An Ambient Computing System

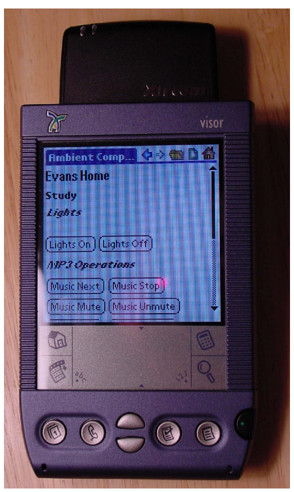
Jesse Davis

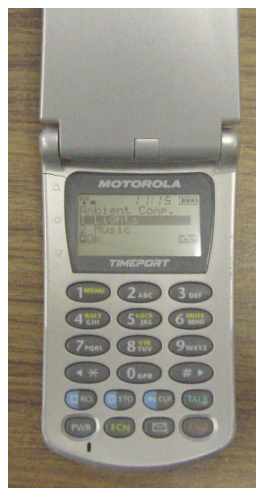
jdavis@ittc.ku.edu

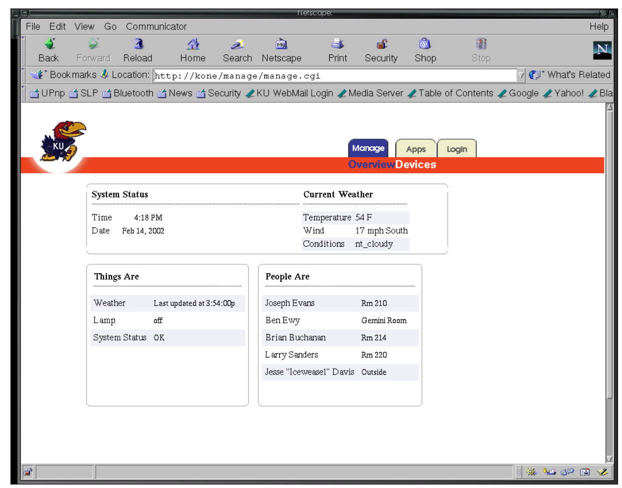
February 1, 2002

Introduction

- What is an Ambient Computing System?
  - Software framework that coordinates variety of computing and network-enabled devices to ease their use in home/business environments
  - Diverse set of computing, network and software resources can then be managed by one seamless, customizable interface
  - By easing use, system becomes pervasive (invisible to user)





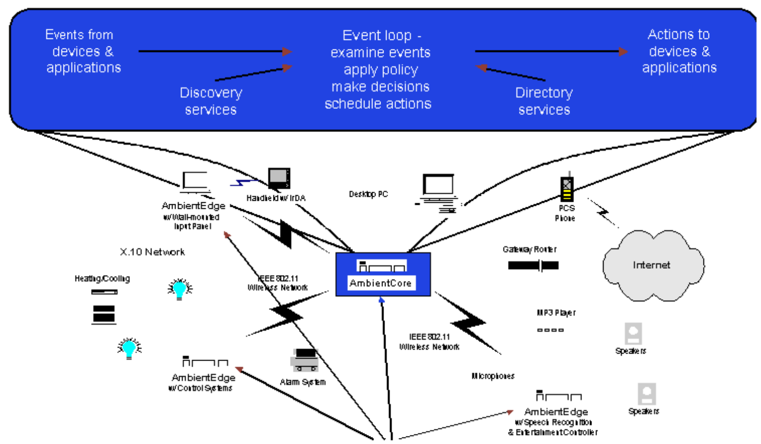


Introduction

- Why Create Such a System?
  - Traditional device control systems do not provide type of infrastructure needed
  - Enabling technologies have become commonplace in home and business
    - Wireless networks, cheap PCs, computer-enabled appliances, voice control, PDAs, cheap home control devices
  - Other projects have tried, none have delivered complete experience yet

This Approach

- Software-based, flexible easy to use computing/networking environment compatible with current and future devices
- Uses idea of MetaOS™
  - Applies traditional definition of OS to software system not dependent on hardware
  - Enables many new services by providing common way to integrate and coordinate devices and applications
  - Naturally incorporates ideas of:
    - Personalization
    - Presence
    - Permissions



Ambient Computing software

Background

- Supporting Technologies
  - LDAP – directory service
  - Speech Recognition – very natural interface
  - IEEE 802.11b
  - Bluetooth –may play roles in PDAs and mobile phones
  - X10 – residential environmental control
  - HomeRF – wireless home networking
  - SOAP (Simple Object Access Protocol) – XML protocol for RPC calls
  - SLP (Service Location Protocol)
  - UPnP (Universal Plug and Play) – service discovery protocol, distributed architecture

Background

- Other Projects/Products
  - Home Automation
    - Not enough intelligence, but can be easily used as pieces in other systems
  - Ninja (Berkeley)
    - Primarily service architecture, doesn't address user interaction
  - Oxygen (MIT)
  - .NET (Microsoft)
    - Not many details on underlying architecture yet
    - UPnP part of architecture would be useful for ambient computing systems
  - Other ambient computing systems
    - Steve Pennington's
      - purely event driven, need for interfaces, devices to send actions directly
    - ACE

Architecture

- MetaOS™ is a meta operating system
  - Like traditional definition of OS, but not dependent on low-level hardware architecture details
  - Properties:
    - Input and output control
    - Operations and job control
    - Scheduling
    - Security and multi-user support (permissions, ACLs, user identification)
    - Separation of mechanism vs. policy at device level (device driver model)
    - Distributed – not multiple processes, but distributed inputs, outputs, states integrated into single domain

Architecture

- Requirements
  - Preferences/Personalization
    - User can set preferences to determine behavior of system
    - Once tailored enough, system becomes a part of environment to user, “invisible”
  - Presence
    - System must be context-sensitive, events and commands will have different behavior dependent on current environment
    - Can enable this by using user’s location as input
  - Permissions
    - System must protect privacy of multiple users
    - Permissions, ACLs can control usage of devices, data
    - User can personalize to create levels of trust

Architecture

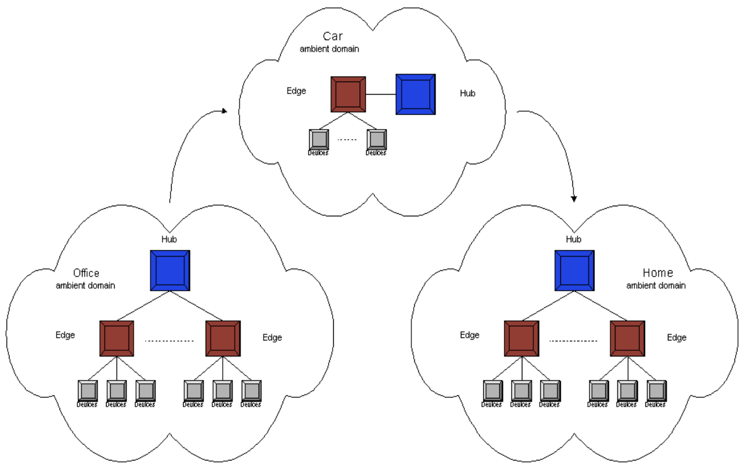
- Requirements
  - Transport Technologies
    - Must use TCP/IP
    - Physical/Data Layer: Ethernet, 802.11b, HomeRF, Bluetooth
    - X10
    - Must incorporate application-layer encryption where applicable: SSL, VPNs, Kerberos
  - Interfaces
    - XML-RPC or SOAP – allows for easy integration, .NET important
    - SLP, UPnP for service discovery
    - Abstract database interface for different DBs on backend (like JDBC)
      - LDAP first choice, quick reads, integration with Active Directory
      - More data: Oracle, etc.

Architecture

- Features of MetaOS
  - Must function on standard PC hardware, embedded system next
  - Must send messages over TCP/IP
  - Messages must be XML and sent as text
  - Common interface required, device-specific logic in device, core logic in system
  - All data structures, device capabilities, user profiles must be dynamically learned
  - Must be event driven
  - Must be very customizable, storage for user, device, other info necessary
  - Must identify/manage multiple users
  - Must protect user info and communicate over secure channels

Architecture

- Architecture Model
  - Client-server model
  - Server, or **hub** – “kernel” of MetaOS, most core logic and services here
  - Client, or **edge** – manages sending of messages from devices to hub
  - Devices send messages to hub to perform operations on system, hub sends back info necessary for device to complete task
  - Ambient domain = set of edges controlled by one hub



Architecture

- Devices
  - Division between MetaOS software and hardware that interacts with physical environment
  - Handle any number of physical inputs
  - Physical input mapped to message, message sent to system, correct response based on reply then taken
- Edge
  - “conduit” between devices and hub, sets up stream to hub
  - Responsible for instantiating devices

Architecture

- Hub
  - Routes all messages
  - Responsible for:
    - Receiving messages from devices through edges and from other hub s
    - Controlling permissions on devices, events and actions
    - Maintaining state on all devices
    - Managing connection to DB
    - Performing events and actions according to requests from devices
    - Sending messages back to devices and other hubs
- Addressing
  - Each device, edge, hub has globally unique address
  - Hierarchical, represent routing levels
  - Used like DNS entries
  - Interdomain not done yet

Architecture

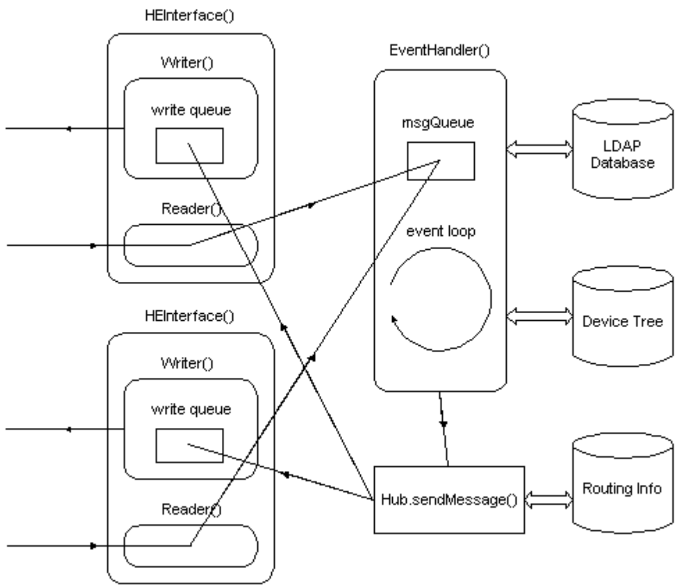
- Messages – XML
  - Registration – devices, events, actions, “todo” items, routing info
  - Event – when event has occurred
  - Action – tells device to execute given action
  - Information – query, response, add, delete, modify, device and action listing
  - Miscellaneous – ACK, NAK

Architecture

- User Management
  - User identification takes place at input points (web login, speech recognizer, other biometrics such as fingerprint readers)
  - Permissions control access, profile accessible from other domains
- Preference Management
  - Preferences stored for each user, device
  - Events, actions use preferences as arguments
  - Macros used to chain actions
- Permissions
  - Every device, event, action has identity and ACL
  - Modeled after Unix permissions

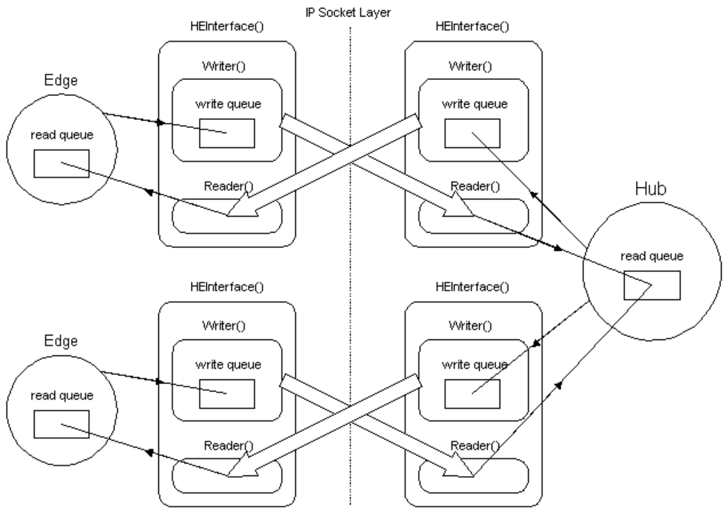
Implementation

- Languages: Java 2 1.3.1, Perl 5.6.1
- Hub – receives and sends messages
  - For each connection to listen socket, starts network interface thread for that edge and add routing info
  - Starts event handler thread
- Event handler thread – controls most of logic in hub
  - Sits in loop, waits for messages in event queue
  - For each message, traverses device tree, reads or modifies based on permissions, sends response back using hub routing table



Implementation

- Network Interface Threads
  - Manage bidirectional stream between edge and hub
  - Written to remove blocking
    - – Java has no select()
  - 2 helper threads (read/write) block while reading or writing to stream or queue
  - send() places messages in write queue, edge or hub then just wait on read queue

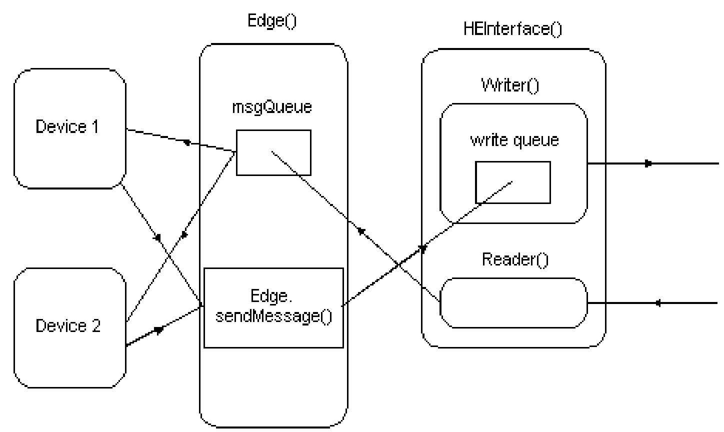


Implementation

- Messages
  - XML – easily transformed, more structured than regular text
  - Common elements - <identity>, <acl>
  - Registration messages – register objects in device tree
    - register_device, register_event, register_action, todolistitem, HEInit
  - Event message – signals hub that event occurred
  - Action message – instructs hub to execute given action with given data
  - Informational messages – read and modify info in DB
    - type attribute specifies whether operating on user or device entry
    - query, response, add, delete (value or attribute), modify
    - list – queries device tree for devices and actions
  - Miscellaneous messages
    - ACK, NAK, deviceID for SOAP interface

Implementation

- Edge – message conduit
  - Starts network interface thread, start each device in config file
  - Then just waits for messages from hub, sends to correct device
- Devices – separate mechanism and policy
  - Device-specific logic implements detection of stimuli and execution of commands for response
  - All devices derived from base class



Implementation

- Devices
  - X10Device
    - Sends commands to X10 modules
  - X10Monitor
    - Monitors log file, sends events to X10Device to send commands
  - VoiceRecognizer (mechanism)
    - Transforms spoken words into text
  - VoicePrefs (policy)
    - Executes commands based on text received
  - Framegrabber
    - Controls TV capture card, writes frames as JPGs to directory
  - MP3Player
    - Plays specified file or stream, performs common audio operations

Implementation

- Database/Directory Service
  - LDAP used for fast reads – OpenLDAP 2.0.11 used
  - MetaOS LDAP directory contains 2 sections: users and devices
  - User ID is mail address (globally unique)
  - Wrapper API for JNDI written for MetaOS
  - API base of other classes for managing data in MetaOS
    - User(), UserAdmin() – both can be used by “wizards”
  - Preferences
    - User and device preferences stored in AmbientComputingPrefs attribute
    - Multi-valued, sorted like Xresources preferences

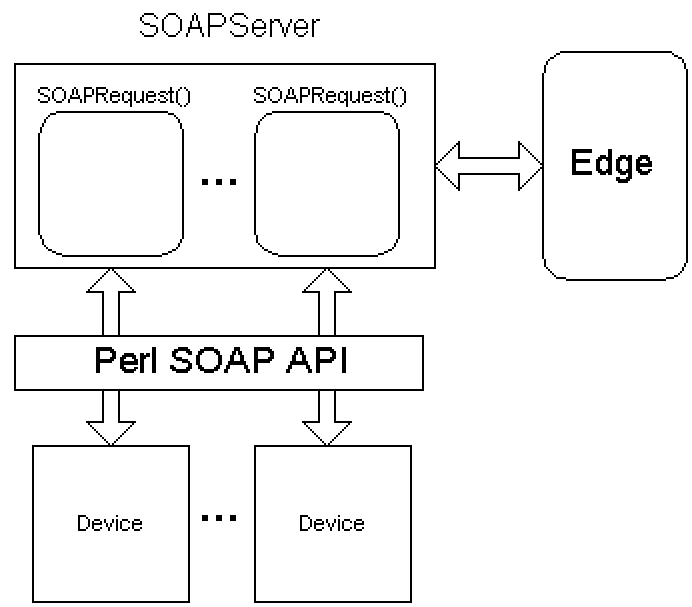
      AmbientComputingPrefs: xmms.client.linux=/usr/bin/xmms

      AmbientComputingPrefs: xmms.client.window=c:\bin\xmms.exe

      AmbientComputingPrefs: xmms.volume=90%

Implementation

- XML-RPC/SOAP Interface
  - Allows other systems/clients to send XML messages to emulate devices
    - Easy integration with other systems
    - Easy to write other APIs for communication with MetaOS
  - SOAPServer – multithreaded “gateway” device
  - Perl API module created using SOAP::Lite toolkit
    - Can use module in CGI scripts for web interfaces, etc.



Results

- Hardware/Software Requirements
  - Pentium II 233 Mhz and above
  - 128 MB RAM and above
  - Linux 2.4 series kernel or Windows 98/2000
  - Sun Java 2 SDK, version 1.3.1
  - Ethernet or 802.11b interface
  - Optional (dependent on edge configuration)
    - IBM ViaVoice™ software (text speech and recognizer)
    - MP3 players (mpg123, Winamp)
    - Sound card
    - X10 modules

Results

- Examples of messages and device setup in thesis
- Demo
  - Audio (MP3Player)
  - Video capture (FrameGrabber)
  - Home control demonstration (X10Device and X10Monitor)
  - Interfaces
    - Voice interface (VoiceRecognizer and VoicePrefs)
    - Web interface (using SOAPServer)
    - WML interface (cell phones)

Conclusions and Future Work

- List of Accomplishments
  - Designed new architecture that incorporated ideas of personalization, presence and permissions
  - Defined events, actions, devices and macros
  - Defined XML messages DTD
  - Wrote new message transport architecture to relieve blocking, scale further
  - Improved database interface
  - Improved personalization interface
  - Wrote SOAP/XML-RPC interface to MetaOS
  - Wrote Perl API module for XML-RPC interface

Conclusions and Future Work

- Conclusions
  - Application of MetaOS idea seemed a natural fit
    - Events, actions modeled as in traditional operating systems
    - Devices easier to write after adoption of device driver model
  - Message transmission architecture caused no blocking, scales well in terms of memory and load
  - Preferences architecture adequate for all objects
  - SOAP interface and Perl module proved to be useful
  - ACLs based on Unix permissions adequate

Conclusions and Future Work

- Future Work
  - SLP interface
  - Secure protocol and mechanism for hub-hub communication necessary to expand to multiple domains and for replication
  - SSL connections for network interface threads and LDAP interface
  - Extended ACLs to provide finer-grain control
  - Universal Plug and Play interface, if .NET takes off
  - JINI™ interface for MetaOS for use in Java enterprise-level environments