        R"""Forward input event

        Forwards the input event to the automaton.

        """
        self.amc.new_event(self.msg)

class AmComp(Automata, Component):
    R"""Automata component

    A class for automata components implemented with the Automata and the Component class.

    """

    def __init__(self, fc2py):
        R"""Initialise the automata component.

        Install the automaton description and create all interfaces (including the control interface and one interface for each input and output event).

        """
        Automata.__init__(self, self.__event__, fc2py)
        interfaces = self._make_ifaces()
        interfaces["a_ctrl"] = IRef(self, ["print_state", "run", "stop"], [])
        Component.__init__(self, interfaces, self)

    def _make_ifaces(self):
        R"""Create input and output interfaces

        Use the information about the edges from all vertice to create the set of input and output interfaces. Each input interface is mapped to a specific input event, and each output event is mapped to a specific output interface. The name of an input interface is the name of its event with the a_in_ prefix, and the name of an output interface is the name of its event with the a_out_ prefix.

        """
        interfaces = {}
        for (name, vertex) in self.vertice.items():
            if vertex.has_key("edges"):
                for (label, events) in vertex["edges"].items():
                    if label:
                        iname = "a_in_" + label
                        eobj = EventObj(self, label)
                        interfaces[iname] = SigSinkIRef(eobj)
                    for event in events:
                        if event.has_key("mesg"):
                            for msg in event["mesg"]:
                                iname= "a_out_" + msg
                                interfaces[iname] = SigSrcIRef()
        return interfaces

    def __event__(self, msg):
        R"""Forward output events

        Forward output events to the appropriate output (signal) interface.

        """
        self.interfaces["a_out_" + msg].event()
```