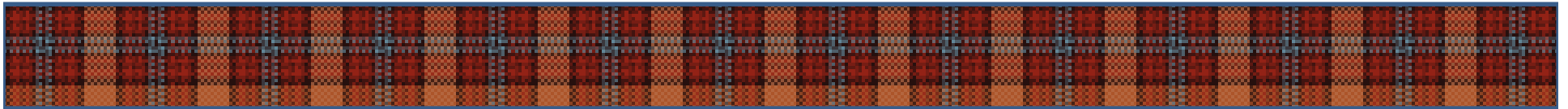


Resource-Based Programming in

PLAID



Jonathan Aldrich

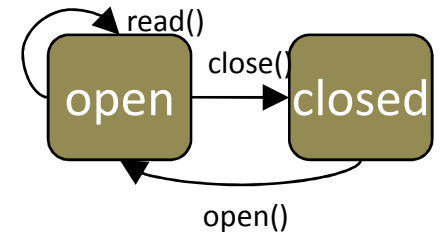
Carnegie Mellon University

Fun and Interesting Thoughts Session – PLDI

June 8, 2010

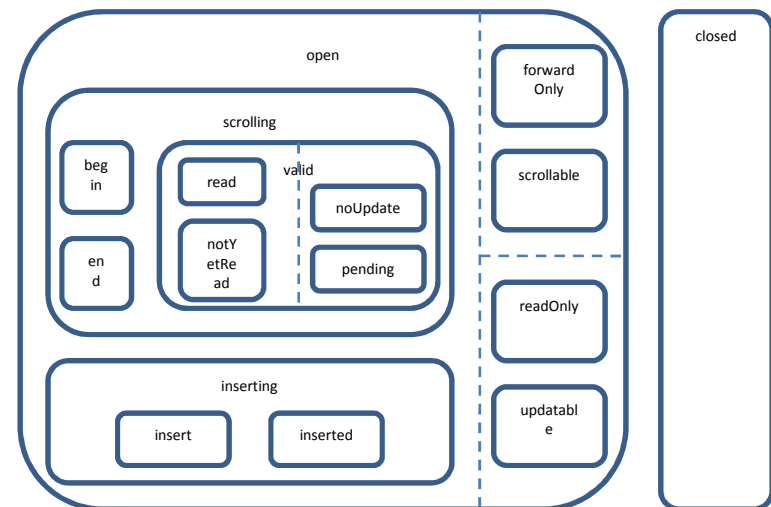
Resources and Composition

- Resource: stateful object whose use is constrained
 - I/O: files, database connections, device access
 - Software abstractions: initialization
 - Web 2.0 pages – AJAX state changes



- Complex
 - JDBC ResultSet: 33 states
- Easy to misuse
 - Often poor documentation
 - Inadequate checking

Java Database Connectivity (JDBC) Library State Space



Resources and Composition

- Resource: stateful object whose use is constrained

- I/O: file handles, sockets, pipes, etc.
- Scoped objects
- Windows

The Wild Idea:

What if we organized a *programming language* around the concept of constrained resources?

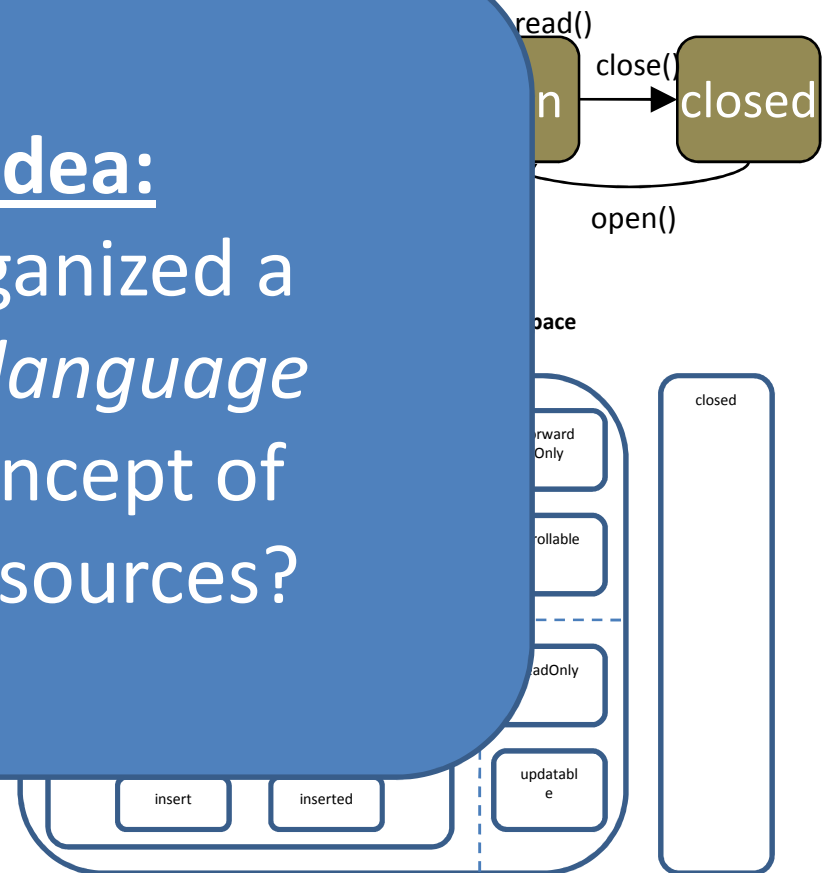
- Com

- JD

- Easy

- OI

- Inadequate checking



Resources are Modeled with Typestates

Typestate-Oriented Programming

a **programming paradigm** in which:

programs are made up of dynamically created **resource objects**,
each object has a **typestate** that is **changeable**
and each state has an **interface**, **representation**, and **behavior**.

- Like Javascript, can add and remove fields and methods at run time
- Use **typestate** and **permissions** to statically typecheck these changes

Typestate-Oriented Programming is embodied in the language

PLAiD

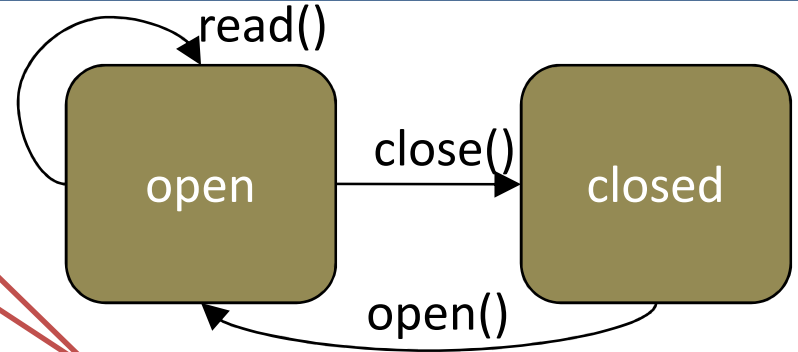
PLAiD

Resource-Based Programming in
Plaid

Typestate-Oriented Programming

```
state File {  
  val String filename;  
}  
state ClosedFile = File with {  
  method void open() [unique ClosedFile>>OpenFile];  
}  
state OpenFile = File with {  
  private val CFile fileResource;  
  
  method int read();  
  method void close() [OpenFile>>ClosedFile];  
}
```

State transition



Permission / aliasing info.

Different representation

New methods

Implementing State Changes

```
method void open() [unique ClosedFile>>OpenFile] {  
  this <- OpenFile {  
    fileResource = fopen(filename);  
  }  
}
```

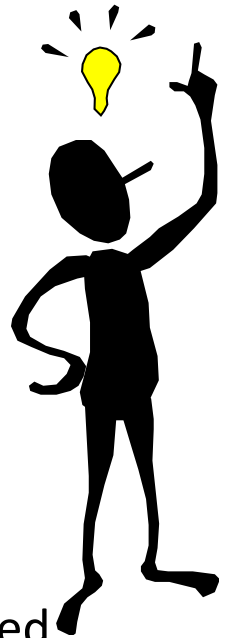
State change
primitive

Values must be
specified for
each new field

:

Why Add State to the Language?

- The world has abstract states – so should programming languages
 - egg -> caterpillar -> butterfly; sleep -> work -> eat -> play; hungry <-> full
- Language influences thought [Boroditsky '09]
 - Language support encourages engineers to **think** about resources
 - Better designs, better documentation, more effective reuse
- Improved library specification and verification
 - Typestate defines when you can call read()
 - Make constraints that are only implicit today, explicit
- Expressive modeling and simpler reasoning
 - Without typestate: fileResource non-**null** if File is open, **null** if closed
 - With typestate: fileResource always non-**null**
 - But only exists in the FileOpen state



Research Challenges

- Practical typechecking
 - Use **linear permissions**, not types, to reason about limited resources
 - New **abstractions** that allow sharing
 - Efficient dynamic checks: how to cast to a **unique**?
 - Gradual types → **gradual permissions**
 - Checking states of **dynamic web pages**
 - Type **parameterization**
- Efficient compilation
 - New programming model requires new representation and optimization approaches
- Concurrency
 - Leveraging **permissions** for **safe deterministic concurrency**
 - Supporting controlled **non-determinism**

Try Plaid!

The screenshot shows a web browser window with the URL `http://plaid.isri.cmu.edu:8080/plaidWebTerminal/plaid.cgi`. The page features a plaid-themed header with the text "plaid web terminal". Below the header are two navigation tabs: "Home" and "Web Terminal", with "Web Terminal" being the active tab. The main content area contains the heading "Try out Plaid in your browser!" followed by a paragraph: "Just click the run button at the bottom of the page to run your Plaid program. You can also insert some example code by using the menu on the right and experiment with it."

The interface is divided into two main sections. On the left is a code editor with a toolbar at the top showing icons for undo, redo, search, and other editing functions. The code in the editor is as follows:

```
1 state Cell {
2   method Cell getLeft() {
3     left;
4   }
5   method getRight() {
6     right;
7   }
8   val left;
9   val right;
10
11  method doPrint() {
12    printVal();
13    java.lang.System.out.print(" ");
14    val rt = this.getRight().doPrint();
15  }
16  method print() {
17    val lt = this.getLeft();
18    lt.print();
19  }
20 }
21
22 state Entry { }
```

At the bottom of the code editor, a status bar shows "Position: Ln 1, Ch 1" and "Total: Ln 253, Ch 4381". Below the code editor is a large blue "Run" button.

On the right side of the interface, there is a section titled "Select an example program:" with a dropdown menu currently set to "examples/turing.plaid" and an "Insert example" button below it. Below this is a section titled "Result:" which contains two lines of output, each enclosed in a dashed box:

```
running 1 state busy beaver:
0 1 0

running 2 state busy beaver:
0 1 1 1 1 0
```

Resource-Based Programming in Plaid

- Supports programming with resources
 - First-class abstractions for characterizing resource state
 - Promotes program understanding, precludes errors
 - Naturally safe concurrent execution
 - Practical mix of static & dynamic checking
- Opens a new area of research
 - Languages based on resources with changeable states and permissions
- Work in progress
 - Web-based implementation available, command-line compiler soon
 - Typechecker (written in Plaid) underway

<http://www.plaid-lang.org/>