wasp: a platform for prototyping ubiquitous computing devices

Steve Hodges and Shahram Izadi, Microsoft Research Cambridge
Simon Han, UCLA Networked & Embedded Systems Laboratory
{shodges, shahrami}@microsoft.com; simonhan@gmail.com

20th April 2006

The importance of prototyping

Prototyping is a powerful way of assessing the value of ubiquitous computing applications, deciding if they warrant further development, and understanding how best to do this. Indeed, putting prototypes in the hands of 'real users' is increasingly important in assessing their potential impact and relevance. Prototypes can be developed to many different levels of sophistication, but typically early prototypes are quite basic, and as the concept is refined so too is the prototype. Experience shows that each successive level of refinement requires considerably more effort than the previous. Unfortunately, today's 'real users' have very high expectations of technology. This means that they increasingly expect even prototypes to be refined and robust, and without this they often find it hard to evaluate them fairly. Our experience shows that only when a prototype is sufficiently well developed do users see past its prototypical nature, and only then do the real insights about how it can be used become apparent. Of course, developing prototypes to this level of refinement is difficult and time-consuming. This especially applies to the development of the embedded hardware which is often integral to ubiquitous computing applications, due to the electronic and industrial design requirements that accompany software development.

wasp, a new platform for prototyping ubiquitous computing devices

We are currently developing a new embedded system development platform, called *wasp*, which facilitates the effective and efficient prototyping of both hardware and firmware for ubiquitous computing applications. *wasp* (the wireless actuator and sensor platform) is specifically designed to accelerate the development of reliable, compact, physically robust wireless prototypes with good battery lifetime. There are many other prototyping tools for ubiquitous computing hardware, but in the authors' experience, these do not meet all of the above criteria, and the resulting prototypes are typically unsuitable for full-scale user trials of the technology that they demonstrate.

From a hardware point of view, *wasp* is based around a series of small modules that can be connected together physically and electrically (via a high-speed daisy-chained serial bus) to form a particular complete embedded device. The 'base' module contains an ARM7 microcontroller[1] with a USB interface, real time clock, and power regulation (including a lithium-ion battery charger). Input, output and communications devices are incorporated via additional 'peripheral' modules – this helps manage the complexity of both the hardware and the firmware, because each peripheral to the main processor has a simple and well-defined interface. Example peripherals under development are a GSM/GPRS modem, Bluetooth, basic I/O for LEDs, servos, buzzers etc., GPS and a VGA camera.[2] If a new type of peripheral is required for a specific application, then a suitable module must be developed – but because the interface is well-defined this is a relatively simple, self-contained task.

In addition to hardware support, *wasp* also provides a powerful environment for development of embedded firmware. The basis of this is a lightweight event-based (i.e. co-operative) kernel called *wasp*-OS. In many ways, *wasp*-OS is similar to TinyOS, which has become very popular in the wireless sensor network community, although it supports a number of additional features and is written entirely in ANSI C. *wasp*-OS includes a tiered hardware abstraction layer which allows performance-sensitive applications direct access to the hardware, but also provides a reasonably high-level API to hardware such as timers and I²C, SPI and UART interfaces. Note that *wasp*-OS does not directly support protocols such as TCP/IP or Bluetooth which keeps the kernel light-weight

---

[1] We are currently concentrating on the ARM7, but *wasp* could be based around any microcontroller.

[2] Each module is one of a number of different sizes, but they are designed to connect together physically in a space-efficient manner.

and simple. Instead, these are supported through the use of peripherals that themselves have processors embedded; for example the GPRS modem module contains a microcontroller that runs a TCP/IP stack.

A prototype ubiquitous computing device built around *wasp* will therefore consist of a number of hardware 'peripheral' modules connected to a 'base' processor module. The processor on the base module runs an embedded C application in conjunction with *wasp*-OS.

Development and debugging using *wasp*

wasp also provides significant firmware development and debugging support. Since both *wasp*-OS and the code for the application itself are written entirely in ANSI C, it is possible to compile them for x86 (using Visual Studio, for example), and run them under Windows (as a single Windows process). This makes it possible to leverage the range of powerful debugging capabilities available in a PC development environment for embedded application development. Of course, a completely different binary will eventually be deployed in the embedded hardware – so there are several aspects of operation that cannot be tested in this manner (such as real-time performance). But in the authors' experience, a large number of errors in embedded system development are of a much simpler nature, and the ability to avoid a time-consuming compile-download-test cycle combined with virtually unlimited use of tools such as breakpoints[3] is incredibly powerful.

The other major issue when running a *wasp* application on the PC (rather than the embedded target) is that the peripheral modules will not be directly available. To overcome this, a different HAL must be used. The most straightforward approach is to use a HAL that re-directs peripheral access to a simulator for each peripheral. These simulators can run on the PC as separate Windows applications, or can be combined into a single Windows app. The simulation can have a UI (e.g. simulated LEDs that 'light up', or simulated push-buttons that can be clicked on), or can be completely embedded (e.g. a simulated flash memory). This approach allows embedded application development to proceed before any hardware is available. However, since many embedded peripheral devices communicate using RS232, I^2C or SPI, it is possible to connect them directly to a PC.[4] In this case, a modified HAL is used to access the real hardware peripherals, which enables a further step of validation in the debugging process.

Following migration of the application from a desktop to the embedded hardware, the *wasp* base module may itself be connected to a PC via USB. This results in a serial port connection to the *wasp* CPU. We plan to add a simple command-line interface to *wasp*-OS which will allow the real-time operation of the embedded target to be monitored and controlled very simply using this serial interface. It may be possible to extend this command-line interface to support simple scripting, which would make it easier for non-experts to control various aspects of operation. We are also investigating ways in which information may be communicated automatically between a *wasp* device and desktop applications such as Macromedia Flash.

Summary

In summary, *wasp* is a complete platform for prototyping ubiquitous computing devices effectively and efficiently. It supports a development process that naturally integrates hardware and firmware, and leverages powerful, established debugging tools and practices common to desktop application development. *wasp* is very-much work-in-progress, but we hope that over time it will prove valuable in a wide range of ubiquitous computing applications.

---

[3] Embedded microcontroller development environments often limit the number of simultaneous breakpoints.

[4] Many devices to convert USB to RS232, I^2C and SPI are readily commercially available.