Towards a Generic Proxy Execution Service for Small Devices

Roger Kehr, Andreas Zeidler
Darmstadt University of Technology
{kehr,az}@informatik.tu-darmstadt.de

Harald Vogt
T-Nova GmbH
vogth@tzd.telekom.de

FuseNetD Workshop Position Paper, October 1999

Abstract

Small devices such as PDAs, smartcards, or other gadgets with limited resources in terms of memory size, communication bandwidth, and battery consumption are becoming increasingly popular. Making these devices network-enabled is an effort currently undertaken by academic as well as commercial organizations.

Though these devices will grow in the near future to become even more powerful computing machines, we believe that there will always be small devices around with inherently limited resources. Since we expect that future services will federate into an infrastructure for service discovery, some of the devices will require special assistance from their environment for this integration to be successful.

We propose a general proxy execution service that offers small devices a platform for executing necessary proxy objects on a network node and give a taxonomy of the different requirements. Our concrete research prototype is a PDA which uses Infrared communication to send its device proxy to an execution service running on a network node. This proxy registers services running on the PDA in a Jini environment and acts as the gateway between the PDA and the network.

1 Motivation

The need for an open proxy execution service is motivated by the lack of features and capabilities of a large amount of devices. Figure 1 shows the design space for devices concerning these capabilities according to three different dimensions.

The first dimension denotes the amount of *processing power* a device has. Here we subsume the three main items CPU performance, memory size, and size of persistent storage, into one singular dimension, since usually all three of them grow at similar factors as devices get larger.

Along the second dimension we consider the *ability to communicate* as another relevant factor. It includes bandwidth, the underlying communication

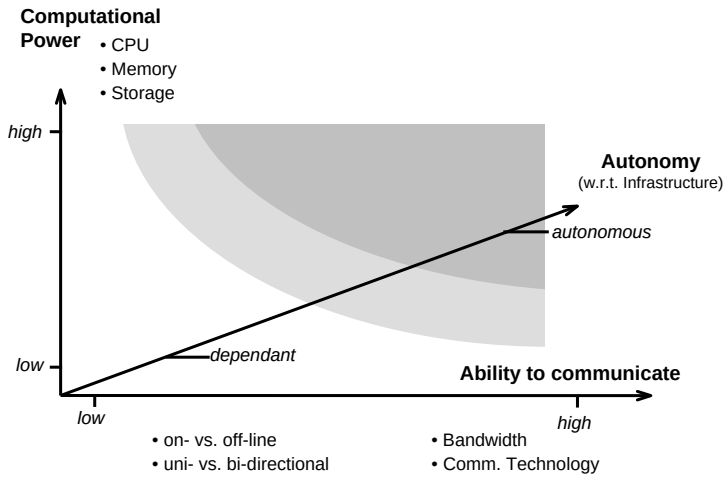

Figure 1: Design Dimensions of Small Devices

technology, and discrete issues, such as on- vs. off-line operation and support for uni- vs. bi-directional communication.

The third dimension is *autonomy*, an indication of how dependant a device is from support of the surrounding infrastructure. Obviously, autonomy usually depends on the first two dimensions and is therefore not truly orthogonal. The shaded regions are meant to indicate that the more powerful a device is along the first two dimensions, the more the device is likely to be autonomous.

As an application of our design space we classify devices of interest. This classification gives a rough impression of the character of these devices and the kind of support they need from the infrastructure for proper operation.

Sensors

Sensor devices are equipped with only minimal computing power and memory size. They have some communication facility based on technologies such as Infrared, or Bluetooth-based [Blu99] links. These devices are pure information sources and cannot be controlled or managed from outside, thus have a uni-directional link. Additionally, they may send their information in short time intervals.

Typical examples of such devices are the Infrared-emitters in the Active Badge location system [WHFG92] or sensors in medical health-care systems.

To make use of these devices, appropriate receivers must be installed at every location they are to be used at. Information processing occurs outside the devices somewhere in the infrastructure and/or at the application level.

Sensors and Actuators

In contrast to simple sensors, these devices are able to receive commands which can in turn control the device, or perform actions with the actuators, if such exist. They implement a protocol that can be used to adjust parameters such as rate of sensing, power management, etc. In our design space these devices are probably located in the light grey shaded area of Fig. 1. Many embedded systems probably fall into this category. An example of such a device is the temperature sensor described in [SA99].

Since some of these devices may not be constantly on-line, e.g. for reasons of battery consumption or aspects of mobility, interaction with these devices will mostly happen via a proxy object for that device running somewhere on a network node. This proxy knows about the on-line times of the device and exchanges data in both directions. The device proxy can be constantly on-line and accepts commands for parameter changes. The proxy waits until the next time the device is on-line and then transmits the control information to the device. It is up to the proxy to implement a synchronous or asynchronous API for its clients.

Information-Processing Devices

For devices of this category we can assume that they offer enough computational power to perform most of the tasks including the integration into an infrastructure on their own, though they still might lack enough memory or bandwidth to perform resource-intensive tasks. The dark shaded area in Fig. 1 might be the place to find devices of this category.

State-of-the-art Handhelds and PDAs are likely to belong to this category.

The Device Driver Problem

From the rough categorization given above it should be clear that different devices need different levels of support from the infrastructure, thus show different levels of autonomy. Generally, devices are bound to a particular environment with the help of software components, usually referred to as *device drivers*, that implement a protocol to map a high-level API onto native device operations.

Besides software maintenance problems, the installation of a device driver is generally unacceptable in an environment of spontaneously interacting devices. In such scenarios the device owner transports her device into an environment that is not under her own control. Consider a device, such as a smartcard, that offers several different services its owner might use in an unknown environment. Another option would be a PDA that is used as a client for different services offered in an unknown network.

In general the question arises, how a) can devices physically interact with other nodes in a network, b) can such devices be given access to services in the network, and b) how can such devices offer services on their own? These questions directly lead to the following problem domains:

Physical Link
: Different technologies are available for interconnecting devices. For the devices of interest at least the following technologies may play an important role: Proprietary Infrared, IrDA [IrD99], Bluetooth, DECT, Ethernet, etc.

Transport Protocol
: Different transport protocols are necessary to communicate with the device, ranging from TCP/IP to lower-level protocols based on the underlying link technology (e.g. IrDA session protocols IrObex, IrLan, etc.). It is important to notice that the transport protocol should ensure that the device is somehow addressable from the communication layer (e.g. by assigning an IP address)

Infrastructure Integration
: This topic essentially addresses issues concerning the kind of middleware used for service description and discovery. Examples of such service-trading middleware are Service Location Protocol [VGPK97], Jini [Wal99, Sun99], and Universal Plug and Play [UPn99]. Protocol gateways might be necessary to map between different kinds of technologies. Beyond pure service-trading, the underlying communication middleware such as RPC, CORBA, DCOM, or Java RMI adds another level of complexity for solving the integration problem.

In an open environment that allows many kinds of devices with different capabilities and technologies to seamlessly interact with each other, for each of these problem domains suitable solutions must be found. Though a general, uniform solution is unlikely to be found easily, it is obvious that an infrastructure for connecting devices built on top of pre-installed device drivers will not be flexible enough to fulfill the demands of those devices.

2 Research Outline

Beyond technological solutions we think that more fundamental research must be done to find solutions to this problem. Based on the observation of the device driver problem, we conclude that a better approach would be to invert the paradigm of device drivers. We believe that device drivers need to be present in the devices and that installation and activation of these drivers is not a task that is performed once by an administrator, but isoccurs as often as a device is linked spontaneously to a network. This activity should happen with as little user intervention as possible and not require the skills of experts.

Additionally, we want to separate the device driver into a network-resident *device proxy* and a device resident portion. Our basic claim is that devices acting in spontaneous networks need to

a) publish enough information to enable the infrastructure to load and activate the proxy from the net, e.g. a URL, or

b) carry their device proxy with them, and are able to inject them into a general execution platform in the infrastructure.

Common to both claims is the notion of *code mobility* which we consider fundamental for this kind of operation. Based on this claim, our current activities include research that addresses the following questions:

- What level of support from the infrastructure is needed by different device classes?
  If there would be a system that enables devices to inject proxies into the infrastructure, would that make manufacturing of devices much cheaper than without?

- What are the necessary protocols for establishing bindings between a device and the execution platform?
  Here the underlying transport protocols must be taken into consideration and solutions must be found to enable devices to perform efficient link level negotiation.

- How does discovery of an execution platform occur?
  This includes issues of proxy upload, instantiation and activation, something that has already been addressed for legacy devices in [ADH+99]. One could, for example, implement an execution service as a Java virtual machine running on a network node that offers a certain set of Java classes. Device proxies could then consist of serialized Java objects and an additional JAR-file containing the implementation of the device proxy.

- What does security mean in such an open environment?
  Essentially, this includes further questions such as the choice of interpreted languages, security issues (authentication, monitoring, code signing), etc. Besides the technical aspects of this question, the more general problem of security in spontaneous, *ad hoc* networking environments is fundamental and needs to be addressed.

- What Internet background infrastructure is needed to make such a service widely usable?
  This includes questions like central directory services for proxies for (very) small devices, versioning of proxies, etc.

- In what respect are such execution platforms comparable to agent platforms?
  Here it is necessary to cleanly separate both domains and on the other hand identify similarities as well.

- How does integration into a service discovery infrastructure take place?

The architecture of a widely applicable system that offers solutions for the problems outlined above is an interesting research topic.

3 Conclusion

Based on our initial observation of the device driver problem in spontaneous networks and a rough taxonomy of the design space of small devices that might become (spontaneously) networked in the near future, we have stressed the need for a general execution platform for small devices. This requires a general architecture for the design of such a service with focus on the dynamics of changes in device technologies and networks.

Our prototype, an execution service for PDAs communicating via Infrared, aims at understanding the different issues that must be solved for an execution service in a particular domain of interest. Based on these results we aim at a more general framework for supporting a variety of services and integrating middleware systems.

References

[ADH+99] Gerd Aschemann, Svetlana Domnitcheva, Peer Hasselmeyer, Roger Kehr, and Andreas Zeidler. A Framework for the Integration of Legacy Devices into a Jini Management Federation. In *Tenth IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM '99)*, October 1999.

[Blu99] Bluetooth Consortium. The Bluetooth Project Homepage. `http://www.bluetooth.com/`, 1999.

[IrD99] Infrared Data Association. `www.irda.org/`, 1999.

[SA99] Frank Stajano and Ross Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *Proceedings of 7th International Workshop on Security Protocols*, LNCS. Springer, April 1999. Available at `www.cl.cam.ac.uk/~fms27/duckling/`.

[Sun99] Sun Microsystems Inc. *Jini Architecure Specification – Revision 1.0*, January 1999.

[UPn99] Universal Plug and Play Homepage. `www.upnp.org`, 1999.

[VGPK97] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service Location Protocol (SLP). Internet RFC 2165, June 1997.

[Wal99] Jim Waldo. The Jini Architecture for Network-centric Computing. *Communications of the ACM*, 42(7):76–82, July 1999.

[WHFG92] R. Want, A. Hopper, V Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1), 1992.