Programming for Pervasive Computing Environments

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

University of Washington
The Vision

• Ubiquitous smart devices
  – Deployed in our working and living spaces
  – Some mobile, some stationary
• Powerful network services
  – Shared within the infrastructure
• Devices and services dynamically connect and coordinate with each other
  – Provide immediate access to information
  – Support users in accomplishing their tasks
The Reality

• Hardware is almost there
  – Handheld / tablet / car computers
  – Wireless networking
  – Location sensing

• Applications are missing
  – Too hard to design, build, and deploy in a giant, ad-hoc distributed system
The Challenge

- Applications need to
  - Adapt to a changing environment
  - Work even if
    - Devices are roaming
    - Users switch devices
    - Network provides only limited services, or none at all
Our Approach

• Alter way programmers think about applications
  – Programming for change
    • Remote resources: devices, services, people
    • Local resources: CPU, storage, networking
    • Stored data
• Provide systems support
  – Integrated architecture
    • Designed to support pervasive applications
    • Provides powerful primitives to help programmers
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 limiting
• Let them evolve independently
• Provide the ability to group the two
  – Preserve independent access
Our Approach Revisited

Adaptable applications

Programming for change

Expose change
Compose dynamically
Separate data and functionality
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
Basic Abstractions

• Tuples
  – Are self-describing records
• Components
  – Import and export event handlers
• Environments
  – Contain tuples, components, environments
• Leases
  – Provide timeouts when accessing unavailable resources
An Environment Hierarchy

Environment   Tuple   Components

replicator

log

<;>  <;>  <;>

app

<;>  <;>
Core Services

- Migration
  - Moves or copies environment tree
- Remote event passing
  - Sends events to remote receivers
- Replication
  - Makes data available on several nodes
- Checkpointing
  - Captures and restores execution state
Digging Deeper

• Dynamic composition
  – Basic event handling
  – Remote event passing
  – Request/monitor mechanism

• Migration
  – Design
  – Capturing execution state
  – Composing for migration
Basic Event Handling

• Asynchronous events
  – Processed by event handlers
    • Uniform interface: void handle(Event e);
  – Exported and imported by components
    • Statically declared
    • Dynamically linked and unlinked
  – Activated by environments
    • <Event handler, event> queue
    • Thread pool
Remote Event Passing

- Three simple operations
  - Export event handler
    - Under identifier or arbitrary tuple
  - Send event
    - \(<\text{Node}, \text{identifier}>\) of specific handler
    - Discovery query
  - Resolve
    - Discovery query \(\rightarrow\) \(<\text{node}, \text{identifier}>\)
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
Request/Monitor Mechanism

- Patterned after nested process model
- Provides interposition
  - An outer environment has complete control over a nested environment’s interactions with environments higher up the hierarchy
Example: Replicating Storage
Request/Monitor Mechanism

- Environments appear as components
  - Request handler accepts events
  - Monitor handler processes events
  - Kernel linked to root environment’s monitor handler
Request/Monitor Mechanism

• Simple
  – Just link one event handler to interpose

• Extensible
  – Just add code to process new events

• Flexible
  – Security: reference monitor
  – Auditing
  – Debugging
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
  - \(<\text{Event handler, event}>\) 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

one.world
Implementation

- Written mostly in Java
  - Berkeley DB for tuple storage
- Runs on Windows and Linux PCs
  - Port to iPAQ under way
- Released as open source
  - Currently version 0.4
  - Version 0.5 scheduled for June
Evaluation Criteria

• Programmability
  – Design and implementation process
  – Comparative studies

• Performance
  – Microbenchmarks
  – End-to-end

• Reactivity/resilience
  – How do applications react to change?
Evaluation So Far

- Microbenchmarks
- CSE 490dp as an experiment
  - Senior-level undergraduate course
  - Projects compared Java with one.world
    - Music search engine
    - Universal inbox
- Replication as a case study
  - Implemented on top of core
Universal Inbox

- Java & Jini & T Spaces team
  - Implemented services as single-node applications
  - Subsequently “jinified”
  - Arduously refined as network services
  - Worked around synchronous design of Java RMI
    - Remote objects may hang
    - Resulted in relatively few services
      - Each a single point of failure for all users

one.world
Universal Inbox

- *one.world* team
  - Found primitives well-matched
    - Local tuple storage, late binding
    - But components too repetitive/verbose
  - Considered appropriate logic for managing change from the beginning
  - Chose a less centralized architecture
    - Per-user services
Replication as a Case Study

- Makes extensive use of core features
  - Leases to detect disconnection
  - Nested environments for interposition
  - Components provide reconciliation policy
- Exploits collocation vs. remote interaction
  - Log migration for reconciliation
  - Remote event passing for connected operation
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 recovery code

• Composition
  – if-then-else’s, loops
  – Combination is an operation as well
Summary

• Pervasive applications require dedicated systems support
  – Expose change
  – Compose dynamically
  – Separate data and functionality

• one.world is a viable platform
  – Leases
  – Nested environments and late binding
  – Tuples and components
http://one.cs.washington.edu