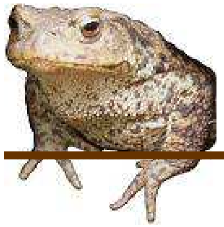
Course Wrap-up: the Past and Future of Objects

Principles of Software System Construction

Jonathan Aldrich and Charlie Garrod

Fall 2012



Outline

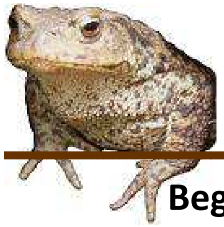
- The Beginnings of Objects
  - Simulation in Simula
- Pure OO in Smalltalk
  - Historical context, demo
- Current research in OO at CMU

How OO Began

- 1957 Kristen Nygaard begins work on computer simulation
  - Observes a challenge representing heterogeneity
- 1962 Ole-Johan Dahl joins Nygaard, work on SIMULA I

Dahl and Nygaard at the time of Simula's development

- 1966 Tony Hoare (of Hoare logic) proposes Record Classes
  - Records classes and subclasses, somewhat like Java without methods
- 1967 Nygaard and Dahl present Simula 67 [Dahl, Nygaard 1967]
  - First OO language
  - Key addition to Hoare's record calculus: ***virtual procedures***
    - Dispatch is the essence of OO [Cook, 2009]

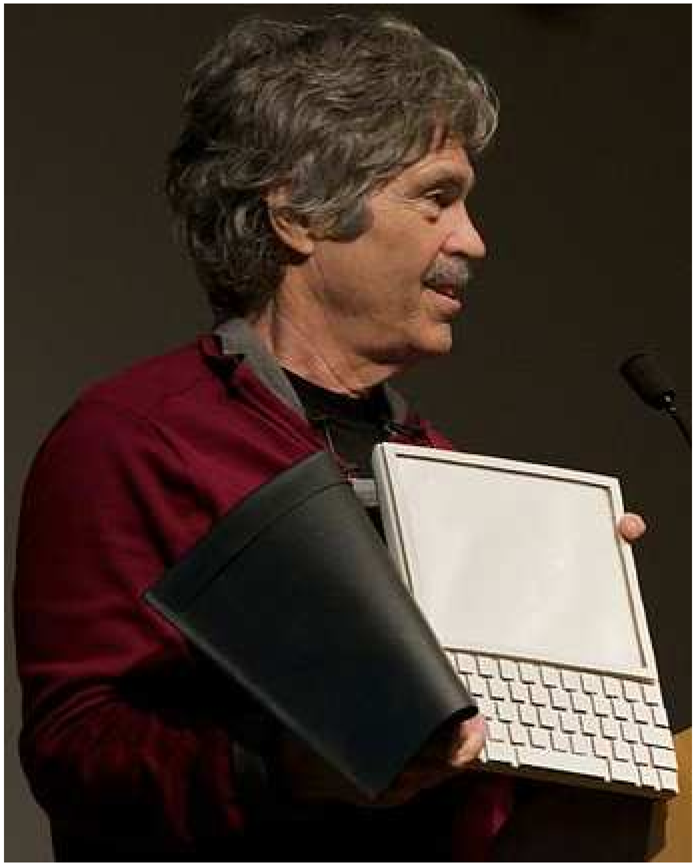


Sample Simula 67 Code

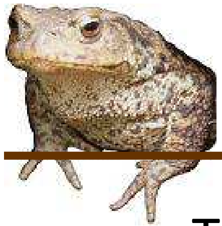
```
Begin
    Class Glyph;
        Virtual: Procedure print Is Procedure print;;
    Begin End;

    Glyph Class Char (c);
        Character c;
    Begin
        Procedure print;
            OutChar(c);
    End;

    Glyph Class Line (elements);
        Ref (Glyph) Array elements;
    Begin
        Procedure print;
        Begin
            Integer i;
            For i:= 1 Step 1 Until UpperBound (elements, 1) Do
                elements (i).print;
            OutImage;
        End;
    End;
    Ref (Glyph) rg;
    Ref (Glyph) Array rgs (1 : 4);

    ! Main program;
    rgs (1):- New Char ('A');
    rgs (2):- New Char ('b');
    rgs (3):- New Char ('b');
    rgs (4):- New Char ('a');
    rg:- New Line (rgs);
    rg.print;
End;
```

Smalltalk Context: Personal Computing

- The Dynabook at Xerox PARC: “A Personal Computer for Children of All Ages”
- Funded by US Govt (ARPA, the folks who brought you the internet) to facilitate portable maintenance documentation
- Alan Kay’s goal
  - Amplify human reach
  - Bring new ways of thinking to civilization (*CMU still pursuing this goal with computational thinking*)



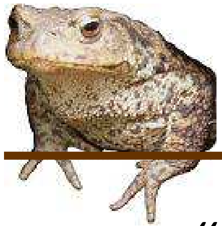
Alan Kay with a Dynabook prototype



Smalltalk

- The name
  - “Children should program in…”
  - “Programming should be a matter of…”
- Pure OO language
  - Everything is an object (including true, “hello”, and 17)
  - All computation is done via sending messages
    - 3 + 4 sends the “+” message to 3, with 4 as an argument
    - To create a Point, send the “new” message to the Point class
      - Naturally, classes are objects too!
- Garbage collected
- Reflective
  - Smalltalk is implemented (mostly) in Smalltalk
    - A few primitives in C or assembler
  - Classes, methods, objects, stack frames, etc. are all objects
    - You can look at them in the debugger, which (naturally) is itself implemented in Smalltalk

Smalltalk Demo



Smalltalk, according to Alan Kay

- “In computer terms, Smalltalk is a recursion on the notion of computer itself. Instead of dividing “computer stuff” into things each less strong than the whole—like data structures, procedures, and functions which are the usual paraphenalia of programming languages—**each Smalltalk object is a recursion of the entire possibilities of the computer**.
- “…everything we describe can be represented by the recursive composition of a single kind of behavioral building block that hides its combination of state and process inside itself and can be dealt with only through the exchange of messages.
- “Thus [Smalltalk’s] semantics are a bit like having thousands and thousands of computers all hooked together in a very fast network.”

Impact of Smalltalk and Simula

- Mac (and later Windows): inspired by Smalltalk GUI
- C++: inspired by Simula 67 concepts
- Objective C: borrows Smalltalk concepts, syntax
- Java: garbage collection, bytecode from Smalltalk
- Ruby: pure OO model almost identical to Smalltalk
  - All dynamic OO languages draw from Smalltalk to some extent

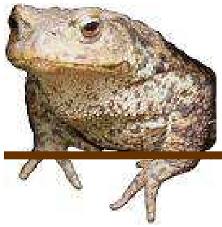
Research in Analysis, OO, Concurrency

Analysis Course – Spring 2012

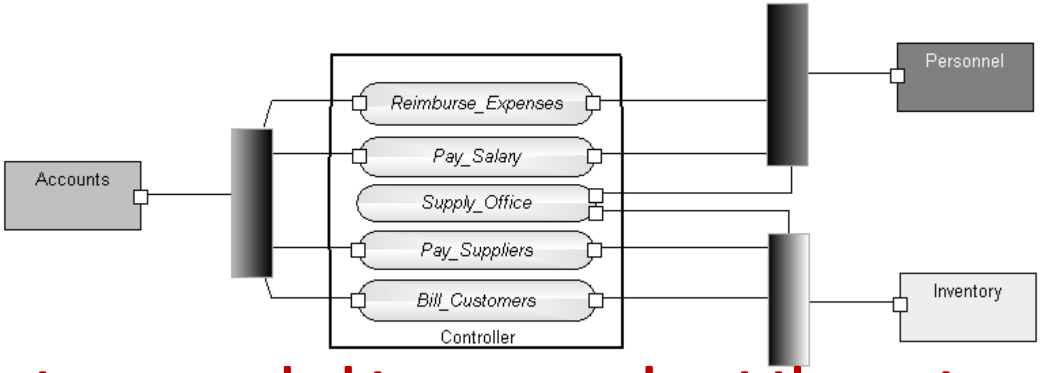
- 15-819 O: Program Analysis
  - MW 1:30-2:50pm, GHC 4101

- Topics
  - Program representation
  - Dataflow analysis and abstract interpretation (e.g. FindBugs)
  - Type- and constraint-based analysis (e.g. JSure)
  - Interprocedural analysis
  - Extended static checking (e.g. ESC/Java)
  - Concolic analysis and counterexample-guided abstract refinement

- Assignments
  - Mix of theory (reasoning about analyses) and engineering (building analyses)



Software Architecture



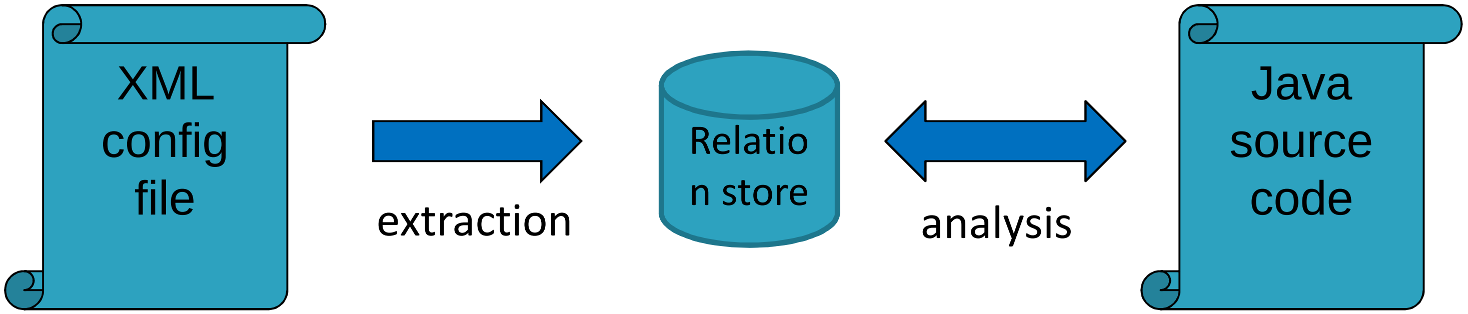
- the set of **structures needed to reason about the system**, which comprise software elements, relations among them, and properties of both – Clements et al.

- the set of **principal design decisions** made about the system – Taylor et al.

- Software architecture enables reasoning about a software system based on its design characteristics.
  - Can we leverage architecture to reason about web app security?
  - Can we link architecture to application implementation?

Configuration Files as Architecture

- Architecture already in industry frameworks
  - Framework configuration files describe structure, properties
  - Spring: web app framework
    - Describes structure, security properties of web site
  - Android framework
    - Describes event-based communication, UI flow, security properties
- Can we check these for consistency?
  - Specific tools for some frameworks—can we do it generally?
- FUSION tool at CMU/Cal Poly Pomona [C. Jaspan thesis, 2011]



XML config file

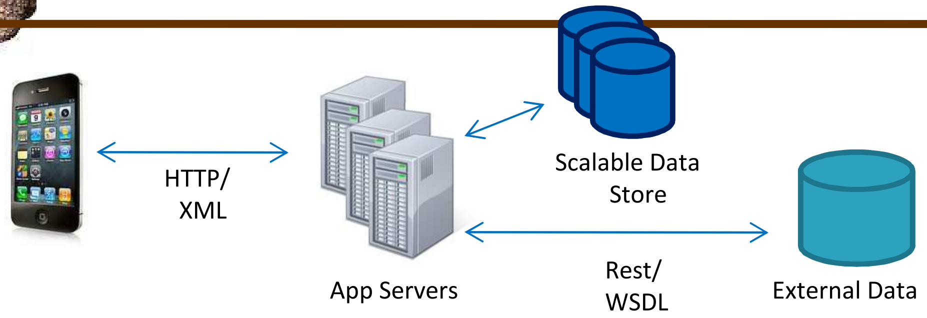
extraction

Relation store

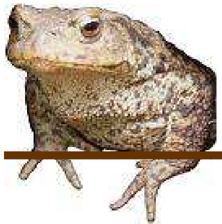
analysis

Java source code

Vision: Mobile/Web App Architecture in Code



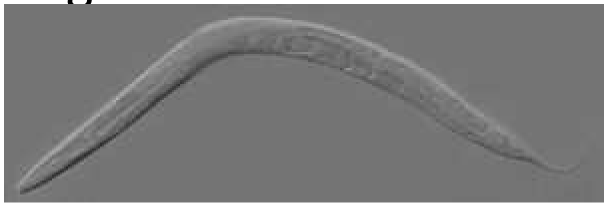
- Concept: ***Executable documentation***
  - E.g. declaring a protocol defines encoding used in components
  - Structure, redundancy, wire protocol, format, interfaces
  - Typechecking/analysis tools ensure consistency with code
- Enables analysis capabilities: attack graphs, threat models
- Challenge: making it open
  - Nothing “built-in” – implement security protocols as libraries
  - Thus libraries must also extend analysis capabilities
- End-to-end guarantee for what you implement “in the system”
  - Bridge to external systems via separate analysis tools



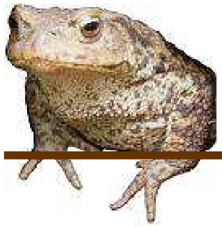
Why Ruby on Rails Works

- Flexible language syntax that supports embedded DSLs
  - But not much checking!

- Challenge: extensible language with extensible checking
- Approach: type-driven compilation and checking
  - Ability to pair a type with
    - Code generation
    - Semantic checks
  - Open source prototype: cl.oquence (OpenCL + C. Elegans)
    - Python syntax, C type system, OpenCL code generation for neuroscience [Cyrus Omar, ongoing work at CMU]

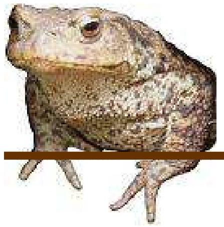


- Applications
  - Prepared SQL statements – best defense against SQL injection
  - Communication protocols



Plaid / Aeminium

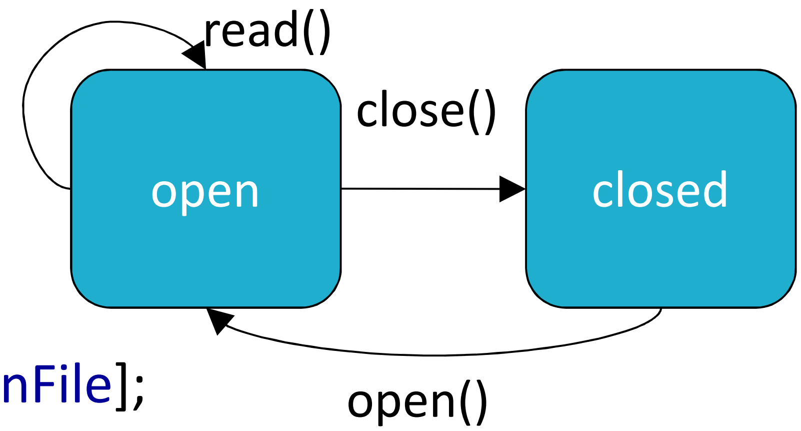
- Typestate support in the object model
- Concurrent execution by default



Typestate-Oriented Programming

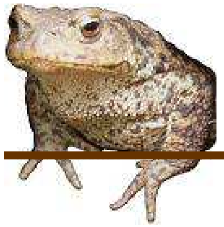
```
state File {
    val String filename;
}
state ClosedFile = File with {
    method void open() [ClosedFile>>OpenFile];
}
state OpenFile = File with {
    private val CFile fileResource;

    method int read();
    method void close() [OpenFile>>ClosedFile];
}
```

State transition



Different representation

New methods



Implementing Typestate Changes

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

Typestate change primitive – like Smalltalk *become*

Values must be specified for each new field

:

Why Typestate in the Language?

- The world has state – so should programming languages
  - egg -> caterpillar -> butterfly; sleep -> work -> eat -> play; hungry <-> full
- Language influences thought [Sapir '29, Whorf '56, Boroditsky '09]
  - Language support encourages engineers to **think** about states
    - Better designs, better documentation, more effective reuse
- Improved library specification and verification
  - Typestates define when you can call read()
  - Make constraints that are only implicit today, explicit
- Expressive modeling
  - If a field is not needed, it does not exist
  - Methods can be overridden for each state
- Simpler reasoning
  - Without state: fileResource non-**null** if File is open, **null** if closed
  - With state: fileResource always non-**null**
    - But only exists in the FileOpen state