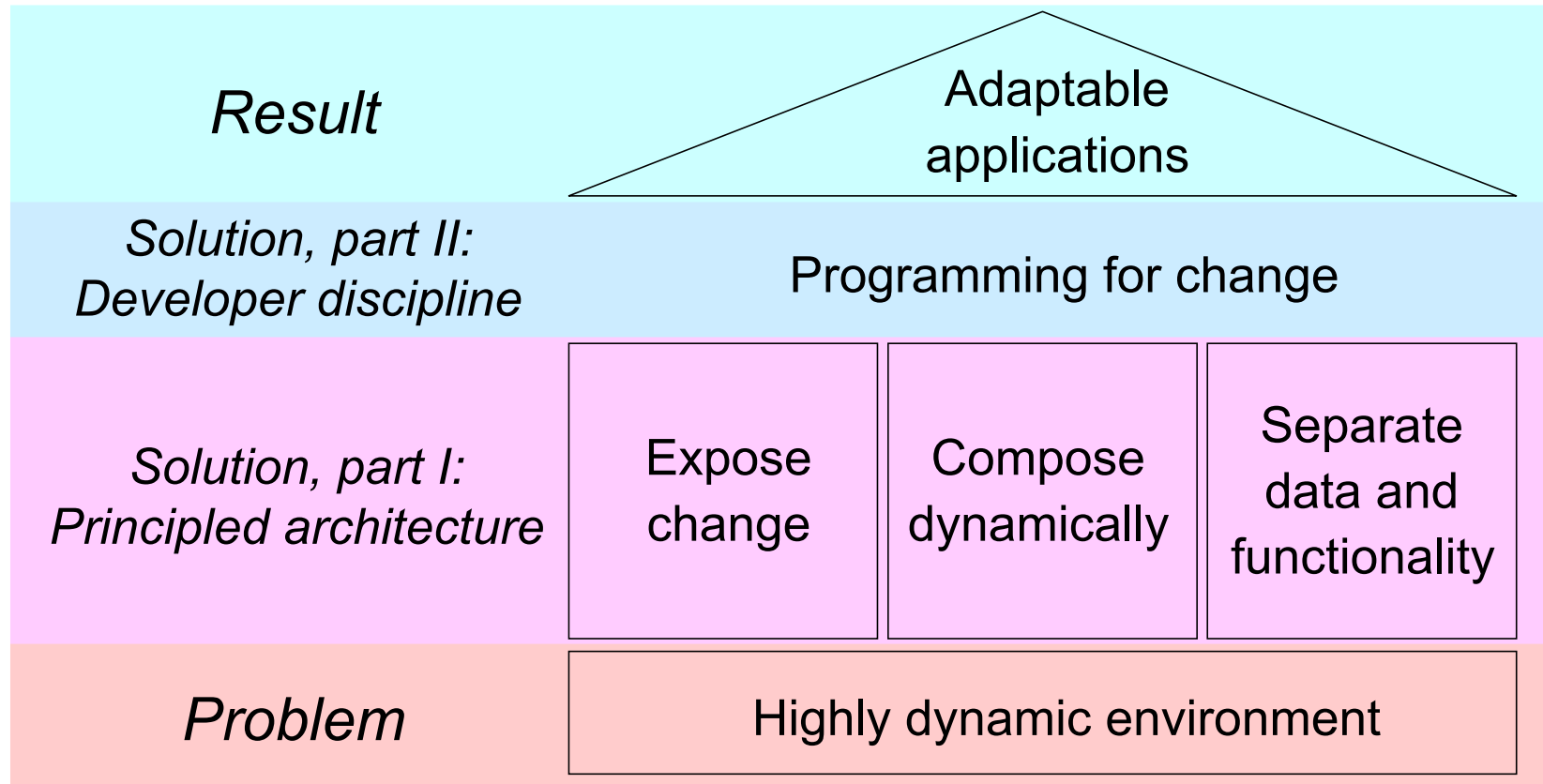


one.world — Programming for Pervasive Computing Environments

Robert Grimm,
Janet Davis, Eric Lemar, Adam MacBeth,
Steven Swanson, Daniel Cheah, Steven
Gribble, Tom Anderson, Brian Bershad,
Gaetano Borriello, David Wetherall

University of Washington

The One Slide Summary



The Vision

- Make computers usable for (computer) illiterate
 - Enabled by ubiquitous smart devices
- Implies a shift in focus
 - Away from devices and technology
 - Towards users and their tasks



one.world

The Reality

- Hardware is almost there
 - Handhelds, tablets, cars, fridges, dogs
 - Wireless networking
 - Location sensing
- Applications are missing
 - Too hard to design, build, and deploy in a giant, ad-hoc distributed system
 - Stuck with email and WWW
 - Case in point: AIBO Messenger



The Challenge

- Average hacker needs to develop applications that
 - Adapt to a changing environment
 - Work even if
 - Devices are roaming
 - Users switch devices
 - Network provides only limited services, or none at all



This Is A Systems Problem!

- Need dedicated systems support to make programmers' task feasible
- But existing approaches to building distributed systems are not suitable
 - Extend single-node programming models
 - Designed for smaller, less dynamic environments

Talk Outline

- Problem
- Principled Architecture
 - Principles
 - Architecture
- Programming for Change
- Evaluation
- Conclusion

Structure of Systems Support

- Three principles guide the design
 - Expose change
 - Compose dynamically
 - Separate data and functionality

Expose Change

- Let applications handle change, incl. failures
 - Do not hide distribution
 - Distributed file systems break single-node applications
- Provide primitives that simplify this task
 - “Checkpoint” and “restore”
 - “Move to a remote node”
 - “Find matching resource”

Compose Dynamically

- Recompose applications at runtime
 - Simplify interposition on interactions
 - Modify and add behaviors
 - Replication
 - Migration logic
 - Eschew composition through interfaces
 - Hard to interpose on
 - Hard to make extensible

Separate Data and Functionality

- Manage them separately
 - Simplify sharing, searching, translating of data
 - Relational databases do this well
 - Do not hide them behind unifying interface
 - Objects are too general and too complex
- Let them evolve independently
 - Standard bodies define data formats
 - Vendors compete on functionality
- Provide the ability to group the two
 - Preserve independent access

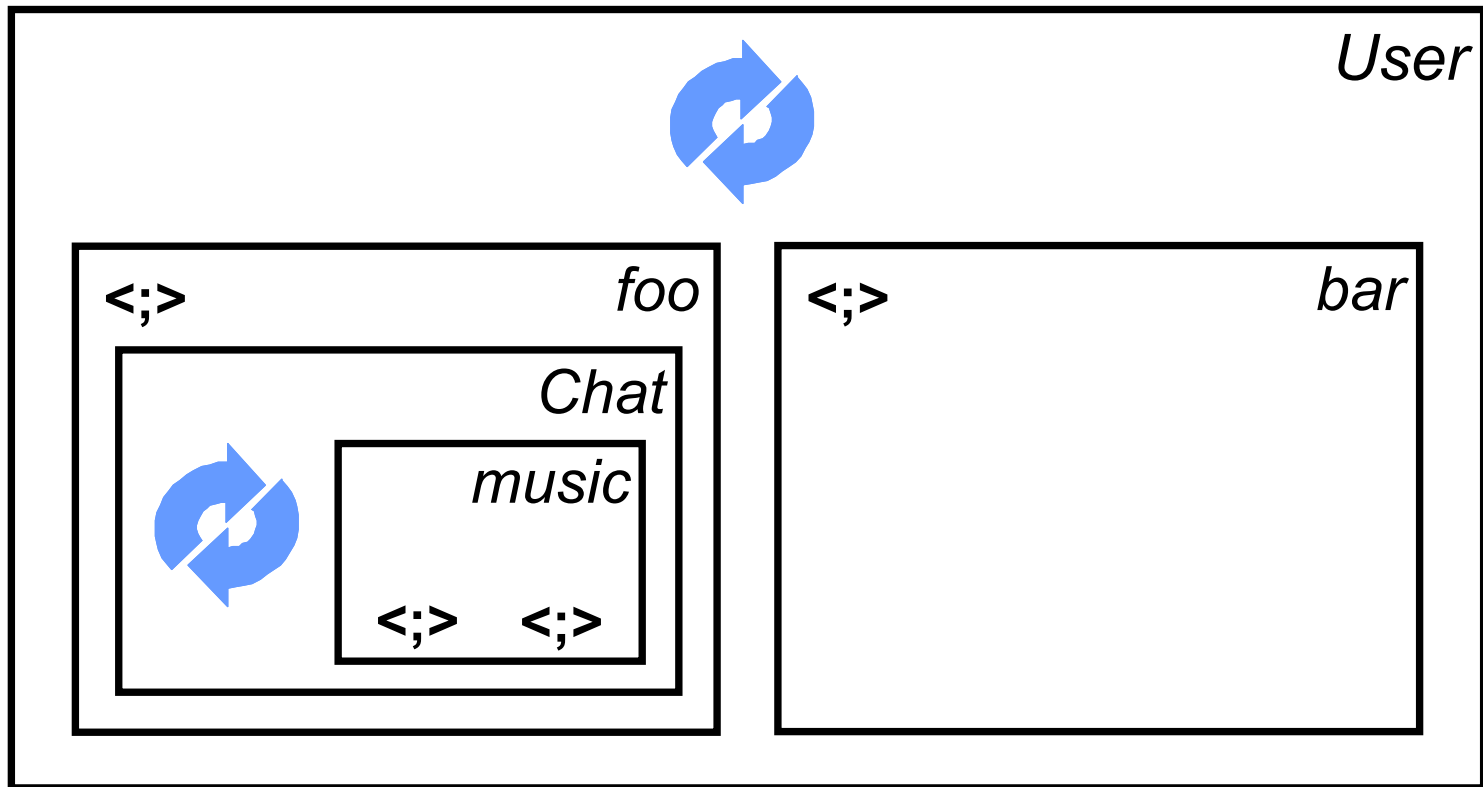
Architecture

- Each device runs one instance of *one.world*
 - Independent of other instances
 - Shared by applications
- Each such node provides a uniform platform
 - Same core abstractions and services
 - Same safe instruction set
 - Mobile code
- Written mostly in Java
 - Relies on Berkeley DB for storage management
 - Released as open source

Basic Abstractions

- Tuples
 - Represent data as self-describing records
 - Used for storage, networking, and events
- Components
 - Implement functionality
 - Interact by exchanging asynchronous events
 - Import and export event handlers
- Environments
 - Group stored data and application functionality
 - Contain tuples, components, environments

An Environment Hierarchy



 Environment  Tuple  Components

Core Services

- Checkpointing
 - Captures and restores execution state
- Migration
 - Moves or copies environment tree
- Remote event passing
 - Sends events to remote receivers, including those with an unknown location
- Replication
 - Makes data available on several nodes

Digging Deeper

- Remote event passing
- Migration
- Managing asynchrony

Remote Event Passing

- Three simple operations
 - Export event handler
 - Under identifier or arbitrary tuple
 - Resulting binding is leased
 - Send event
 - $\langle \text{Node}, \text{identifier} \rangle$ of specific handler
 - Discovery query
 - Resolve
 - Discovery query $\rightarrow \langle \text{node}, \text{identifier} \rangle^*$

Remote Event Passing

- Considerable power
 - Early and late binding
 - Selected by type of resource descriptor
 - Anycast and multicast for late binding
 - Selected by flag
- Reasonable implementation
 - Central discovery server
 - Elected from nodes on local network
 - Routes events for late binding

Migration

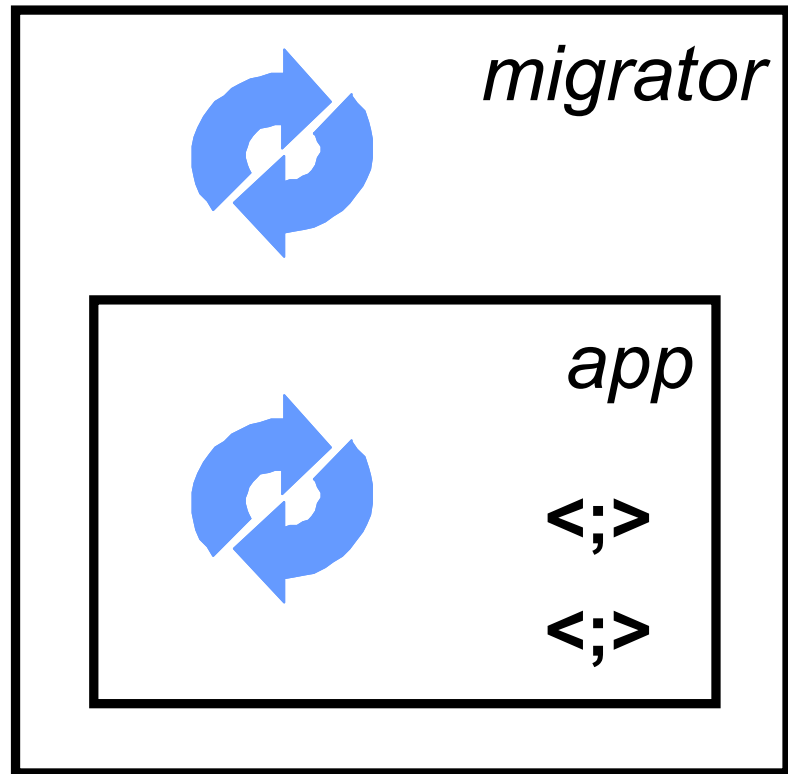
- Moves/copies an application and its data
- Affects entire environment tree
 - Tuples
 - Components
 - Environments
 - But nothing outside the tree
 - Breaks bindings to outside

Capturing Execution State

- Quiesce environments
- Serialize state
 - Components
 - <Event handler, event> queues
- Null out references to outside event handlers
 - Need to be restored by application

Composing for Migration

- Root of tree controls
 - When to migrate
 - Where to migrate to
- Compose for migration
 - Isolate migration logic in separate environment
 - Embed application in that environment



How to Structure Applications?

- Considerable uncertainty
 - Leases expire
 - Event queues fill up
 - Resources are temporarily unavailable
- Established programming styles don't scale
 - State machines

The Logic/Operation Pattern

- Logic
 - Computations that do not fail
- Operations
 - Interactions that may fail
 - Implementation includes time-out and retry code
- Composition
 - if-then-else's, loops
 - Combination is an operation as well

Talk Outline

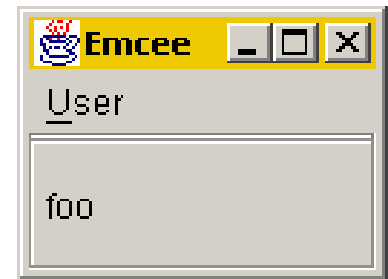
- Problem
- Principled Architecture
 - Principles
 - Architecture
- Programming for Change
- Evaluation
- Conclusion

Programming for Change

- “No application is an island!”
 - An application’s runtime environment
 - May change quite frequently
 - May be changed by others
- Examples
 - Emcee
 - Chat

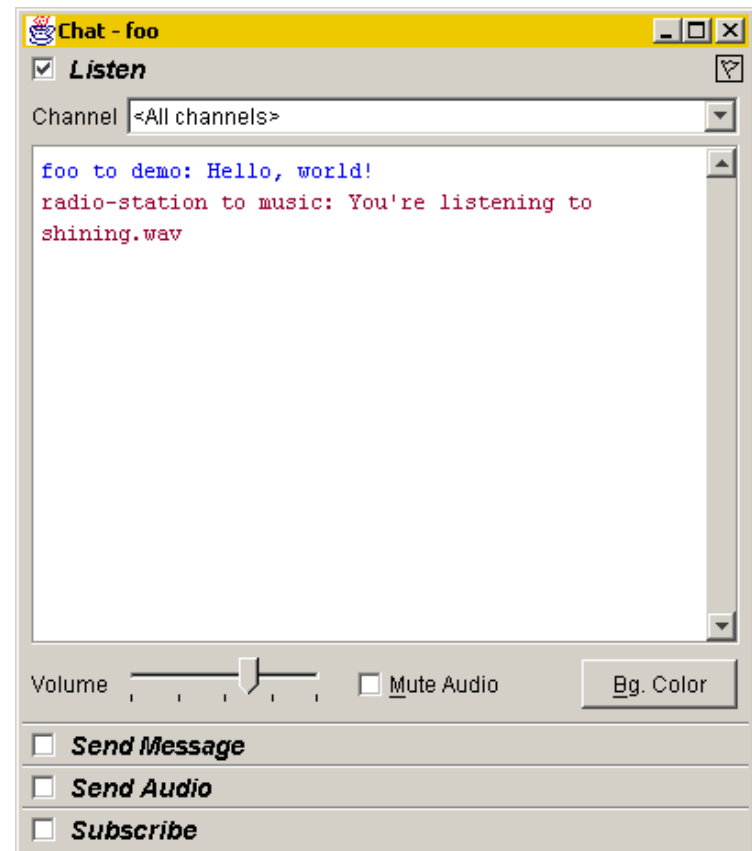
Emcee

- Manages users and their applications
 - Provides support for
 - Creating and deleting users and applications
 - Migrating users and individual applications
 - Structures environment hierarchy
 - / User / <user-name> / <application>
- Dynamically scans environments
 - Once a second for users
 - On demand for applications



Chat

- Supports text and audio messaging across channels
 - Relies on discovery for message routing
- Verifies user after activation, restoration, or migration
- Runs without audio if hardware is missing
- Silences audio channel if music has been deleted
- Checks for concurrent termination before handling events (including chat messages)



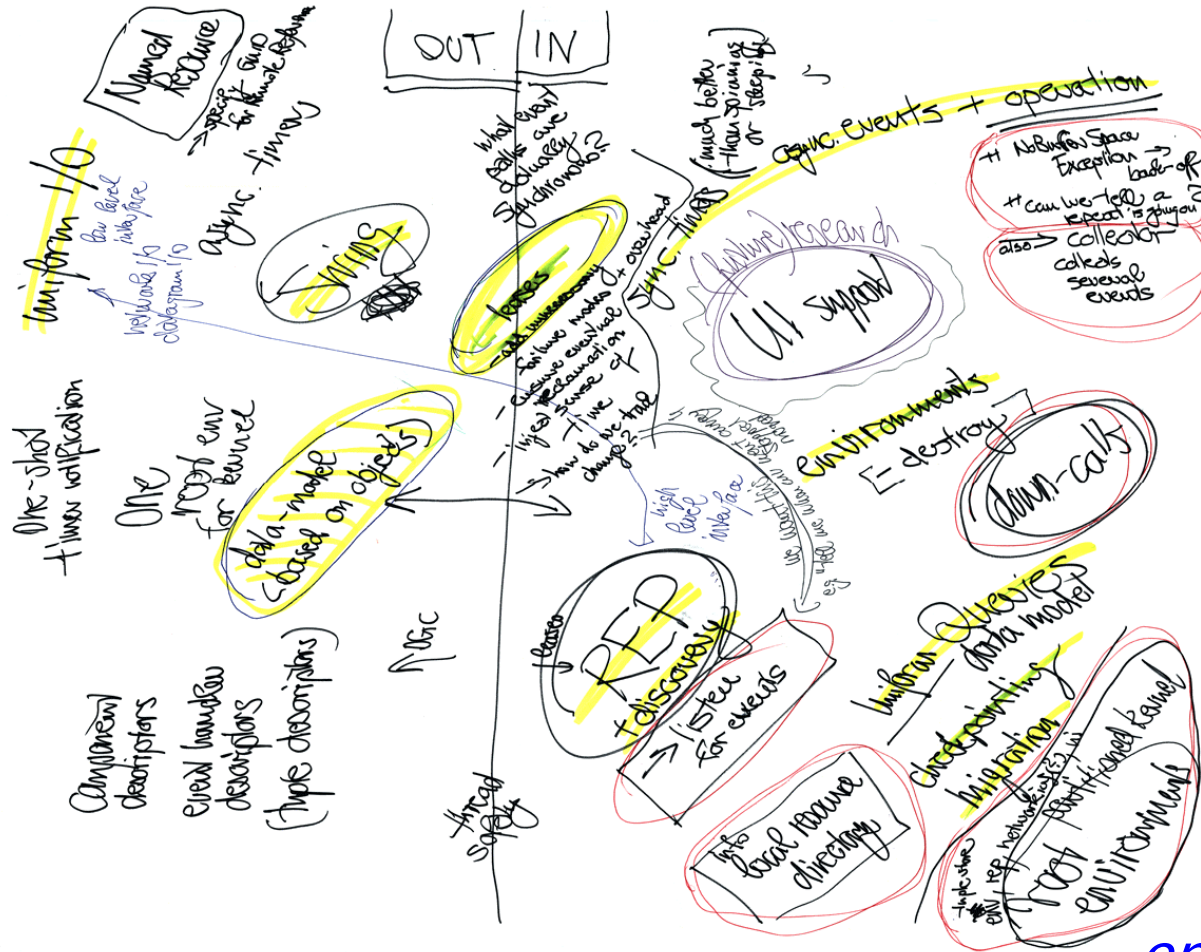
Talk Outline

- Problem
- Principled Architecture
 - Principles
 - Architecture
- Programming for Change
- Evaluation
- Conclusion

Evaluation Criteria

- Programmability
 - Design and implementation process
 - Comparative studies
- Performance
 - Microbenchmarks
 - End-to-end
- Reactivity/resilience
 - How do applications react to change?

Lessons Learned



Things That Worked

- Asynchronous events
 - + Synchronous timers
 - + Operations
- Nested environments
 - + Checkpointing
 - + Migration
- Remote event passing
 - + Integration with discovery

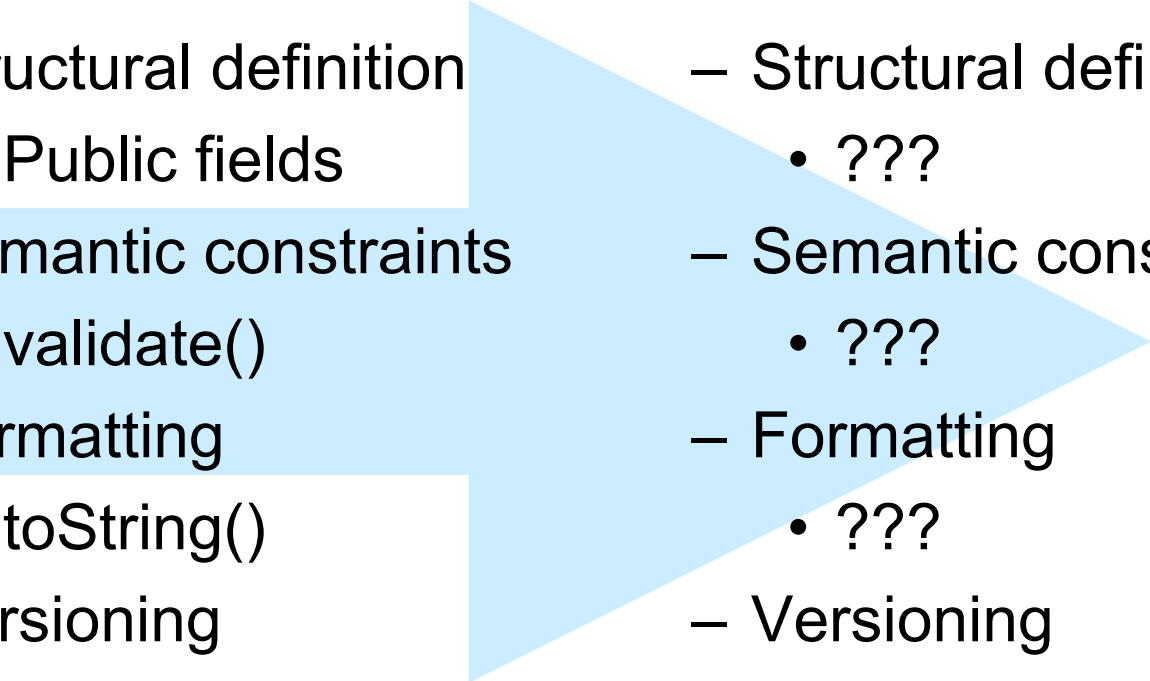
Data Model Is Contentious

- Conflicting requirements
 - Integration with programming language
 - Java objects restricted to public fields
 - Flexibility
 - Expressive enough for applications
 - Simplicity
 - Easy to exchange between applications and services

Data Model

- Programmatic model
 - Structural definition
 - Public fields
 - Semantic constraints
 - validate()
 - Formatting
 - toString()
 - Versioning
 - serialVersionUID

Data Model

- Programmatic model
 - Structural definition
 - Public fields
 - Semantic constraints
 - validate()
 - Formatting
 - toString()
 - Versioning
 - serialVersionUID
 - Declarative model
 - Structural definition
 - ???
 - Semantic constraints
 - ???
 - Formatting
 - ???
 - Versioning
 - ???
- 

XML Is Not It!

- XML fails on three accounts
 - Integration with programming language
 - DOM is unwieldy
 - Simplicity
 - XML + XML-Schema + XSL
 - Versioning still not addressed
 - Performance
 - Incredibly verbose encoding

“To a Lisp hacker, XML is s-expressions in drag.”

Bob Bane

one.world

Summary

- Pervasive applications require dedicated systems support
 - Expose change
 - Compose dynamically
 - Separate data and functionality
- *one.world* is a viable platform
 - Asynchronous events, leases
 - Nested environments and discovery's late binding
 - Tuples and components
- We can build adaptive applications on top of it

<http://one.cs.washington.edu>