

# Objects as Software Services

Gilad Bracha

# Bit Rot

- Bits don't rot
- It would be better if they did
- Dynamically typed languages can help us rot them

# Software requires Maintenance

- We expect software updates with
  - bug fixes
  - new features
- Dynamic update is standard practice for, e.g., OS vendors

# First, Pester

## Software Update



**New software is available for your computer.**

If you're not ready to install now, you can use the Software Update preference to check for updates later.

Install	Name	Version	Size
<input checked="" type="checkbox"/>	QuickTime	7.0.2	33.2 MB

QuickTime 7.0.2 delivers numerous important bug fixes and compatibility enhancements. This update is highly recommended for all QuickTime 7 users.

### **Important Notice to QuickTime Pro Users**

Installation of QuickTime 7 will disable the QuickTime Pro functionality in prior versions of QuickTime, such as QuickTime 5 or QuickTime 6. If you proceed with this installation, you must purchase a new QuickTime 7 Pro key to regain QuickTime Pro functionality. After installation, visit [www.apple.com/quicktime](http://www.apple.com/quicktime) to purchase a QuickTime 7 Pro key.

Restart will be required.

Quit

Install 1 Item

# Then, ask for ID

Authenticate




Software Update requires that you type your password.

Name:

Password:

► Details



# Then, legalese

## License Agreement

### QuickTime

English

APPLE COMPUTER, INC.  
SOFTWARE LICENSE AGREEMENT FOR QUICKTIME

**IMPORTANT NOTICE TO QUICKTIME PRO USERS:**  
INSTALLATION OF QUICKTIME 7 WILL DISABLE THE QUICKTIME PRO FUNCTIONALITY IN PRIOR VERSIONS OF QUICKTIME. IF YOU PROCEED WITH THIS INSTALLATION, YOU MUST PURCHASE A NEW QUICKTIME 7 PRO KEY TO REGAIN QUICKTIME PRO FUNCTIONALITY. AFTER INSTALLATION, VISIT [WWW.APPLE.COM/QUICKTIME](http://WWW.APPLE.COM/QUICKTIME) TO PURCHASE A QUICKTIME 7 PRO KEY.

PLEASE READ THIS SOFTWARE LICENSE AGREEMENT ("LICENSE") CAREFULLY BEFORE USING THE APPLE SOFTWARE. BY USING THE APPLE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, DO NOT USE THE SOFTWARE. IF YOU DO NOT AGREE TO THE TERMS OF THE LICENSE, YOU MAY RETURN THE APPLE SOFTWARE TO THE PLACE WHERE YOU OBTAINED IT FOR A REFUND. IF THE APPLE SOFTWARE WAS ACCESSED ELECTRONICALLY, CLICK "DISAGREE/DECLINE". FOR APPLE SOFTWARE INCLUDED WITH YOUR PURCHASE OF HARDWARE, YOU MUST RETURN THE ENTIRE HARDWARE/SOFTWARE PACKAGE IN ORDER TO OBTAIN A REFUND.

**IMPORTANT NOTE:** This software may be used to reproduce materials. It is licensed to you only for reproduction of non-copyrighted materials, materials in which you own the copyright, or materials you are authorized or legally permitted to reproduce. If you are uncertain about your right to copy any material, you should contact your legal advisor.

**1. General.** The software, documentation and any fonts accompanying this License whether on disk, in read only memory, on any other media or in any other form (collectively the "Apple Software") are licensed, not sold, to you by Apple Computer, Inc. ("Apple") for use only under the terms of this License, and Apple reserves all rights not expressly granted to you. The rights granted herein are limited to Apple's and its licensors' intellectual property rights in the Apple Software and do not include any other patents or intellectual property rights. You own the media on which the Apple Software is recorded but Apple and/or Apple's licensor(s) retain ownership of the Apple Software itself. The rights granted under the terms of this License include any software upgrades that replace and/or

Disagree

Agree

# Then, the coup de grace




**The new software requires that you restart your computer now.**

Click Restart to quit all applications and restart.

Shut Down

Restart

# But wait, there's more



**There are currently logged in users who may lose unsaved changes if you restart this computer.**

Restarting or shutting down the computer will quit applications in other sessions where documents have not been saved. The data will be lost. Enter an administrator's name and password and click Restart, or choose Cancel to dismiss.

Name:

Password:



# Expect better

- Make maintenance as transparent as possible
  - No questions, hassles
  - Nothing should ever boot or reboot
- Always up to date
  - Like a web app/service

# Web Apps have downsides

- System software has to be local
- UI issues
- Depend on network being:
  - Reliable
  - Fast
  - Cheap
- Still make you “reboot” - it’s called:  
*session expired*

# Software Services

- Combine advantages of web services and traditional client applications
  - Always Available (even w/o network)
  - Always Up to date
- Run locally, think globally

# Dynamically Typed Languages to the Rescue

- Lots of experience with updating code on the fly
- Much easier to do in the absence of mandatory static type system

# Self Modifying Code

- Makes people nervous
  - Lots of issues:
    - What happens if the modified code is still active (on the stack)
    - What happens to instances of modified classes
      - schema changes, representation invariants
  - Security

# Self Modifying Code

- Needs structure
  - Mirror based Reflection
- Much easier if program is quiescent
  - This does not mean waiting until the program restarts.

# When is the Program Quiescent?

- Many applications perform data synchronization over the network
- Synchronizing with server
  - Provides reliable backup, audit trail
  - Allows access from multiple devices
  - Supports collaboration

# Synchronization

- Natural point for program update
  - Applications are quiescent
  - Transition is user-visible
- Program as Data
  - Sync program as well as data



# Orthogonal Synchronization

- All persistent data is sync'ed
  - Data is persistent if it is reachable from a persistent root, and not marked transient
- Transient data is lazily recomputed after every sync
  - This can be enforced with aid of context free syntax, e.g., *transient f [initExpr]*

# Orthogonal Synchronization

- Criticisms of orthogonal persistence do not apply

# Orthogonal Synchronization

- Criticisms of orthogonal persistence do not apply
- *Data outlives Program:*

# Orthogonal Synchronization

- Criticisms of orthogonal persistence do not apply
- ~~*Data outlives Program:*~~ Program and data live as long as the service

# Orthogonal Synchronization

- Criticisms of orthogonal persistence do not apply
- ~~*Data outlives Program:*~~ Program and data live as long as the service
- *Transient data pollutes database:*

# Orthogonal Synchronization

- Criticisms of orthogonal persistence do not apply
- ~~*Data outlives Program:*~~ Program and data live as long as the service
- ~~*Transient data pollutes database:*~~ transient data is zapped at every sync

# Orthogonal Synchronization

- Criticisms of orthogonal persistence do not apply
- ~~*Data outlives Program:*~~ Program and data live as long as the service
- ~~*Transient data pollutes database:*~~ transient data is zapped at every sync
- *No cross-program interchange format:*

# Orthogonal Synchronization

- Criticisms of orthogonal persistence do not apply
- ~~*Data outlives Program:*~~ Program and data live as long as the service
- ~~*Transient data pollutes database:*~~ transient data is zapped at every sync
- ~~*No cross-program interchange format:*~~ XML

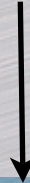


# Orthogonal Synchronization

- Efficient synchronization requires knowing what has changed
- Persistent objects should log changes
- A good language will ensure that all access is mediated by getters/setters
- When an object becomes persistent, change its setters so that they log changes

# Key Points

**Dynamic Typing**



**Hotswapping**



**Orthogonal Synchronization**



**Software Services**

# Security

- Hotswapping based upon network input is scary
- Must verify identity of server and clients
- Need strong security model for code
  - Can we achieve this w/o typed assembly language/wire format?

# Security

- Wire format must be
  - dynamically type safe
  - pointer safe
- Objects must be strongly encapsulated
  - Objects will serve as capabilities

# Security

- Mirrors act as capabilities for reflection
  - Provide single, centralized access to all reflective operations
  - Specific subsets available through particular mirrors
- Not something you can get from traditional reflective API or from popular scripting languages

# Security

- Strong sandbox - no global/static state  
(aka *No Ambient Authority*)

# Modules

- For Development & Deployment
- No static state
  - Global internet-style namespace for immutables only
- No versions

# Modules

- For Development & Deployment
- No static state
  - Global internet-style namespace for immutables only
- No versions



# Modules

- Self contained parametric namespace
  - No imports
    - imports are for localizing couplings, not for decoupling
  - All external dependencies are pluggable parameters
  - Only parameter declarations see surrounding namespace
- Explicit export of module elements

# Modules

- Are instantiated into stateful objects
- Top level module instantiation happens in namespaces with access to globals
- Parameters are objects/capabilities that determine per-module sandbox

# Speculative Syntax

```
main(platform, args) {  
    letrec  
        app = new com.foo.bar.demo(sandbox, args);  
        sandbox = platform.restrictedSandbox();  
    in app.run();  
}
```

# Modules

- For Development & Deployment
- No static state
  - Global internet-style namespace for immutables only
- No versions

# No Static

- Classes, Modules, Namespaces are values (and so are numbers, ...)
- Good for
  - Distribution
  - Security
  - Startup
  - Memory management

# Modules

- For Development & Deployment
- No static state
  - Global internet-style namespace for immutables only
- No versions

# No Versions

- Users subscribe to a software service
- Bug fixes and updates included in subscription
- Only one current version at any time
- No releases!

# No Releases?!

- This is a radical change in the development model
- Relatively easy for applications
- Hard for libraries and components
  - How do we deal with incompatibilities



# No Releases!

- Developers subscribe to pre-release libraries
- Change cycle is very rapid - days/weeks rather than months/years
- Expect libraries to morph on you daily, and be prepared to adapt
- Development model is more like open source: ***Bits Rot, deal with it***

# No Releases/Versions

- Can this work?
- As a producer of incompatible code you can find out if anyone cares
  - Do *senders-of* globally on the entire planet
  - Refactor callers
  - If anything breaks worldwide - you'll know.

# No Releases/Versions

- As a consumer of an incompatible API you can respond rapidly
- Manage transition with conditional code - and get rid of the mess the next day.
- If anything breaks worldwide - you'll know.
- Bugs that aren't caught in development can still be fixed almost immediately.

# Connections

- Mirrors
  - Self
  - Strongtalk, JDI, APT ... See OOPSLA 04
- No static
  - Scala
  - Fortress
  - E

# Connections

- Security

  - E

  - Java

- Modules

  - Jigsaw, 1991

  - Units

  - ML

  - Fortress

# Connections

- Representation independence
  - Self
- Networked Clients
  - Rich, Thin, Fat, Smart, Managed ...
    - AJAX
    - Flash
    - Avalon, XAML
    - dotmac
    - Many others ...

# Connections

- Synchronization and networked stores
  - SyncML
  - WebDAV

# Summary

Object based Encapsulation

Dynamic Typing

Security

Hotswapping

Software Services

Version free Software



# Rotting Bits for a better World

- The indestructability of bits is a hidden curse
- A model which expects incompatibility as a matter of course is better than denying change
- Dynamically typed, secure, modular languages can enable such a model