

Open APIs for Embedded Systems: A Challenge for the Science of Design

Carl A. Gunter and Rajeev Alur
University of Pennsylvania

October 2003

Embedded systems are computers installed in devices such as appliances, vehicles, medical devices, and so on. They differ from computers in servers and PCs because of limits on size, power consumption, form factor, and location (*e.g.* mobility), resulting in limits on computational power, memory, and communication connectivity. Embedded systems are increasingly common; they control important devices in military, government, industrial, and, increasingly, consumer contexts. Because of the many constraints on such devices, embedded systems typically do not enjoy many of the desirable features of servers and desktop systems. In particular, such devices are typically programmed by their vendor and do not offer an open API to enable owners and users to customize the device by installing software written by third parties. Often there are benefits to customization, but applications are limited by risks to safety and security. For instance, vehicles do not currently provide open APIs for users to customize their engine functionality with third party programs, but there are users that do this to evade tax and environmental regulations. On the other hand, some embedded systems, like PDAs, offer rich open APIs and others, like cell phones, will probably follow this trend. It is interesting to speculate on which systems could benefit from open APIs. For instance, one might wonder whether there is any reason to provide an open API for a vacuum cleaner—how many different things could it do? However, there are now autonomous cleaning robots that have complex navigation systems: perhaps some important applications require custom navigation not envisioned by the robot hardware vendor. Or, the robot may need to interact in specialized ways with dynamic features of its operating environment, like sensing whether there is a meeting underway in a conference room it is expected to clean.

The challenges entailed by open APIs for embedded systems are an excellent case study for the state of the art in the science of design for software systems. Many of the issues are new twists on familiar

problems from other computers, such as the need for portability. Other issues seem more important for such devices. For instance, resource limitations are a dominant theme for embedded devices: competition for scarce resources between software from different vendors is likely to be a serious problem for a programmable embedded device. Other features seem to set embedded systems even further apart. For instance, how should one update software in a household appliance? Will such appliances be networked, and, even if they are, is there a safe way to update the software that respects the privacy wishes of its owner? The operational context of embedded systems is often much more important than that of desktop computer systems: users and physical processes are often much more directly involved in operations, and this often entails special issues for safety and security. Overall, the special challenges for open APIs on embedded systems are mainly in five areas: portability, deliverability, extensibility, flexibility, and predictability. We will illustrate these features with three brief case studies and close with some speculation on a few of the technologies that can contribute to their realization.

Household appliances are often embedded devices, and many vendors have plans for enriching the features of their appliances by computer control. Consider as an example, the idea of a *programmable microwave oven*. Currently microwaves have some limited configurability in the form of a keypad for punching in cooking times. However, in this area, the hardware vendors (like Samsung, Sharp, GE, *etc.*) would like to create more complex devices to be exploited by the software vendors (like Stouffer's, Pillsbury, *etc.*) *Multi-modal ovens*, which combine microwave and convection cooking elements are an example of the possibilities. A key challenge is how to convey more complex recipes to the oven. Sharp markets an oven in Japan that can download recipes over the Internet. Another idea is to make a microwave work like a TiVo DVR and download

recipes periodically. These could be activated with a linear barcode reader on the device read from a frozen food package. In the OpEm project at Penn (securitylab.cis.upenn.edu/open) we developed a prototype system based on two-dimensional barcodes: a food package has a smudge about a square inch in size that contains Java recipe of about one kilobyte. We were able to show that such simple programs can carry out interesting operations not currently possible and can be analyzed formally using model checking to discover features like the power usage. RFID tags are likely to provide another range of interesting programming opportunities. Consider, for instance, a washing machine that uses the RFID tags in its load to determine the best washing instructions. What kinds of standards and what software architecture will make this possible given the many vendors involved?

Payment cards come in many flavors: credit cards that allow limited borrowing, charge cards that consolidate charges to a monthly bill, debit cards that withdraw money directly from a bank account, and so on. Vendors compete to provide the kinds of cards consumers want. A *programmable payment card* is one on which a ‘second issuer’ different from a bank or payment gateway can program policy. A typical case is an enterprise that imposes its own policies on employees to whom it provides cards. For instance, they can program a card so it can only be used during regular business hours and never for items costing more than \$100. Other examples of second issuers are families that provide credit cards to children, or businesses that provide GSM cell phones to employees. In the OpEm project we developed a prototype programmable card using the Java Card running the GlobalPlatform framework for third-party programming. The principal challenge is developing a security architecture that can accommodate policies from multiple parties within the extreme limitations of a smart card.

In the near future, a collection of devices and protocols will provide location information about the growing number of people who carry them. In particular, triangulation of cell phones, GPS satellite services (especially in vehicles), and information based on DHCP (especially for WiFi), will open a new range of interesting *geo-location services*. They will also raise a wide range of privacy issues. Emerging architectures for these location-based services will ideally provide substantial individual control. This will entail a new level of user configuration for the location-reporting mobile embedded devices. Work in Penn Security Lab (securitylab.cis.upenn.edu) has been exploring languages and software architec-

tures for these rules based on those for digital rights management; these are currently aimed at licensing rights for multimedia entertainment, but could have beneficial applications in other areas such as privacy licensing. Software that respects privacy requirements will be a crucial aspect of design for embedded systems for consumers.

A range of new techniques will be needed to design software for open APIs on embedded systems. We are exploring the use of *model-based development* and *formal verification* for this domain. The need for high assurance concerning safety and security is self-evident in contexts such as medical devices and smart cards. One possibility, then, is to require the user to write applications in a domain-specific high-level modeling language that supports powerful analysis tools, and generate software from the models. Analysis using formal verification is also feasible in this domain since, compared to general-purpose software, the programs are small. For example, for programmable payment cards, we are developing a modeling language that combines extended state machines with non-monotonic logic that allows orderly resolution of decisions regarding approval of transaction requests. This allows programming of high-level policies in a modular way, and our tool reveals potential conflicts with respect to the base policies at design time. An interesting research direction concerns precise trust models that would allow automatic configuration of the security architecture. For example, we need models for determining which components of the transaction protocol should be migrated to the host computer as on-card memory becomes increasingly scarce due to dynamic addition of information needed to process policies. It is worth noting that in all these applications, we are interested in system-level guarantees, and not just properties of the software. This calls for precise specification of the requirements and modeling of the environment (for example, to prove properties of microwave recipes, we had to model how frequently the user can open and close the door).

In summary, we believe that embedded devices with open APIs will offer an exciting range of new applications. The bottleneck will be our ability to create a programming environment that provides system-level assurance. Software science technologies such as model-based development, formal verification, requirements analysis, and language-based security, should be explored in the context of this challenge.