System Support for Pervasive Applications

Robert Grimm
New York University
Altogether Now: The Three Questions

- What is the problem?
- What is new or different or notable?
- What are the contributions and limitations?
What is Pervasive Computing?

Pervasive computing is about making our lives simpler.

Pervasive computing aims to enable people to accomplish an increasing number of personal and professional transactions using a *new class of intelligent and portable devices.*

It gives people *convenient access* to relevant information stored on powerful networks, allowing them to easily take action *anywhere, anytime.*
And this changes everything!

**TODAY**

"The Computer"

**TOMORROW**

"The Workspace"
The Challenges of Pervasive Computing

1. The user’s focus is on the activity, not the computer
2. Devices are buried within the landscape, not vice versa
3. Tasks last days, span many devices, people, places
4. Task requirements change all the time
5. Resources degrade frequently

Adapting to constant change
The Problem

- Contemporary system services are based on the wrong set of assumptions
  - Static machines, static applications, altogether static systems
- ...and hide distribution
  - RPC, distributed objects, distributed file systems
- It is much too difficult to write seamless pervasive applications
- Users are forced to “stitch up” the seams
Goals of this work

- Develop a practical system that makes it easy to build, deploy, and use pervasive applications
  - Applications adapt, not users
- Recruit developers and users to work with the system
- Evaluate their experiences and their artifacts
- Establish a foundation for future work
1. Capture requirements for pervasive applications
2. Focus on requirements ill-served by contemporary systems
3. Define a system architecture that serves those requirements well
4. Validate with application builders
5. Iterate
Focusing on the unique requirements

- Embrace Contextual Change
  - Application context changes all the time
  - Impractical to make it a “user problem”

- Encourage Ad Hoc Composition
  - Users expect that devices and applications just plug together
  - Impractical to ask the users to do the composition

- Recognize sharing as the default
  - Applications need to easily share information across time and devices
  - Impractical to ask users to manage shared files and convert formats
The Biology Lab: An Example Application Domain

Goal: Capture, organize, and present laboratory processes
Challenges in the Digital Lab

- Embrace Contextual Change
  - Track biologists as they move through the lab and switch between experiments

- Encourage Ad Hoc Composition
  - Connect instruments to biologists using them
  - Integrate outside devices (PDAs, laptops)

- Recognize sharing as the default
  - Let biologists hand off and compare experiments
Specific Shortcomings of the Status Quo

- What happens when you try to do the digital lab with conventional systems?
  - Hard to move between machines
    - Manually log in, start applications, load data, …
  - Hard to connect devices to people
    - Computers control who uses a device, not the users
  - Hard to share data
    - File systems support only coarse-grained sharing
    - Databases are difficult to set up and administer
Jini Makes the Wrong Assumptions

- Jini (and Java RMI) require
  - A statically configured infrastructure
    - Name server, discovery server
  - A well-behaved computing environment
    - Transparent and synchronous invocations
    - No isolation between objects
    - No independence between devices
      - Distributed garbage collection
Outline

- Motivation
- Design Methodology
- **System Architecture**
- Some System Services
- Evaluation
- Conclusion
Architectural Principles

- Bias foundation services for change, ad hoc composition, and pervasive sharing
- Build specific system services and utilities in terms of these foundation services
- Employ classic user/kernel split
- Remain neutral on other issues
  - Client/server, peer-to-peer, …
one.world System Architecture

USER SPACE
- Application
- Application
- Libraries
- System Utilities

SYSTEM SERVICES
- Migration
- Discovery
- Unified I/O
- Checkpointing
- Remote Events
- Query Engine

FOUNDATION SERVICES
- Asynchronous Events
- Environments
- Tuples
- Virtual Machine

Change  Composition  Sharing
Design Rationale for Foundation Services

- Virtual machine
  - Support ad hoc composition

- Tuples (self-describing data)
  - Simplify sharing

- Environments
  - Act like address spaces, including protection
  - Store persistent data
  - Facilitate composition, checkpointing, migration

- Events
  - Make change explicit to the application
one.world: The Big Picture

Tuples & events

Migrating environments
Outline

- Motivation
- Design Methodology
- System Architecture
- Some System Services
- Evaluation
- Conclusion
System Services

Address the common requirements of pervasive applications

<table>
<thead>
<tr>
<th>Application Requires</th>
<th>System Provides</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>Query Engine</td>
</tr>
<tr>
<td>Locate</td>
<td>Discovery</td>
</tr>
<tr>
<td>Move</td>
<td>Migration</td>
</tr>
<tr>
<td>Fault-Protect</td>
<td>Checkpointing</td>
</tr>
<tr>
<td>Communicate in space</td>
<td>Remote Events</td>
</tr>
<tr>
<td>Communicate in time</td>
<td>Unified I/O</td>
</tr>
</tbody>
</table>
Discovery

- Rendezvous mechanism: Finds resources with unknown or changing location
  - In *one.world* parlance: Locates event handlers and routes events to them
- Supports coping with change and ad hoc composition
- Provides a lookup service mapping resource descriptors to event handlers
  - Self-managing: Elected discovery server
Discovery Elections

- Discovery server broadcasts heartbeat
- Per device election manager observes server
  - Calls election after two missed announcements, on server failure or shutdown
    - Each device broadcasts a score
    - Device with highest score wins
- Devices tolerate any resulting inconsistencies
  - Export resources to all servers, but look up on only one
  - Discovery server with lower score shuts itself down
- Discovery reconfigures quickly
  - We can’t stop the world to wait for perfect consistency
Migration

- Moves or copies an application and its data
  - In *one.world* parlance: Moves or copies an environment and all its contents
- Supports coping with change and ad hoc composition
  - Application deployment
- Captures a checkpoint and then moves everything
Migration Details

- Checkpointing: Capturing the execution state
  - Quiesces environments
    - Captures consistent checkpoint
  - Serializes application state and environment state
    - Builds on Java serialization
    - Limits captured state to environment tree

- Network protocol: Moving the execution state
  - Send request, metadata, stored tuples, checkpoint
    - Receiving environments can override the initial request

→ Migration works because applications already expect change
Unified I/O

- Provides a unified interface to storage and networking
  - In *one.world* parlance: Reads and writes tuples
    - Across the network
    - From/to environments
- Lets applications share data
- Converts between tuples and byte strings
  - Builds on Java serialization
  - Uses query engine to filter tuples while reading
Unified I/O Experience

- Only kernel services use network I/O
  - Remote events and discovery use network I/O
  - Applications use remote events and discovery instead

- In other words
  - Sharing in space better covered by remote events and discovery
  - Little need for unified I/O service
    - Much simpler networking layer might suffice
**The migrating, persistent counter**

- Update
- Save
- Display
- Sleep
- Move

```java
void handleEvent(e: Event) {
    if (event is arrived) {
        count++;
        checkpoint(myself);
        send(display for device, count);
        schedule-timer(in 5 mins,
                        move-on, this);
    } else if (event is move-on) {
        move(next location);
    }
}
```
Real Code

checkpoint

```java
request.handle(new EnvironmentEvent(this, null,
    EnvironmentEvent.CHECK_POINT,
    getEnvironment().getId()));
```

schedule

timer

```java
DynamicTuple moveOn = new DynamicTuple();
moveOn.set("msg", "move-on");
timer.schedule(Timer.ONCE,
    SystemUtilities.currentTimeMillis(),
    5 * Duration.MINUTE,
    this, moveOn);
```

move

```java
request.handle(new MigrationRequest(this, null,
    getEnvironment().getId(),
    "sio:///"+nextLocation()+"/",
    false));
```
Emcee

- Emcee is *one.world’s* graphical shell
  - Think ‘Finder’
- Manages users and applications
  - Builds on environment nesting
    - `/Users/<user>/<application>`
- Moves users between nodes
  - Supports both push and pull
  - Uses discovery to locate users
  - Relies on migration to move users
  - Scans environments to detect arrival
Chat

- Provides text & audio messaging
  - Location independent
    - Uses late binding for message routing
  - User and device context aware
    - Verifies user after activation, restoration, migration
  - Graceful degradation
    - Runs without audio state
  - Integrates persistent music libraries
Emcee and Chat in Action

One.Globe

Emcee

Chat

Discovery server

one.world
Outline

- Motivation
- Design Methodology
- System Architecture
- Some System Services
- Evaluation
- Conclusion
Implementation of *one.world*

- Written mostly in Java
  - Berkeley DB for tuple storage
- Runs on Linux and Windows PCs
- Released as open source
  - Currently version 0.7.1
  - 109,000 lines of code (40,000 statements)
  - 6 man years of development
  - 400 downloads of source release
Evaluation

- Key question: Is one.world good enough to build pervasive applications?

- Consider
  - **Completeness:** Can we build additional services using one.world’s primitives?
  - **Complexity:** How hard is it to write code in one.world?
  - **System performance:** Is system performance acceptable?
  - **Utility:** Have we enabled others to be successful?
Reasonable Programmer Productivity

- Does programming for pervasive applications seem harder than for conventional applications?
  - Tracked development times and tasks for Chat and Emcee
    - 3 people over 3 month period
      - Approximately 250 hours
    - $2,800 + 1,400 = 4,200$ statements
    - Productivity of 16.5 statements/hour
      - Generally measured systems range from 8 to 30 statements/hour
  - Conclusion: Nothing Herculean about coding for change, ad hoc composition, or sharing
Key concern: Can applications respond quickly to changes in context?
  - Example: Service failure
    - Avoid the “Outlook not responding” problem

Consequently, focus has been on system and application *reactivity*
  - Not on classic performance metrics (e.g., round trip message exchange)
Round Trip Message Exchange

- **Java RMI**
  - Synchronous
  - Unprotected
  - TCP connection per object
- **one.world**
  - Asynchronous
  - Protected
  - TCP connection per device
Applications and Services React Quickly to Change

- Discovery server shut down
- Discovery server fails
- Chat migrates
Other People's Experiences
The Labscape Project

- **Goal:** Seamlessly capture, organize, and present laboratory processes

- **Constraints**
  - Built by programmers, not systems experts
  - Has got to be good enough to be used everyday by everyone
    - Responsive
    - Stable
    - Robust
Labscape Architecture

Web Server

WebDav Service

State Service

XML Repository

HTTP

Proximity Service

Device Access Service

900MHz Wireless Basestation

IR Proximity Sensors and Serial Ports

Scanner

Active Badge

RF

IR

Device Access Service

900MHz Wireless Basestation

IR Proximity Sensors and Serial Ports

Scanner

Active Badge

RF

IR
Project History

- **Version 1**
  - Centralized processing, remote windowing
  - Not responsive, not robust

- **Version 2**
  - Distributed processing, code and data follow user
  - Not stable, not robust

- **Version 3**
  - Version 2 logic, but built on *one.world*
  - Responsive, stable, robust
What the Labscape People Say about *one.world*

- Development time went way down
  - From 9 man months to 4 man months
- Migration takes seconds, not minutes
  - Migration happens all the time in a laboratory
- Lab MTBF is days, not 30 minutes
- Can recover from service/device failures piecemeal, rather than through whole system restart
What They Didn’t Like

- *one.world* events are harder to use than Swing events
  - Want to write more concise event code
  - Want better support for managing asynchronous interactions

- *one.world* has its own data model and network communications
  - Want better support for interacting with legacy and web systems
Outline

- Motivation
- Design Methodology
- System Architecture
- Some System Services
- Evaluation
- Conclusion
Conclusion

Contributions

- Identified system requirements for a new style of applications
  - Pervasive applications require support for change, composition and sharing
- Developed a system that satisfies those requirements
- Validated the system internally and externally
- Foundation for further work
  - See: http://www.cs.nyu.edu/rgrimm/
Acknowledgements

- one.world team
  - Daniel Cheah, Ben Hendrickson
  - Janet Davis, Eric Lemar, Adam MacBeth, Steven Swanson
  - Tom Anderson, Brian Bershad, Gaetano Borriello, Steven Gribble, David Wetherall

- Users
  - Kaustubh Deshmukh, Liang Sun
  - Students of CSE 490dp—building distributed and pervasive applications
  - Labscape project
What Do You Think?