2.2 P2 Data

A tuple (tuple.h) represents the data type that is passed between Elements in a Dataflow. A tuple contains a list of value (value.h) types, which are defined in the 'p2core' system directory and have a file name prefix='Val_'. The types defined include all the base C++ types (e.g., int, float, strings) as well as some non traditional types (e.g., identity, lists, ip address, etc.). A tuple is an imd utable object in the sense that once an Element creates, it cannot be changed by some other Element in the data flow [1].

2.3 Element

Each element defines a set of input ports and output ports. Elements process the tuples that arrive on its input ports and send, possibly new, tuples on its output ports. The kind of processing that is performed on a tuple is specific to the Element type and possibly the port on which a tuple is received. There are three types of ports that an Element can define on its interface, and dependingo

```
class Element {
    . . .
  virtual int push(int port, TuplePtr, b_cbv cb);

  virtual TuplePtr pull(int port, b_cbv cb);

  virtual TuplePtr simple_action(TuplePtr p);
    . . .
}
```

Every Element defines three methods for receiving tuples, as shown above. The actual methods that are called on receipt of a tuple depend on the port type. For all port types, the simple_action method will be called. An Element that defines

2.4 Dataflow

A Dataflow

2.5 Plumber

```
class Plumber {
        . . .
        int install(DataflowPtr d);
        . . .
 }
```

3.2 Importing P2 Data Structures

ited by this example. The first comes from the fact that a P2 value does not define a constructor, which is indicated by the no_

Python

dynamic dispatch to the respective overridden methods in the Python class instance. That is, a Python class that inherits from the Element class will actually inherit from the ElementWrap class, which automatically invokes any methods that the Python class overrides. Since the C++ compiler is unaware of the Python class definition, it will call the overridden methods of the ElementWrap class, which will in turn invoke the Python interpreter to call the respective overridden Python method. If the Python class doesn't override a particular method then the ElementWrap class will invoke the method defined in the parent Element class.

The only known limitation of Boost.Python is the ability to pass a function from Python to C++, and have that function be called from within C++. Given that

If your callback requires the use of arguments then you should use wrap it inside of a Python lambda function that will pass it the required arguments at call time.

3.3.2 Example P2 Element class in Python

```
class Terminal(Element):
  def __init__(self, name):
      Element.__init__(self, name, 1, 1)
      self.self(self) # Pass the ElementWrap class a reference to the self object.
  def class_name(self): return "Terminal"
  def processing(self): return "h/h"
  def flow_code(self):  return "-/-"
  def initialize(self):
      self.timer = self.set_delay(0, self.delay_callback)
      return 0
  def callback(self, port):
      self.timer = self.set_delay(0, self.delay_callback)
  def delay_callback(self):
      # Read string from terminal and send
      # it in a tuple to push output port 0.
      line = raw_input("P2 Terminal >> ")
      t = Tuple.mk()
      t.append(Val_Str.mk("terminal"))
      t.append(Val_Str.mk(text))
      t.freeze()
      if self.py_push(0, t, self.callback) > 0:
        self.timer = self.set_delay(1, self.delay_callback)
  def push(self, port, tp, cb):
      # Received some tuple on input port 0
      return 0
```

The above Terminal class is written entirely in Python 28 0 Td 2.2462 0 Td (can)Tj 20.9341 0

delay_callback method defined by the Python Terminal instance. The delay

kmethod

Table 1: P2DL Terms

comment	All characters on a single line following #
numeric	Both integer and float syntax is supported.
string	

```
'dataflow' <DataflowType> {
     ( assignment; )*
     ( strand; )+
}
. # END OF PROGRAM
```

The declaration of a data o

the array syntax for specifying the port of an element. For instance, in the 'Foobar' dataflow example, '[0]' indicates port 0 on the output of 'TimedPushSource' and the input of 'Discard'. The array syntax is port

elements but until then see ifelements suchas a Demux or RoundRobin for input and a Mux for the outpu[illegible]

4.4 Edit [illegible] Td (application)Tj /R16 11.9551 Tf -90.7588 -22.167

5 Incremental P2 Rule Installer

The primary contribution of this work is to provide an interface that allows for OverLog rules to be installed at runtime. The incremental planner relies on the P2 Python Library module and the P2 Dataflow Language modules that were built in the course of this semester. Some modifications were made to the native P2 runtime environment, most notably the support for dataflow edits (as given by the Plumber::DataflowEdit class) and a few new elements for transferring large files (Frag/Defrag), compiling OverLog (OverlogCompiler), compiling and installing a P2DL description (DataflowInstaller), and establishing a dissemination tree over a set of P2

!"55".008 Tm (!)Tj 4.65059 0 Td (")Tj

to the DataflowInstaller. The DataflowInstaller accepts tuples containing P2DL edits, provided by the OverlogCompiler or from some outside source [5].

P2 stub process and the new P2 stub will exit.

References

[1] E.