On the Development of Personal Pervasive Environments: An Experience Report

Francisco J. Ballesteros, Gorka Guardiola, and Enrique Soriano
Laboratorio de Sistemas
Universidad Rey Juan Carlos
Madrid, Spain
{nemo, paurea, esoriano}@lsub.org
Share our experience

- Worked on two systems for pervasive comp.:
  - Plan B
  - Octopus
- Tried different paradigms
- Actually used them for everyday tasks
- Present:
  - lessons learnt
  - useful components/approaches
Plan B

- Peer to peer
- Native
- Import resources from other OS
- Export volumes, filesystem trees, e.g. voice
  - Mount volumes by description
  - Advertise new volumes to the network
  - Can control programs from any OS (exec)
- Only Plan B has access to the resources
Octopus

- Centralized namespace in the PC
- Runs hosted (or native), thanks to inferno
- Export applications/resources as filesystems (like Plan B)
- Reexported to the native OS via DAV (Upperware)
- Choose volumes by description
Things that worked well

- Global shared namespace
- High level of abstraction
- Available everywhere
- Latency aware
Global shared namespace

- In practice things are centralized, we don't introduce a single point of failure
  - Filesystem, network services
- Centralization makes the system *simpler*
  - Locate resources as they come and go
- Centralize control, decentralize data (like Google FS)
View and print

- Simple services, exported directories
- Copying to view directory opens the file
- Copying to print directory prints the file
- Simple, easy to implement, very very useful
Idle and voice

- Idle, simple estimation of activity
  - Machines know where they are (gps, ip, configuration)
  - Detects activity on authenticated machine
  - Very useful for location (where is the user, where is the user attention)
- Voice, write strings to a file, emit messages
  - Use it only for long running commands (more than 2 seconds)
  - When there is a change of state
  - If used well **not all the time**, very useful
High level of abstraction

- Export resources at the highest level of abstraction possible
- Try to export the common functionality so it is available everywhere
- Keep interactions at high level of abstraction
  - Avoid micromanaging
  - Avoid latency issues
  - Important for adaptation
  - Portable
Available everywhere

- Reexport resources to native OS
- In a way it is understood by it
  - We use filesystems (DAV)
  - Spooler devices (Drag and drop)
- Important not only to integrate services, but to export them
- They must be accessed from all OS
  - Not only export resources, but import them
Latency aware

- Latency is high in many important cases
  - Long distances (internet, limit is c)
  - Mobile environments
  - ADSL
  - Bad routers
- Protocols must be aware of latency, less RPCs, one if possible
  - 9P vs. OP
Pitfalls

- Data maintenance (spotify vs. mp3 volume)
- Infrastructure too high maintenance
  - Carrying sensors does not work well
  - Laying out the infrastructure is costly
  - Requires constant maintenance
  - Couple of times does not work, loose user's trust
Lessons learnt

- Keep it simple
- Ancient technology is your friend
  - Filesystem interfaces+conventions better than XML
- Mainstream systems should be integrated
  - I like my (mac, linux, windows)
- Latency is harmful
  - Pile up round trips, too slow: not usable
- High level of abstraction
  - Better an incomplete service everywhere than a perfect service somewhere
Open issues

- The user is always connected (false, even today)
- The user is \textbf{almost always} connected
  - Have a local cache
  - Tolerate disconnections/roaming
  - Maintain coherency (like Coda or Odyssey)
  - Still keep things centralized
Questions?

Demos, software, papers and more at: http://lsub.org