wasp: A platform for prototyping ubiquitous computing devices

Steve Hodges, Shahram Izadi and Simon Han

2nd June 2006

Microsoft Research Cambridge

Overview

- Motivation for *wasp*
- Novel aspects of the new approach
- Project status

Building UbiComp prototypes

- Building prototypes is valuable…
- … but hard
- Different 'levels' of prototype are progressively harder
- Focus on embedded devices
  - e.g. SenseCam
- 'Social connectivity' application domain
  - BuddyBall, Whereabouts Clock, TouchTalk

Microsoft Research Cambridge

What tools exist today?

What tools exist today?

- Hardware
  - Smart-its, Particles, Motes (x n), scatterweb, .NET CPU, Phidgets, Sun SPOT, Fleck, BTnodes, …
- Software
  - TinyOS, FreeRTOS, AwareCon, embOS, Salvo, Contiki, Tiny PLUS, uC/OS-II, …
- Each provides different pros and cons
  - Target specific application domains
  - Not enough flexibility for highest levels
  - Development time consuming

Microsoft Research Cambridge

wasp

- A <u>w</u>ireless <u>a</u>ctuator and <u>s</u>ensor <u>p</u>latform
  - New point in the prototyping space
- Provides hardware and software development support
  - New classes of application
  - Flexibility, control, performance, robustness
  - Proof-of-concept and beyond
  - New development paradigm

Microsoft® Research Cambridge

wasp hardware

- Compact, modular design
  - Physical and electrical interconnect
  - Base ARM7 processor module ~2x3cm
  - SPI variant for communications
- Largely application agnostic
  - Diverse set of modules (BT, GPRS, GPS, …)
- Power efficient, reasonable performance
  - Modules have <10uA standby
- USB for recharge and debug

Microsoft® Research Cambridge

wasp software (*wasp*-OS)

- Lightweight kernel to ease coding
  - Co-operative, event-based
  - Allows partial completion (yield) of tasks
  - Completely ANSI C (transparent, portable)
- Simple hardware abstraction layer
- Library support for h/ware modules
  - e.g. HTTP over GPRS

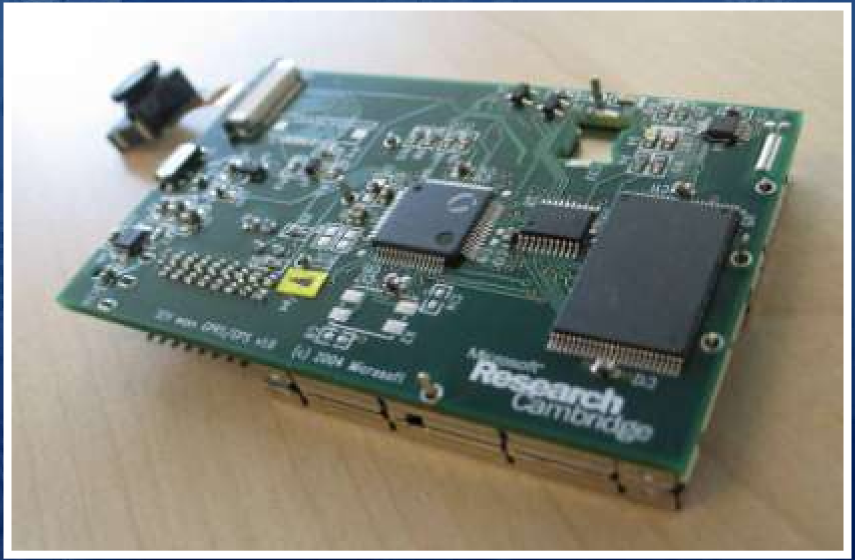
Microsoft® Research Cambridge

Debugging *wasp* applications

- Compile the ‘firmware’ under Windows
  - Run the application as Windows process
  - Leverage the power of desktop tools
- Hardware integration approaches
  - Simulate hardware on PC
  - Proxy to real hardware
- Finally re-target to embedded h/ware
  - Use USB for debug

wasp status and next steps

- Early prototype hardware complete
  - Runs *wasp*-OS
  - Basic libraries
- Much more to do
  - Re-spin with ARM7
  - Physical re-design
  - Interconnect design
  - USB proxy and debug
  - Build real applications!



Microsoft Research Cambridge