Technical Challenges & Infrastructure –
Designing Data & Interaction Models

Alexander Nelson
January 20, 2022

University of Arkansas - Department of Computer Science and Computer Engineering
Why Ubicomp Research?

Possible avenues for Ubicomp research:\(^1:\)

- Prototyping future systems
- Exploration of user interaction & reaction
- Obtaining datasets on which to build Ubicomp systems
- Creating experiences for public engagement/performance
- Security & Privacy

\(^1\)Much of the content for these slides comes from Chapter 2 of Ubiquitous Computing Fundamentals, Authored by Jakob Bardram & Adrian Friday
Scientific Method vs. Ubicomp Development

Must be able to adapt to necessary changes discovered during evaluation
Pre-Testing your Idea

As important as the original idea is the evaluation. Building the end solution may take time and uncover necessary changes. Alternatives:

- **Wireframe/paper prototypes** – Communicate idea directly to end users
- **Video prototypes** – Mock up solution in video
- **Rapid prototypes** – Low fidelity solutions that show it work
- **“Wizard of Oz” prototypes** – Final behavior emulated by human intervention
Designing The System
Divide between responsibilities of the user & the system

Designer needs to identify:

- knowledge the system can have about the physical world
- Knowledge the system can have about the user
- Knowledge the user has about the state of the system
- Knowledge the user has about how to change the state of the system
- What happens if this knowledge is not perfect
Questions for the designer:

1. What can be reliably sensed?
2. What can be reliably known?
3. What can be reliably inferred?
Handling Uncertainty

“Tolerance for Ignorance” – How capable the system is for imperfect information

Interaction in M2M paradigms can be modeled statistically

Interaction with users with imperfect information is difficult

Four strategies:

- **Pessimistic** – Only show information known to be correct
- **Optimistic** – Show everything as if it were correct
- **Cautious** – Present the uncertainty
- **Opportunistic** – Exploit the uncertainty (Benford et al. 2004)
What does the user actually understand about the system?

Two extreme assumptions:

1. System understands everything about the user
   - System takes actions preemptively for the user

2. System understands nothing about the user
   - System must obtain consent for any action and assent of any sensed action
Likely, there is some middle ground
System has partial understanding of user’s actions & needs
Things to consider:

- Frequency or inconvenience of user involvement
- Severity or undesirability of incorrect assumptions
- Reliability of detecting the appropriate moment & action for intercession
- Acceptability to the user of automating the behavior
User Involvement & Mental Model

These questions are specific examples of the encapsulating question:
“What do you intend for the user to understand or perceive of the system in operation?”

To feel comfortable, user must develop a mental model of the system, and appropriately identify cause→effect relationships.
Systems are composed of multiple partially connected devices that are potentially distributed geographically.
Should be available at all times.
Designer/Developer may not have access to all elements at once for maintenance.
Implications:

- Startup order cannot be assumed
- Connection events need to be handled gracefully
- System needs to be tolerant of component unavailability or failure
- System needs a data management strategy during connection lapse
- Explicit versioning in communications may be advantageous
Buffering is commonly used during network drops
Buffering in ubicomp systems should be handled strategically
e.x.

- Untimestamped GPS – misleading when network reconnects
- Time sensitive data – FIFO buffer results in long delay to new data
- Frustrated user – Likely to perform more actions, resulting in longer delays
To recover from network or component failures, existing strategies include:

- Optimistic replication of state
- Convergence on eventually consistent state
- Persistent stores or journals
- Externalizing state to a middleware platform
- Peer caches to replicate state
- Epidemic propagation of state using “gossip” protocols
Debugging embedded & distributed systems is challenging

Strategies:

- Conventional methods – e.g. log files, packet tracing
- Status protocol messages in network
- Status displays on hardware
- Diagnostic interfaces (e.g. web servers)
- Enable remote access to components (ssh, etc...)
- Externalizing of state or communication using middleware