

Piccolo: A Simple Python Framework for Introducing Components Principles

`Raphael.Marvie@lifl.fr`

LIFL – University of Lille 1 (France)

Göteborg – June 28, 2005

EuroPython 2005

Few Words About the Speaker

- ✓ Assistant Professor, software engineering
 - ▶ Teaching : objects, components, and distributed computing
 - ▶ Research : metamodeling and model driven engineering
 - ▶ Standardization : Object Management Group (CCM, MOF QVT)
 - ▶ Consulting : from small to big companies

- ✓ What about Python ?
 - ▶ Fond of scripting languages since 1998 (Tcl, IDLscript)
 - ▶ As much as possible since 2003
 - ▶ Research, teaching, and more

Menu

✓ Motivations

- ▶ Why components ?
- ▶ Introduction material (very light)

✓ Picolo : a component activity toolkit

- ▶ Picolo component model (a quick overview)
- ▶ Building a component-based application (incrementally)

✓ Concluding remarks

Motivations

Why components ?

- ✓ Assembling software more than programming
 - ▶ Objects : programming in the small
 - ▶ Components : programming in the large

- ✓ A software component is a **unit of composition** with **contractually specified interfaces** and **explicit context dependencies** only. A software component can be **deployed independently** and is subject to composition by third parties.

- ✓ However, components do not replace objects !

Introducing Component is Difficult

- ✓ Existing platforms are not for introduction
 - ▶ Industrial : too complex
 - ▶ Research : too specific

- ✓ There is a need for a *Component Activity Kit*
 - ▶ Learning the basic concepts
 - ▶ Being able to understand all the pieces

- ✓ Explaining the inner mechanisms

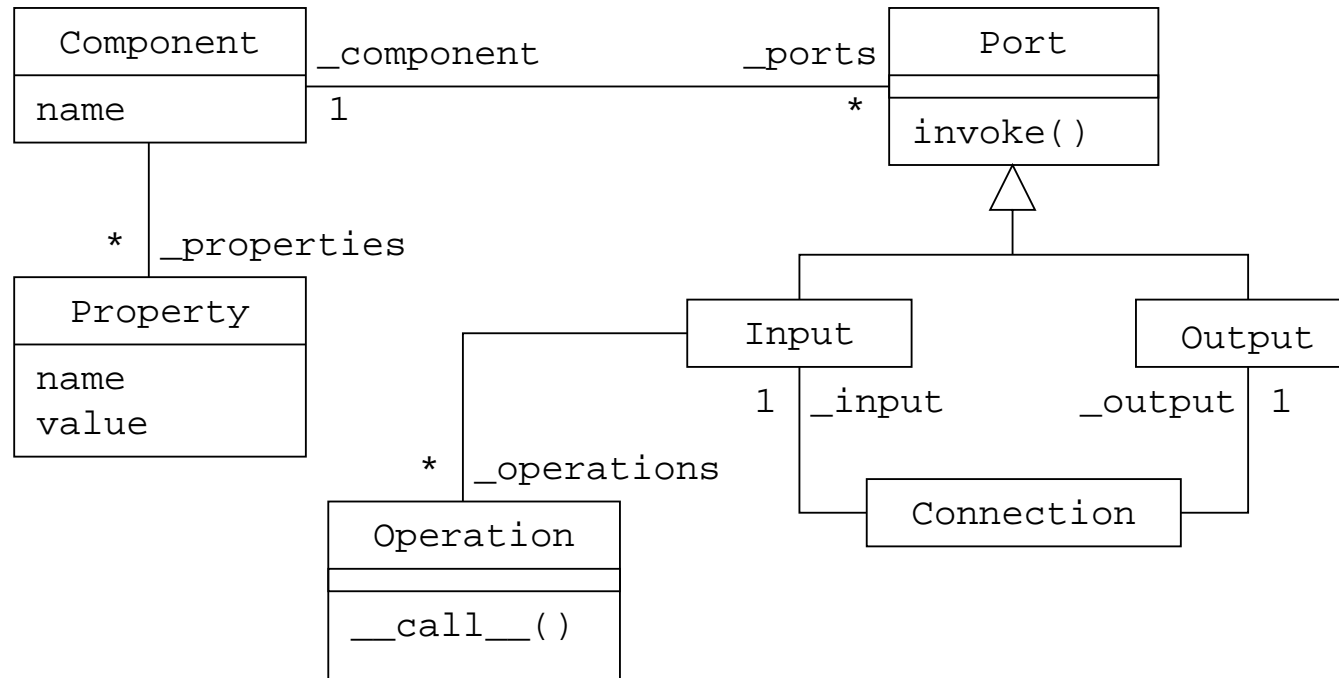
Piccolo : A Component Activity Kit

A Pico Component

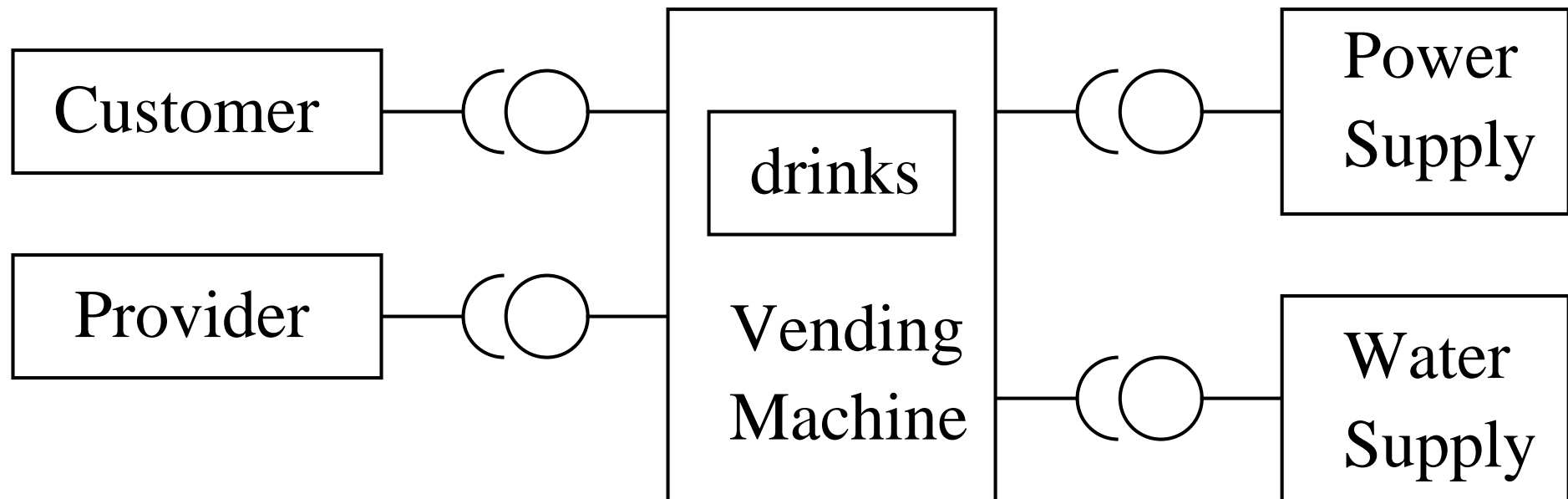
- ✓ Three-level structure (different concerns)
 - ▶ A component is intended to define a well identified **software unit**.
 - ▶ A port is intended to establish **connections** between components.
 - ▶ Operations and properties reflect **the behavior and the state** of a component.

- ✓ Ports are connection points
 - ▶ Input ports provide operations to other components (clients)
 - ▶ Output ports provide access to input ports of other components

Piccolo Component Model



A Component “hello world”

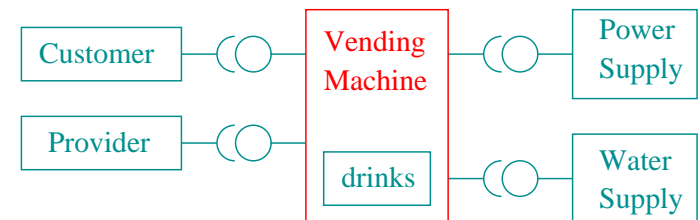


Defining Components

```
>>> import piccolo.core as piccolo
>>> vm = piccolo.Component ('vending_machine')
>>> vm.get_ports ()
{}
>>> vm.get_properties ()
{}
```

✓ Components are empty shells

- ▶ Managing the structure : ports and properties
- ▶ Providing introspection and dynamic definition

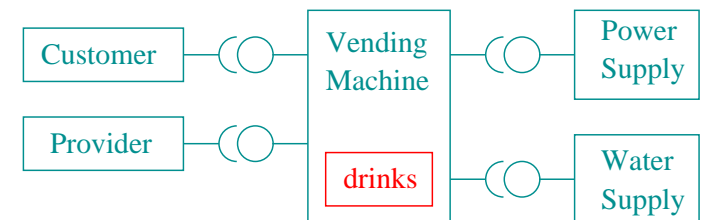


Defining Properties

```
>>> from vm_impl import *
>>> vm.set_property ('drinks', Stock ())
>>> vm.get_properties ()
{'drinks': <vm_impl.Stock instance at 0x3767d8>}
```

✓ Properties are Python objects

- ▶ Mutable objects for read-write properties
- ▶ Non mutable objects for read-only properties

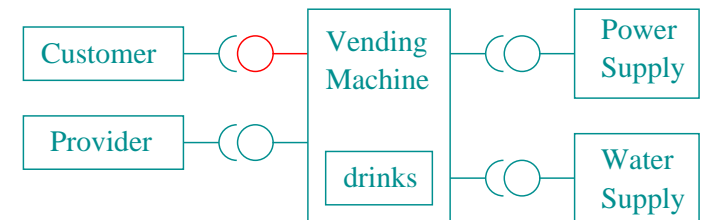


Defining Input Ports

```
>>> cport = piccolo.Input ()
>>> cport.get_operations ()
{}
>>> vm.bind_port ('customer', cport)
>>> vm.get_ports ()
{'customer': <piccolo.core.Input instance at 0x3768c8>}
```

✓ Input ports are empty shells

- ▶ Managing list of operations
- ▶ Providing a lookup / invocation mechanism

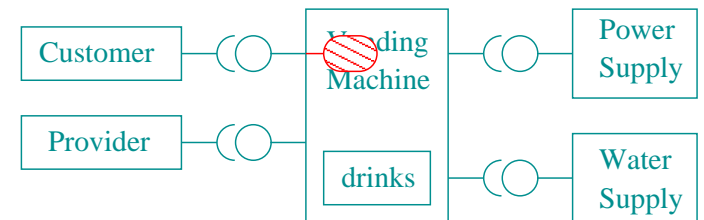


Defining Input Port Operations

```

class ListOfDrinks (picolo.Operation):
    def __call__ (self, cmp, args):
        stock = cmp.get_property ('stock')
        return stock.get_drink_names ()
>>> cport.set_operation ('list_of_drinks', ListOfDrinks ())
>>> cport.get_operations ()
{'list_of_drinks': <vm_impl.ListOfDrinks instance at 0x3767b0>}

```

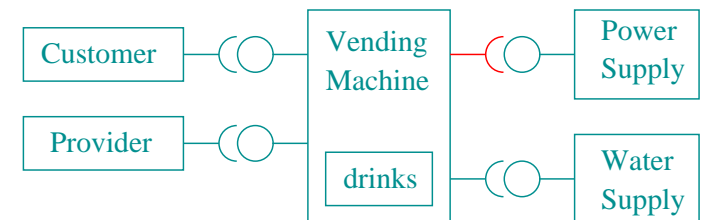


✓ Operations are callable objects

- ▶ First parameter is the component reference
- ▶ Second parameter is the sequence of arguments

Defining Output Ports

```
>>> pwater = piccolo.Output ()
>>> vm.bind_port ('water', pwater)
>>> vm.get_ports ()
{'customer': <piccolo.core.Input instance at 0x3768c8>,
 'water': <piccolo.core.Output instance at 0x376af8>}
```



✓ Output ports

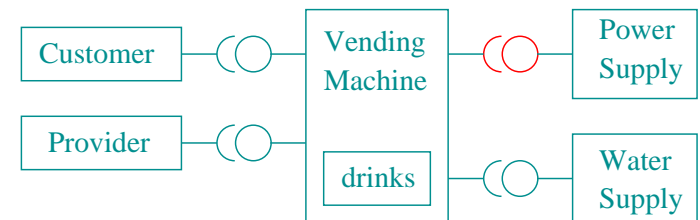
- ▶ Provide access to other component operations
- ▶ Used from inside the implementation of the component (operations)

Connecting Component Ports

```
>>> water.get_ports ()
{'tape': <picolo.core.Input instance at 0x376c60>}
>>> ptape = water.get_port ('tape')
>>> cnx = picolo.Connection (vm.get_port ('water'), ptape)
>>> cnx.connect ()
```

✓ Connection objects

- ▶ Defined between an output and an input port
- ▶ Used for performing (de)connection



Using Components

```
>>> cust = vm.get_port ('customer')
>>> cust.invoke ('pay_drink', None)
>>> cust.invoke ('select_drink', ('tea',))
>>> cust.invoke ('take_drink', None)
tea
```

- ✓ Two levels of interactions
 - ▶ Select the required input port
 - ▶ Invoke operations on the port

Concluding Remarks

Piccolo Framework

✓ Introduction material

- ▶ Intended to understand how components work
- ▶ Appropriate for experimenting (like research)
- ▶ Not suited for building *real* applications

✓ Make it your own

- ▶ Activity kits are quickly boring or limited
- ▶ Second level exercises are related to code generation

Picolo Implementation

✓ Core module

- ▶ Main mechanisms of components
- ▶ About 350 lines of code

✓ Extension modules

- ▶ Description, life cycle, trace
- ▶ Extension framework : write your own ones

That's all folks !

Component Assembly Description

```
<picolo name="vending_machine">

  <component name="VendingMachine">
    <input name="customer">
      <operation name="list_of_drinks" type="vm_impl.ListOfDrinks" />
      <operation name="pay_drink" type="vm_impl.PayDrink" />
      <operation name="select_drink" type="vm_impl.SelectDrink" />
      <operation name="take_drink" type="vm_impl.TakeDrink" />
    </input>

    <input name="provider">
      <operation name="init_drinks" type="vm_impl.InitDrinks" />
      <operation name="refill_drinks" type="vm_impl.RefillDrinks" />
      <operation name="empty_cashier" type="vm_impl.EmptyCashier" />
    </input>

    <output name="water" />
    <output name="power" />

    <property name="cashier" type="vm_impl.Cashier" />
    <property name="stock" type="vm_impl.Stock" />
    <property name="selection" type="vm_impl.Selection" />
  </component>
</picolo>
```

```
<component name="WaterSupply">
  <input name="tape">
    <operation name="get_water" type="vm_impl.GetWater" />
  </input>
</component>

<component name="PowerSupply">
  <input name="plug">
    <operation name="get_power" type="vm_impl.GetPower" />
  </input>
</component>

<component name="Provider">
  <input name="main">
    <operation name="init" type="vm_impl.Initialize" />
  </input>
  <output name="machine" />
</component>

<component name="Customer">
  <input name="main">
    <operation name="run" type="vm_impl.Run" />
  </input>
  <output name="machine" />
</component>
```

```
<assembly>
  <instance name="machine" type="VendingMachine" />
  <instance name="water" type="WaterSupply" />
  <instance name="power" type="PowerSupply" />
  <instance name="provider" type="Provider" />
  <instance name="customer" type="Customer" />

  <connection>
    <source component="machine" port="water" />
    <target component="water" port="tape" />
  </connection>

  <connection>
    <source component="machine" port="power" />
    <target component="power" port="plug" />
  </connection>

  <connection>
    <source component="provider" port="machine" />
    <target component="machine" port="provider" />
  </connection>

  <connection>
    <source component="customer" port="machine" />
    <target component="machine" port="customer" />
  </connection>
</assembly>
</picolo>
```

Example of Operation Implementation

```
class Run (picolo.Operation):  
  
    def __call__ (self, cmp, args):  
        machine = cmp.get_port ('machine')  
        machine.invoke ('pay_drink', None)  
        machine.invoke ('select_drink', args)  
        drink = machine.invoke ('take_drink', None)  
        print 'Here is your', drink
```

Example Deployment

```
import picorine.core as picorine
import piccolo.lifecycle as lifecycle
import piccolo.description as description

desc = picorine.factory(description).create('vm_desc.xml')
factory = lifecycle.ExtensionFactory(locals())
factory.create_assembly(desc)

customer = factory.find_component('customer')
main = customer.get_port('main')
main.invoke('run', ('tea',))
```

Non Functional Property : Tracing (i)

```
class TracedInput (picolo.Input):  
  
    def __init__ (self, ops=None):  
        piccolo.Input.__init__ (self, ops)  
  
    def invoke (self, opname, args):  
        print '[trace] ', opname,  
        print 'invoked on', self._component.get_name (),  
        print 'with args', str (args)  
        return piccolo.Input.invoke (self, opname, args)
```

Non Functional Property : Tracing (ii)

```
>>> ops = vm.get_port('customer').get_operations()
>>> vm.unbind_port('customer')
>>> p1 = trace.TracedInput (ops=ops)
>>> vm.bind_port('customer', p1)

>>> cust = vm.get_port ('customer')
>>> cust.invoke ('pay_drink', None)
[trace] pay_drink invoked on vending_machine with args None
>>> cust.invoke ('select_drink', ('tea',))
[trace] select_drink invoked on vending_machine with args ('tea'
>>> cust.invoke ('take_drink', None)
[trace] take_drink invoked on vending_machine with args None
tea
```